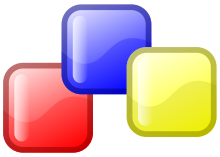


wxWidgets

Cross-Platform GUI Library

3.1.0 Reference Manual

February 8, 2015

Contents

Chapter 1

Documentation

Author

Julian Smart, Vadim Zeitlin, Robin Dunn, Stefan Csomor, Bryan Petty, Francesco Montorsi, Robert Roebling et al

Date

November 19, 2014

Welcome to wxWidgets, a stable and powerful open source framework for developing native cross-platform GUI applications in C++!

If you are new to wxWidgets, please start with the introduction and follow with the programming guides, with maybe a look at the samples as you go. If you are already familiar with wxWidgets, please read about the changes in the latest version compared to 2.8 series. And you can also follow the links in the reference section or jump directly to the `alphabetical list of classes` to find out more about the topic you are interested in.

1.1 User Manual

- Introduction
- Programming Guides
- Library List
- Overview of Available Classes
- Changes Since wxWidgets 2.8

1.2 Reference

- Class List by Category
- Functions and Macros by Category
- Constants

1.3 Other Information

- Samples Overview
- Utilities Overview
- Translations to Other Languages
- Cross-Platform Development Tips
- Copyrights and Licenses

Chapter 2

Overview of Available Classes

This page contains a summarized listing of classes, please see the Class List by Category page for a full listing by category or the `full list of classes` in alphabetical order.

For a more visual approach, see the screenshots page.

2.1 Basic Windows

The following are the most important window classes

- wxWindow: base class for all windows and controls
- wxControl: base class (mostly) for native controls/widgets
- wxPanel: window which can smartly manage child windows
- wxScrolledWindow: Window with automatically managed scrollbars (see wxScrolled)
- wxTopLevelWindow: Any top level window, dialog or frame

2.2 Window Layout

There are two different systems for laying out windows (and dialogs in particular). One is based upon so-called sizers and it requires less typing, thinking and calculating and will in almost all cases produce dialogs looking equally well on all platforms, the other is based on so-called constraints and is deprecated, though still available.

Related Overviews: Sizers Overview

These are the classes relevant to sizer-based layout:

- wxSizer: Abstract base class
- wxBoxSizer: A sizer for laying out windows in a row or column
- wxGridSizer: A sizer for laying out windows in a grid with all fields having the same size
- wxFlexGridSizer: A sizer for laying out windows in a flexible grid
- wxGridBagSizer: Another grid sizer that lets you specify the cell an item is in, and items can span rows and/or columns.
- wxStaticBoxSizer: Same as wxBoxSizer, but with a surrounding static box
- wxWrapSizer: A sizer which wraps its child controls as size permits

Other layout classes:

- wxLayoutAlgorithm: An alternative window layout facility

2.3 Managed Windows

There are several types of window that are directly controlled by the window manager (such as MS Windows, or the Motif Window Manager). Frames and dialogs are similar in wxWidgets, but only dialogs may be modal.

Related Overviews: Common Dialogs

- wxDialog: Dialog box
- wxFrame: Normal frame
- wxMDIChildFrame: MDI child frame
- wxMDIParentFrame: MDI parent frame
- wxMiniFrame: A frame with a small title bar
- wxPopupWindow: A toplevel window without decorations, e.g. for a combobox pop-up
- wxPropertySheetDialog: Property sheet dialog
- wxSplashScreen: Splash screen class
- wxTipWindow: Shows text in a small window
- wxWizard: A wizard dialog

2.4 Menus

- wxMenu: Displays a series of menu items for selection
- wxMenuBar: Contains a series of menus for use with a frame
- wxMenuItem: Represents a single menu item

2.5 Controls

Typically, these are small windows which provide interaction with the user. Controls that are not static can have wxValidator associated with them.

- wxAnimationCtrl: A control to display an animation
- wxControl: The base class for controls
- wxBitmapButton: Push button control, displaying a bitmap
- wxBitmapComboBox: A combobox with bitmaps next to text items
- wxBitmapToggleButton: A toggle button with bitmaps.
- wxButton: Push button control, displaying text
- wxCalendarCtrl: Control showing an entire calendar month
- wxCheckBox: Checkbox control

- wxCheckListBox: A listbox with a checkbox to the left of each item
- wxChoice: Choice control (a combobox without the editable area)
- wxCollapsiblePane: A panel which can be shown/hidden by the user
- wxComboBox: A choice with an editable area
- wxComboCtrl: A combobox with application defined popup
- wxDataViewCtrl: A control to display tabular or tree like data
- wxDataViewTreeCtrl: A specialized wxDataViewCtrl with a wxTreeCtrl-like API
- wxDataViewListCtrl: A specialized wxDataViewCtrl for displaying and editing simple tables.
- wxEditableListBox: A listbox with editable items.
- wxFileCtrl: A control for selecting a file. Useful for custom file dialogs.
- wxGauge: A control to represent a varying quantity, such as time remaining
- wxGenericDirCtrl: A control for displaying a directory tree
- wxGrid: A control to display spread-sheet like data in tabular form
- wxHeaderCtrl: a small control to display the top header of tabular data
- wxHtmlListBox: An abstract class for creating listboxes showing HTML content
- wxHyperlinkCtrl: A static text which opens an URL when clicked
- wxListBox: A list of strings for single or multiple selection
- wxListCtrl: A control for displaying lists of strings and/or icons, plus a multicolumn report view
- wxListView: A simpler interface (façade) for wxListCtrl in report mode
- wxNotebook: A notebook class
- wxOwnerDrawnComboBox: A combobox with owner-drawn list items
- wxPropertyGrid: A complex control to display hierachical, editable information
- wxRadioBox: A group of radio buttons
- wxRadioButton: A round button to be used with others in a mutually exclusive way
- wxRearrangeCtrl: A control allowing the user to rearrange a list of items.
- wxRichTextCtrl: Generic rich text editing control
- wxSimpleHtmlListBox: A listbox showing HTML content
- wxStaticBox: A static, or group box for visually grouping related controls
- wxScrollBar: Scrollbar control
- wxSearchCtrl: A text input control used to initiate a search
- wxSpinButton: A spin or 'up-down' control
- wxSpinCtrl: A spin control - i.e. spin button and text control displaying an integer
- wxSpinCtrlDouble: A spin control - i.e. spin button and text control displaying a real number
- wxStaticText: One or more lines of non-editable text
- wxTextCtrl: Single or multiline text editing control
- wxToggleButton: A button which stays pressed when clicked by user.

- wxTreeCtrl: Tree (hierarchy) control
- wxTreeListCtrl: Multi-column tree control with simple interface
- wxStaticBitmap: A control to display a bitmap
- wxStyledTextCtrl: A wxWidgets implementation of the Scintilla source code editing component for plain text editing.
- wxSlider: A slider that can be dragged by the user
- wxVListBox: A listbox supporting variable height rows

2.6 Validators

These are the window validators, used for filtering and validating user input.

Related Overviews: wxValidator Overview

- wxValidator: Base validator class
- wxTextValidator: Text control validator class
- wxGenericValidator: Generic control validator class
- wxIntegerValidator: Text control validator class for integer numbers
- wxFloatingPointValidator: Text control validator class for floating point numbers

2.7 Picker Controls

These controls provide the user with the possibility to choose something (file or directory, font or colour, ...) directly from the window containing them.

- wxColourPickerCtrl: A control which allows the user to choose a colour
- wxDirPickerCtrl: A control which allows the user to choose a directory
- wxFilePickerCtrl: A control which allows the user to choose a file
- wxFontPickerCtrl: A control which allows the user to choose a font
- wxDatePickerCtrl: Small date picker control

2.8 Miscellaneous Windows

The following are a variety of classes that are derived from wxWindow.

- wxCollapsiblePane: A panel which can be shown/hidden by the user
- wxPanel: A window whose colour changes according to current user settings
- wxScrolledWindow: Window with automatically managed scrollbars (see wxScrolled)
- wxHScrolledWindow: As wxScrolledWindow but supports columns of variable widths
- wxVScrolledWindow: As wxScrolledWindow but supports rows of variable heights
- wxHVScrolledWindow: As wxScrolledWindow but supports scroll units of variable sizes.

- wxGrid: A grid (table) window
- wxInfoBar: An information bar usually shown on top of the main window.
- wxRichToolTip: A customizable tooltip.
- wxSplitterWindow: Window which can be split vertically or horizontally
- wxStatusBar: Implements the status bar on a frame
- wxToolBar: Toolbar class
- wxNotebook: Notebook class
- wxListbook: Similar to notebook but using list control
- wxChoicebook: Similar to notebook but using choice control
- wxTreebook: Similar to notebook but using tree control
- wxSashWindow: Window with four optional sashes that can be dragged
- wxSashLayoutWindow: Window that can be involved in an IDE-like layout arrangement
- wxSimplebook: Another book control but one allowing only the program, not the user, to change its current page.
- wxWizardPage: A base class for the page in wizard dialog.
- wxWizardPageSimple: A page in wizard dialog.
- wxCustomBackgroundWindow: A window allowing to set a custom bitmap.

2.9 Window Docking (wxAUI)

wxAUI is a set classes for writing a customizable application interface with built-in docking, floatable panes and a flexible MDI-like interface.

Related Overviews: wxAUI Overview

- wxAuiManager: The central class for managing the interface
- wxAuiNotebook: A replacement notebook class with extra features
- wxAuiPaneInfo: Describes a single pane
- wxAuiDockArt: Art and metrics provider for customizing the docking user interface
- wxAuiTabArt: Art and metrics provider for customizing the notebook user interface

2.10 Common Dialogs

Common dialogs are ready-made dialog classes which are frequently used in an application.

Related Overviews: Common Dialogs

- wxDialog: Base class for common dialogs
- wxColourDialog: Colour chooser dialog
- wxDirDialog: Directory selector dialog
- wxFileDialog: File selector dialog

- wxFindReplaceDialog: Text search/replace dialog
- wxFontDialog: Font chooser dialog
- wxMessageDialog: Simple message box dialog
- wxMultiChoiceDialog: Dialog to get one or more selections from a list
- wxPageSetupDialog: Standard page setup dialog
- wxPasswordEntryDialog: Dialog to get a password from the user
- wxPrintDialog: Standard print dialog
- wxProgressDialog: Progress indication dialog
- wxRearrangeDialog: Dialog allowing the user to rearrange a list of items.
- wxRichTextFormattingDialog: A dialog for formatting the content of a wxRichTextCtrl
- wxRichMessageDialog: Nicer message box dialog
- wxSingleChoiceDialog: Dialog to get a single selection from a list and return the string
- wxSymbolPickerDialog: Symbol selector dialog
- wxTextEntryDialog: Dialog to get a single line of text from the user
- wxWizard: A wizard dialog.

2.11 HTML

wxWidgets provides a set of classes to display text in HTML format. These classes include a help system based on the HTML widget.

- wxHtmlHelpController: HTML help controller class
- wxHtmlWindow: HTML window class
- wxHtmlEasyPrinting: Simple class for printing HTML
- wxHtmlPrintout: Generic HTML wxPrintout class
- wxHtmlParser: Generic HTML parser class
- wxHtmlTagHandler: HTML tag handler, pluginable into wxHtmlParser
- wxHtmlWinParser: HTML parser class for wxHtmlWindow
- wxHtmlWinTagHandler: HTML tag handler, pluginable into wxHtmlWinParser

2.12 Device Contexts

Device contexts are surfaces that may be drawn on, and provide an abstraction that allows parameterisation of your drawing code by passing different device contexts.

Related Overviews: Device Contexts

- wxAutoBufferedPaintDC: A helper device context for double buffered drawing inside **OnPaint()**.
- wxBufferedDC: A helper device context for double buffered drawing.
- wxBufferedPaintDC: A helper device context for double buffered drawing inside **OnPaint()**.

- wxClientDC: A device context to access the client area outside **OnPaint()** events
- wxPaintDC: A device context to access the client area inside **OnPaint()** events
- wxWindowDC: A device context to access the non-client area
- wxScreenDC: A device context to access the entire screen
- wxDC: The device context base class
- wxMemoryDC: A device context for drawing into bitmaps
- wxMetafileDC: A device context for drawing into metafiles
- wxMirrorDC: A proxy device context allowing for simple mirroring.
- wxPostScriptDC: A device context for drawing into PostScript files
- wxPrinterDC: A device context for drawing to printers

2.13 Graphics Context classes

These classes are related to drawing using a new vector based drawing API and are based on the modern drawing backend GDI+, CoreGraphics and Cairo.

- wxGraphicsRenderer: Represents a drawing engine.
- wxGraphicsContext: Represents a graphics context currently being drawn on.
- wxGraphicsBrush: Brush for drawing into a wxGraphicsContext
- wxGraphicsPen: Pen for drawing into a wxGraphicsContext
- wxGraphicsFont: Font for drawing text on a wxGraphicsContext
- wxGraphicsMatrix: Represents an affine matrix for drawing transformation
- wxGraphicsPath: Represents a path for drawing

2.14 Graphics Device Interface

These classes are related to drawing on device contexts and windows.

- wxColour: Represents the red, blue and green elements of a colour
- wxDCClipper: Wraps the operations of setting and destroying the clipping region
- wxBrush: Used for filling areas on a device context
- wxBrushList: The list of previously-created brushes
- wxFont: Represents fonts
- wxFontList: The list of previously-created fonts
- wxPen: Used for drawing lines on a device context
- wxPenList: The list of previously-created pens
- wxPalette: Represents a table of indices into RGB values
- wxRegion: Represents a simple or complex region on a window or device context
- wxRendererNative: Abstracts high-level drawing primitives

2.15 Image and bitmap classes

These classes represent images and bitmap in various formats and ways to access and create them.

Related Overviews: Bitmaps and Icons

- wxAnimation: Represents an animation
- wxBitmap: Represents a platform dependent bitmap
- wxBitmapHandler: Class for loading a saving a wxBitmap in a specific format
- wxCursor: A small, transparent bitmap representing the cursor
- wxIcon: A small, transparent bitmap for assigning to frames and drawing on device contexts
- wxImage: A platform-independent image class
- wxImageHandler: Class for loading a saving a wxImage in a specific format
- wxImageList: A list of images, used with some controls
- wxMask: Represents a mask to be used with a bitmap for transparent drawing
- wxMemoryDC: A device context for drawing into bitmaps
- wxPixelData: Class template for direct access to wxBitmap's and wxImage's internal data

2.16 Events

An event object contains information about a specific event. Event handlers (usually member functions) have a single, event argument.

Related Overviews: Events and Event Handling

- wxActivateEvent: A window or application activation event
- wxCalendarEvent: Used with wxCalendarCtrl
- wxCalculateLayoutEvent: Used to calculate window layout
- wxChildFocusEvent: A child window focus event
- wxClipboardTextEvent: A clipboard copy/cut/paste treebook event event
- wxCloseEvent: A close window or end session event
- wxCommandEvent: An event from a variety of standard controls
- wxContextMenuEvent: An event generated when the user issues a context menu command
- wxDateEvent: Used with wxDatePickerCtrl
- wxDialUpEvent: Event send by wxDialUpManager
- wxDropFilesEvent: A drop files event
- wxEraseEvent: An erase background event
- wxEvent: The event base class
- wxFindDialogEvent: Event sent by wxFindReplaceDialog
- wxFocusEvent: A window focus event
- wxKeyEvent: A keypress event

- wxIconizeEvent: An iconize/restore event
- wxIdleEvent: An idle event
- wxInitDialogEvent: A dialog initialisation event
- wxJoystickEvent: A joystick event
- wxKeyboardState: State of the keyboard modifiers.
- wxListEvent: A list control event
- wxMaximizeEvent: A maximize event
- wxMenuEvent: A menu event
- wxMouseCaptureChangedEvent: A mouse capture changed event
- wxMouseCaptureLostEvent: A mouse capture lost event
- wxMouseEvent: A mouse event
- wxMouseState: State of the mouse
- wxMoveEvent: A move event
- wxNavigationKeyEvent: An event set by navigation keys such as tab
- wxNotebookEvent: A notebook control event
- wxNotifyEvent: A notification event, which can be vetoed
- wxPaintEvent: A paint event
- wxProcessEvent: A process ending event
- wxQueryLayoutInfoEvent: Used to query layout information
- wxRichTextEvent: A rich text editing event
- wxScrollEvent: A scroll event from sliders, stand-alone scrollbars and spin buttons
- wxScrollWinEvent: A scroll event from scrolled windows
- wxSizeEvent: A size event
- wxSocketEvent: A socket event
- wxSpinEvent: An event from wxSpinButton
- wxSplitterEvent: An event from wxSplitterWindow
- wxSysColourChangedEvent: A system colour change event
- wxTimerEvent: A timer expiration event
- wxTreebookEvent: A treebook control event
- wxTreeEvent: A tree control event
- wxUpdateUIEvent: A user interface update event
- wxWindowCreateEvent: A window creation event
- wxWindowDestroyEvent: A window destruction event
- wxWizardEvent: A wizard event

2.17 Application and Process Management

- wxApp: Application class
- wxCmdLineParser: Command line parser class
- wxDynamicLibrary: Class to work with shared libraries.
- wxProcess: Process class

2.18 Printing Framework

A printing and previewing framework is implemented to make it relatively straightforward to provide document printing facilities.

Related Overviews: Printing Framework Overview

- wxPreviewFrame: Frame for displaying a print preview
- wxPreviewCanvas: Canvas for displaying a print preview
- wxPreviewControlBar: Standard control bar for a print preview
- wxPrintDialog: Standard print dialog
- wxPageSetupDialog: Standard page setup dialog
- wxPrinter: Class representing the printer
- wxPrinterDC: Printer device context
- wxPrintout: Class representing a particular printout
- wxPrintPreview: Class representing a print preview
- wxPrintData: Represents information about the document being printed
- wxPrintDialogData: Represents information about the print dialog
- wxPageSetupDialogData: Represents information about the page setup dialog

2.19 Document/View Framework

wxWidgets supports a document/view framework which provides housekeeping for a document-centric application.

Related Overviews: Document/View Framework

- wxCommand: Base class for undo/redo actions
- wxCommandProcessor: Maintains the undo/redo stack
- wxDocument: Represents a document
- wxView: Represents a view
- wxDocTemplate: Manages the relationship between a document class and a view class
- wxDocManager: Manages the documents and views in an application
- wxDocChildFrame: A child frame for showing a document view
- wxDocParentFrame: A parent frame to contain views
- wxDocMDIChildFrame: An MDI child frame for showing a document view
- wxDocMDIParentFrame: An MDI parent frame to contain views
- wxFileHistory: Maintains a list of the most recently visited files

2.20 Clipboard and Drag & Drop

Related Overviews: Drag and Drop Overview

- wxDataObject: Data object class
- wxDataFormat: Represents a data format
- wxTextDataObject: Text data object class
- wxFileDataObject: File data object class
- wxBitmapDataObject: Bitmap data object class
- wxURLDataObject: URL data object class
- wxCustomDataObject: Custom data object class
- wxClipboard: Clipboard class
- wxDropTarget: Drop target class
- wxFileDropTarget: File drop target class
- wxTextDropTarget: Text drop target class
- wxDropSource: Drop source class

2.21 Virtual File System

wxWidgets provides a set of classes that implement an extensible virtual file system, used internally by the HTML classes.

- wxFSFile: Represents a file in the virtual file system
- wxFileSystem: Main interface for the virtual file system
- wxFileSystemHandler: Class used to announce file system type

2.22 Threading

wxWidgets provides a set of classes to make use of the native thread capabilities of the various platforms.

Related Overviews: Multithreading Overview

- wxThread: Thread class
- wxThreadHelper: Manages background threads easily
- wxMutex: Mutex class
- wxMutexLocker: Mutex locker utility class
- wxCriticalSection: Critical section class
- wxCriticalSectionLocker: Critical section locker utility class
- wxCondition: Condition class
- wxSemaphore: Semaphore class

2.23 Runtime Type Information (RTTI)

wxWidgets supports runtime manipulation of class information, and dynamic creation of objects given class names.

Related Overviews: Runtime Type Information (RTTI)

See also

RTTI Functions and Macros

- wxClassInfo: Holds runtime class information
- wxObject: Root class for classes with runtime information

2.24 Debugging

wxWidgets supports some aspects of debugging an application through classes, functions and macros.

Related Overviews: Debugging

See also

Debugging Functions and Macros

- wxDebugContext: Provides memory-checking facilities
- wxDebugReport: Base class for creating debug reports in case of a program crash.
- wxDebugReportCompress: Class for creating compressed debug reports.
- wxDebugReportUpload: Class for uploading compressed debug reports via HTTP.
- wxDebugReportPreview: Abstract base class for previewing the contents of a debug report.
- wxDebugReportPreviewStd: Standard implementation of wxDebugReportPreview.

2.25 Logging

wxWidgets provides several classes and functions for message logging.

Related overview: Logging Overview

See also

Logging Functions and Macros

- wxLog: The base log class
- wxLogStderr: Log messages to a C STDIO stream
- wxLogStream: Log messages to a C++ iostream
- wxLogTextCtrl: Log messages to a wxTextCtrl
- wxLogWindow: Log messages to a log frame
- wxLogGui: Default log target for GUI programs
- wxLogNull: Temporarily suppress message logging
- wxLogChain: Allows to chain two log targets
- wxLogInterposer: Allows to filter the log messages
- wxLogInterposerTemp: Allows to filter the log messages
- wxStreamToTextRedirector: Allows to redirect output sent to `cout` to a wxTextCtrl

2.26 Data Structures

These are the data structure classes supported by wxWidgets.

- wxAny: A class for storing arbitrary types that may change at run-time
- wxCmdLineParser: Command line parser class
- wxDateSpan: A logical time interval.
- wxDateTime: A class for date/time manipulations
- wxLongLong: A portable 64 bit integer type
- wxObject: The root class for most wxWidgets classes
- wxPathList: A class to help search multiple paths
- wxPoint: Representation of a point
- wxRect: A class representing a rectangle
- wxRegEx: Regular expression support
- wxRegion: A class representing a region
- wxString: A string class
- wxStringTokenizer: A class for interpreting a string as a list of tokens or words
- wxRealPoint: Representation of a point using floating point numbers
- wxSize: Representation of a size
- wxTimeSpan: A time interval.
- wxURI: Represents a Uniform Resource Identifier
- wxVariant: A class for storing arbitrary types that may change at run-time

2.27 Text Conversion

These classes define objects for performing conversions between different multibyte and Unicode encodings and wide character strings.

- wxMBConv: Base class for all converters, defines the API implemented by all the other converter classes.
- wxMBConvUTF7: Converter for UTF-7
- wxMBConvUTF8: Converter for UTF-8
- wxMBConvUTF16: Converter for UTF-16
- wxMBConvUTF32: Converter for UTF-32
- wxCSConv: Converter for any system-supported encoding which can be specified by name.

Related Overviews: wxMBConv Overview

2.28 Containers

These are classes, templates and class macros are used by wxWidgets. Most of these classes provide a subset or almost complete STL API.

Related Overviews: Container Classes

- wxArray<T>: A type-safe dynamic array implementation (macro based)
- wxArrayString: An efficient container for storing wxString objects
- wxHashMap<T>: A type-safe hash map implementation (macro based)
- wxHashSet<T>: A type-safe hash set implementation(macro based)
- wxHashTable: A simple hash table implementation (deprecated, use wxHashMap)
- wxList<T>: A type-safe linked list implementation (macro based)
- wxVector<T>: Template base vector implementation identical to std::vector

2.29 Smart Pointers

wxWidgets provides a few smart pointer class templates.

- wxObjectDataPtr<T>: A shared pointer (using intrusive reference counting)
- wxScopedPtr<T>: A scoped pointer
- wxSharedPtr<T>: A shared pointer (using non-intrusive reference counting)
- wxWeakRef<T>: A weak reference

2.30 File Handling

wxWidgets has several small classes to work with disk files and directories.

Related overview: File Classes and Functions

- wxFileName: Operations with the file name and attributes
- wxDir: Class for enumerating files/subdirectories.
- wxDirTraverser: Class used together with wxDir for recursively enumerating the files/subdirectories
- wxFile: Low-level file input/output class.
- wxFFile: Another low-level file input/output class.
- wxTempFile: Class to safely replace an existing file
- wxTextFile: Class for working with text files as with arrays of lines
- wxStandardPaths: Paths for standard directories
- wxPathList: A class to help search multiple paths
- wxFileSystemWatcher: Class providing notifications of file system changes

2.31 Streams

wxWidgets has its own set of stream classes as an alternative to the standard stream libraries and to provide enhanced functionality.

Related overview: Stream Classes Overview

- wxStreamBase: Stream base class
- wxStreamBuffer: Stream buffer class
- wxInputStream: Input stream class
- wxOutputStream: Output stream class
- wxCountingOutputStream: Stream class for querying what size a stream would have.
- wxFilterInputStream: Filtered input stream class
- wxFilterOutputStream: Filtered output stream class
- wxBufferedInputStream: Buffered input stream class
- wxBufferedOutputStream: Buffered output stream class
- wxMemoryInputStream: Memory input stream class
- wxMemoryOutputStream: Memory output stream class
- wxDataInputStream: Platform-independent binary data input stream class
- wxDataOutputStream: Platform-independent binary data output stream class
- wxTextInputStream: Platform-independent text data input stream class
- wxTextOutputStream: Platform-independent text data output stream class
- wxFileInputStream: File input stream class
- wxFileOutputStream: File output stream class
- wxFFileInputStream: Another file input stream class
- wxFFileOutputStream: Another file output stream class
- wxTempFileOutputStream: Stream to safely replace an existing file
- wxStringInputStream: String input stream class
- wxStringOutputStream: String output stream class
- wxZlibInputStream: Zlib and gzip (compression) input stream class
- wxZlibOutputStream: Zlib and gzip (compression) output stream class
- wxZipInputStream: Input stream for reading from ZIP archives
- wxZipOutputStream: Output stream for writing from ZIP archives
- wxTarInputStream: Input stream for reading from tar archives
- wxTarOutputStream: Output stream for writing from tar archives
- wxSocketInputStream: Socket input stream class
- wxSocketOutputStream: Socket output stream class

2.32 XML

- wxXmlDocument: A class to parse XML files
- wxXmlNode: A class which represents XML nodes
- wxXmlAttribute: A class which represent an XML attribute

2.33 Archive

- wxArchiveInputStream
- wxArchiveOutputStream
- wxArchiveEntry

2.34 XML Based Resource System (XRC)

Resources allow your application to create controls and other user interface elements from specifications stored in an XML format.

Related overview: XML Based Resource System (XRC)

- wxXmlResource: The main class for working with resources
- wxXmlResourceHandler: The base class for XML resource handlers

2.35 Networking

wxWidgets provides its own classes for socket based networking.

- wxDialUpManager: Provides functions to check the status of network connection and to establish one
- wxIPV4address: Represents an Internet address
- wxIPaddress: Represents an Internet address
- wxSocketBase: Represents a socket base object
- wxSocketClient: Represents a socket client
- wxSocketServer: Represents a socket server
- wxSocketEvent: A socket event
- wxFTP: FTP protocol class
- wxHTTP: HTTP protocol class
- wxURL: Represents a Universal Resource Locator

2.36 Interprocess Communication

wxWidgets provides simple interprocess communications facilities based on Windows DDE, but available on most platforms using TCP.

Related overview: Interprocess Communication

- wxClient, wxDDEClient: Represents a client
- wxConnection, wxDDEConnection: Represents the connection between a client and a server
- wxServer, wxDDEServer: Represents a server

2.37 Help

- wxHelpController: Family of classes for controlling help windows
- wxHtmlHelpController: HTML help controller class
- wxContextHelp: Class to put application into context-sensitive help mode
- wxContextHelpButton: Button class for putting application into context-sensitive help mode
- wxHelpProvider: Abstract class for context-sensitive help provision
- wxSimpleHelpProvider: Class for simple context-sensitive help provision
- wxHelpControllerHelpProvider: Class for context-sensitive help provision via a help controller
- wxToolTip: Class implementing tooltips

2.38 Multimedia

- wxMediaCtrl: Display multimedia contents.

2.39 OpenGL

- wxGLCanvas: Canvas that you can render OpenGL calls to.
- wxGLContext: Class to ease sharing of OpenGL data resources.

2.40 Miscellaneous

- wxCaret: A caret (cursor) object
- wxConfigBase: Classes for reading/writing the configuration settings
- wxTimer: Timer class
- wxStopWatch: Stop watch class
- wxMimeTypesManager: MIME-types manager class
- wxSystemSettings: System settings class for obtaining various global parameters
- wxSystemOptions: System options class for run-time configuration

- wxAcceleratorTable: Accelerator table
- wxAutomationObject: OLE automation class
- wxFontMapper: Font mapping, finding suitable font for given encoding
- wxEncodingConverter: Encoding conversions
- wxCalendarDateAttr: Used with wxCalendarCtrl
- wxQuantize: Class to perform quantization, or colour reduction
- wxSingleInstanceChecker: Check that only single program instance is running

Chapter 3

Constants

This chapter describes the constants defined by wxWidgets.

- Standard Event Identifiers
- Stock Items
- Preprocessor Symbols
- wxUSE Preprocessor Symbols

3.1 Standard Event Identifiers

wxWidgets defines a special identifier value `wxID_ANY` which is used in the following two situations:

- when creating a new window you may specify `wxID_ANY` to let wxWidgets assign an unused identifier to it automatically
- when installing an event handler using either the event table macros or wxEvtHandler::Connect, you may use it to indicate that you want to handle the events coming from any control, regardless of its identifier

Another standard special identifier value is `wxID_NONE`: this is a value which is not matched by any other id.

wxWidgets also defines a few standard command identifiers which may be used by the user code and also are sometimes used by wxWidgets itself. These reserved identifiers are all in the range between `wxID_LOWEST` and `wxID_HIGHEST` and, accordingly, the user code should avoid defining its own constants in this range (e.g. by using wxNewId()).

Refer to the list of stock items for the subset of standard IDs which are stock IDs as well.

3.2 Stock Items

The following is the list of the window IDs for which stock buttons and menu items are created.

See the wxButton constructor and the wxMenuItem constructor for classes which automatically add stock bitmaps when using stock IDs.

Also note that you can retrieve stock bitmaps using wxArtProvider.

Stock ID	GTK icon	Stock label

wxID_ABOUT		&About
wxID_ADD		Add
wxID_APPLY		&Apply
wxID_BACKWARD		&Back
wxID_BOLD		&Bold
wxID_BOTTOM		&Bottom
wxID_CANCEL		&Cancel
wxID_CDROM		&CD-Rom
wxID_CLEAR		&Clear
wxID_CLOSE		&Close
wxID_CONVERT		&Convert
wxID_COPY		&Copy
wxID_CUT		Cu&t
wxID_DELETE		&Delete
wxID_DOWN		&Down
wxID_EDIT		&Edit
wxID_EXECUTE		&Execute
wxID_EXIT		&Quit
wxID_FILE		&File
wxID_FIND		&Find
wxID_FIRST		&First
wxID_FLOPPY		&Floppy
wxID_FORWARD		&Forward
wxID_HARDDISK		&Harddisk
wxID_HELP		&Help
wxID_HOME		&Home
wxID_INDENT		Indent
wxID_INDEX		&Index
wxID_INFO		&Info
wxID_ITALIC		&Italic
wxID_JUMP_TO		&Jump to
wxID_JUSTIFY_CENTER		Centered
wxID_JUSTIFY_FILL		Justified
wxID_JUSTIFY_LEFT		Align Left
wxID_JUSTIFY_RIGHT		Align Right
wxID_LAST		&Last
wxID_NETWORK		&Network
wxID_NEW		&New
wxID_NO		&No
wxID_OK		&OK
wxID_OPEN		&Open...
wxID_PASTE		&Paste
wxID_PREFERENCES		&Preferences
wxID_PREVIEW		Print previe&w
wxID_PRINT		&Print...
wxID_PROPERTIES		&Properties
wxID_REDO		&Redo

wxID_REFRESH		Refresh
wxID_REMOVE		Remove
wxID_REPLACE		Rep&lace
wxID_REVERT_TO_SAVED		Revert to Saved
wxID_SAVE		&Save
wxID_SAVEAS		Save &As...
wxID_SELECTALL		Select &All
wxID_SELECT_COLOR		&Color
wxID_SELECT_FONT		&Font
wxID_SORT_ASCENDING		&Ascending
wxID_SORT_DESCENDING		&Descending
wxID_SPELL_CHECK		&Spell Check
wxID_STOP		&Stop
wxID_STRIKETHROUGH		&Strikethrough
wxID_TOP		&Top
wxID_UNDELETE		Undelete
wxID_UNDERLINE		&Underline
wxID_UNDO		&Undo
wxID_UNINDENT		&Unindent
wxID_UP		&Up
wxID_YES		&Yes
wxID_ZOOM_100		&Actual Size
wxID_ZOOM_FIT		Zoom to &Fit
wxID_ZOOM_IN		Zoom &In
wxID_ZOOM_OUT		Zoom &Out

Note that some of the IDs listed above also have a stock accelerator and an associated help string.

3.3 Preprocessor Symbols

These are preprocessor symbols used in the wxWidgets source, grouped by category (and sorted by alphabetical order inside each category).

All of these macros except for the `wxUSE_XXX` variety is defined if the corresponding condition is true and undefined if it isn't, so they should be always tested using #ifdef and not #if.

3.3.1 GUI system

__WXBASE__	Only wxBase, no GUI features (same as wxUSE_GUI == 0)
__WXDFB__	wxUniversal using DirectFB
__WXWINCE__	Windows CE
__WXGTK__	GTK+
__WXGTK12__	GTK+ 1.2 or higher
__WXGTK20__	GTK+ 2.0 or higher
__WXGTK24__	GTK+ 2.4 or higher
__WXGTK26__	GTK+ 2.6 or higher
__WXGTK210__	GTK+ 2.10 or higher
__WXMAC__	old define, same as __WXOSX__
__WXMOTIF__	Motif
__WXMOTIF20__	Motif 2.0 or higher
__WXMSW__	GUI using Windows Controls. Notice that for compatibility reasons, this symbol is defined for console applications under Windows as well, but it should only be used in the GUI code while __WINDOWS__ should be used for the platform tests.
__WXOSX__	OS X GUI using any Apple widget framework (Carbon, AppKit or UIKit)
__WXOSX_IPHONE__	OS X iPhone (UIKit)
__WXOSX_CARBON__	Mac OS X using Carbon
__WXOSX_COCOA__	Mac OS X using Cocoa (AppKit)
__WXOSX_MAC__	Mac OS X (Carbon or Cocoa)
__WXPM__	OS/2 native Presentation Manager (not used any longer).
__WXSTUBS__	Stubbed version ('template' wxWin implementation)
__WXXT__	Xt; mutually exclusive with WX_MOTIF, not implemented in wxWidgets 2.x
__WXX11__	wxX11 (__WXUNIVERSAL__ will be also defined)
__WXWINE__	WINE (i.e. WIN32 on Unix)
__WXUNIVERSAL__	wxUniversal port, always defined in addition to one of the symbols above so this should be tested first.
__X__	any X11-based GUI toolkit except GTK+

There are two wxWidgets ports to Mac OS X. One of them, wxOSX is the successor of the venerable wxMac, it currently exists in three versions: Carbon and Cocoa for the desktop and a very early iPhone port. And there is the Cocoa port named wxCocoa which has not been updated very actively since beginning 2008. To summarize:

- If you want to test for wxOSX on the desktop, use __WXOSX_MAC__.
- If you want to test for wxOSX on the iPhone, use __WXOSX_IPHONE__.
- If you want to test for a particular GUI Mac port under OS X, use __WXOSX_CARBON__ or __WXOSX_COCOA__.
- If you want to test for any port under Mac OS X, including, for example, wxGTK and also wxBase, use __DARWIN__ (see below).

The convention is to use the __WX prefix for these symbols, although this has not always been followed.

3.3.2 Operating Systems

__APPLE__	any Mac OS version
__AIX__	AIX
__BSD__	Any ∗BSD system
__CYGWIN__	Cygwin: Unix on Win32
__DARWIN__	Mac OS X (with BSD C library), using any port (see also __WXOSX__)
__DATA_GENERAL__	DG-UX
__FREEBSD__	FreeBSD
__HPUX__	HP-UX (Unix)
__GNU__	GNU Hurd
__LINUX__	Linux
__MACH__	Mach-O Architecture (Mac OS X only builds)
__OSF__	OSF/1
__QNX__	QNX Neutrino RTOS
__SGI__	IRIX
__SOLARIS__	Solaris
__SUN__	Any Sun
__SUNOS__	Sun OS
__SVR4__	SystemV R4
__SYSV__	SystemV generic
__ULTRIX__	Ultrix
__UNIX__	any Unix
__UNIX_LIKE__	Unix, BeOS or VMS
__VMS__	VMS
__WINDOWS__	Any Windows platform, using any port (see also __WXMSW__)
__WIN16__	Win16 API (not supported since wxWidgets 2.6)
__WIN32__	Win32 API
__WIN64__	Win64 (mostly same as Win32 but data type sizes are different)
__WINE__	Wine
_WIN32_WCE	Windows CE version

3.3.3 Hardware Architectures (CPU)

Note that not all of these symbols are always defined, it depends on the compiler used.

__ALPHA__	DEC Alpha architecture
__INTEL__	Intel i386 or compatible
__IA64__	Intel 64 bit architecture
__POWERPC__	Motorola Power PC

3.3.4 Hardware Type

__SMARTPHONE__	Generic mobile devices with phone buttons and a small display
__PDA__	Personal digital assistant, usually with touch screen
__HANDHELD__	Small but powerful computer, usually with a keyboard
__POCKETPC__	Microsoft-powered PocketPC devices with touch-screen
__WINCE_STANDARDSDK__	Microsoft-powered Windows CE devices, for generic Windows CE applications
__WINCE_NET__	Microsoft-powered Windows CE .NET devices (_WIN32_WCE is 400 or greater)

WIN32_PLATFORM_WFSP	Microsoft-powered smartphone

3.3.5 Compilers

__BORLANDC__	Borland C++. The value of the macro corresponds to the compiler version: 500 is 5.0.
__DJGPP__	DJGPP
__DIGITALMARS__	Digital Mars (not used any more).
__EVC4__	Embedded Visual C++ 4 (can be only used for building wxWinCE)
__GNUG__	Gnu C++ on any platform, see also wxCHECK_GCC_VERSION
__GNUWIN32__	Gnu-Win32 compiler, see also wxCHECK_W32API_VERSION
__INTELC__	Intel C++ compiler
__MINGW32__	Either MinGW32 or MinGW-w64 in either 32 or 64 bits
__MINGW32_TOOLCHAIN	MinGW32 only (32 bits only right now)
__MINGW64__	MinGW-w64 in 64 bit builds
__MINGW64_TOOLCHAIN__	MinGW-w64 in either 32 or 64 bit builds
__SUNCC__	Sun CC, see also wxCHECK_SUNCC_VERSION
__SYMANTECC__	Symantec C++ (not used any more).
__VISAGECPP__	IBM Visual Age (OS/2) (not used any more).
__VISUALC__	Microsoft Visual C++, see also wxCHECK_VISUALC_VERSION. The value of this macro corresponds to the compiler version: `1020` for `4.2` (the first supported version), `1100` for `5.0`, `1200` for `6.0` and so on. For convenience, the symbols **VISUALCn** are also defined for each major compiler version from 5 to 12, i.e. you can use tests such as `#ifdef __VISUALC7__` to test for compiler version being precisely 7.
__XLC__	AIX compiler
__WATCOMC__	Watcom C++. The value of this macro corresponds to the compiler version, `1100` is `11.0` and `1200` is OpenWatcom (not used any more).

3.3.6 Feature Tests

Some library features may not be always available even if they were selected by the user. To make it possible to check if this is the case, the library predefines the symbols in the form `wxHAS_FEATURE`. Unlike `wxUSE_FEATURE` symbols which are defined by the library user (directly in `setup.h` or by running configure script) and which must be always defined as either 0 or 1, the `wxHAS` symbols are only defined if the corresponding feature is available and not defined at all otherwise.

Currently the following symbols exist:

wxHAS_3STATE_CHECKBOX	Defined if wxCheckBox supports wxCHK_3STATE flag, i.e. is capable of showing three states and not only the usual two. Currently defined for almost all ports.
wxHAS_ATOMIC_OPS	Defined if wxAtomicInc() and wxAtomicDec() functions have an efficient (CPU-specific) implementation. Notice that the functions themselves are always available but can be prohibitively slow to use when implemented in a generic way, using a critical section.

wxHAS_BITMAPTOGGLEBUTTON	Defined in wx/tglbtn.h if wxBitmapToggleButton class is available in addition to wxToggleButton.
wxHAS_CONFIG_TEMPLATE_RW	Defined if the currently used compiler supports template Read() and Write() methods in wxConfig.
wxHAS_LARGE_FILES	Defined if wxFile supports files more than 4GB in size (notice that you must include wx/filefn.h before testing for this symbol).
wxHAS_LARGE_FFILES	Defined if wxFFile supports files more than 4GB in size (notice that you must include wx/filefn.h before testing for this symbol).
wxHAS_LONG_LONG_T_DIFFERENT_FROM_LONG	Defined if compiler supports a 64 bit integer type (available as wxLongLong_t) and this type is different from long. Notice that, provided wxUSE_LONGLONG is not turned off, some 64 bit type is always available to wxWidgets programs and this symbol only indicates a presence of such primitive type. It is useful to decide whether some function should be overloaded for both long and long long types.
wxHAS_MULTIPLE_FILEDLG_FILTERS	Defined if wxFileDialog supports multiple ('|'-separated) filters.
wxHAS_IMAGES_IN_RESOURCES	Defined if Windows resource files or OS/2 resource files are available on the current platform.
wxHAS_POWER_EVENTS	Defined if wxPowerEvent are ever generated on the current platform.
wxHAS_RADIO_MENU_ITEMS	Defined if the current port supports radio menu items (see wxMenu::AppendRadioItem).
wxHAS_RAW_BITMAP	Defined if direct access to bitmap data using the classes in wx/rawbmp.h is supported.
wxHAS_RAW_KEY_CODES	Defined if raw key codes (see wxKeyEvent::GetRawKeyCode are supported.
wxHAS_REGEX_ADVANCED	Defined if advanced syntax is available in wxRegEx.
wxHAS_TASK_BAR_ICON	Defined if wxTaskBarIcon is available on the current platform.

3.3.7 Library Selection for MSVC

Microsoft Visual C++ users may use the special wx/setup.h file for this compiler in include/msvc subdirectory. This file implicitly links in all the wxWidgets libraries using MSVC-specific pragmas which usually is much more convenient than manually specifying the libraries list in all of the project configurations. However sometimes linking with all the libraries is not desirable, for example because some of them were not built and this is where the symbols in this section can be helpful: defining them allows to not link with the corresponding library. The following symbols are honoured:

- wxNO_ADV_LIB
- wxNO_AUI_LIB
- wxNO_HTML_LIB
- wxNO_MEDIA_LIB
- wxNO_NET_LIB
- wxNO_PROPGRID_LIB
- wxNO_QA_LIB
- wxNO_RICHTEXT_LIB
- wxNO_WEBVIEW_LIB
- wxNO_XML_LIB
- wxNO_REGEX_LIB
- wxNO_EXPAT_LIB
- wxNO_JPEG_LIB
- wxNO_PNG_LIB
- wxNO_TIFF_LIB
- wxNO_ZLIB_LIB

Notice that the base library is always included and the core is always included for the GUI applications (i.e. those which don't define wxUSE_GUI as 0).

If the makefiles have been used to build the libraries from source and the `CFG` variable has been set to specify a different output path for that particular configuration of build then the `wxCFG` preprocessor symbol should be set in the project that uses wxWidgets to the same value as the `CFG` variable in order for the correct `wx/setup.h` file to automatically be included for that configuration.

3.3.8 Miscellaneous

__WXWINDOWS__	always defined in wxWidgets applications, see also wxCHECK_VERSION
wxDEBUG_LEVEL	defined as 1 by default, may be pre-defined as 0 before including wxWidgets headers to disable generation of any code at all for the assertion macros, see Debugging
__WXDEBUG__	defined if wxDEBUG_LEVEL is 1 or more, undefined otherwise
wxUSE_XXX	if defined as 1, feature XXX is active, see the wxUSE Preprocessor Symbols (the symbols of this form are always defined, use #if and not #ifdef to test for them)
WX_PRECOMP	is defined if precompiled headers (PCH) are in use. In this case, `wx/wxprec.h` includes `wx/wx.h` which, in turn, includes a number of wxWidgets headers thus making it unnecessary to include them explicitly. However if this is not defined, you do need to include them and so the usual idiom which allows to support both cases is to first include `wx/wxprec.h` and then, inside `#ifndef WX_PRECOMP`, individual headers you need.}
_UNICODE and UNICODE	both are defined if wxUSE_UNICODE is set to `1`
wxUSE_GUI	this particular feature test macro is defined to 1 when compiling or using the library with the GUI features activated, if it is defined as `0`, only wxBase is available.
wxUSE_BASE	only used by wxWidgets internally (defined as 1 when building wxBase code, either as a standalone library or as part of the monolithic wxWidgets library, defined as 0 when building GUI library only)
wxNO_RTTI	is defined if the compiler RTTI support has been switched off
wxNO_EXCEPTIONS	is defined if the compiler support for C++ exceptions has been switched off
wxNO_THREADS	if this macro is defined, the compilation options don't include compiler flags needed for multithreaded code generation. This implies that wxUSE_THREADS is 0 and also that other (non-wx-based) threading packages cannot be used neither.
WXMAKINGDLL_XXX	used internally and defined when building the library `XXX` as a DLL; when a monolithic wxWidgets build is used only a single `WXMAKINGDLL` symbol is defined
WXUSINGDLL	defined when compiling code which uses wxWidgets as a DLL/shared library

WXBUILDING	defined when building wxWidgets itself, whether as a static or shared library

3.4 wxUSE Preprocessor Symbols

This section documents the wxUSE preprocessor symbols used in the wxWidgets source, grouped by category (and sorted by alphabetical order inside each category).

These symbols are always defined and whether the given feature is active or not depends on their value: if defined as `1`, feature is active, otherwise it is disabled. Because of this these symbols should always be tested using #if and not #ifdef.

3.4.1 Most Important Symbols

This table summarizes some of the global build features affecting the entire library:

wxUSE_STL	Container classes and wxString are implemented using standard classes and provide the same standard API.
wxUSE_STD_STRING	wxString is implemented using std::[w]string and can be constructed from it (but provides wxWidgets-compatible API, in particular is implicitly convertible to `char*` and not std::[w]string).
wxUSE_STD_IOSTREAM	Standard C++ classes are used instead of or in addition to wx stream classes.
wxUSE_UNICODE	Compiled with Unicode support (default in wxWidgets 3.0, non-Unicode build will be deprecated in the future).
wxUSE_UNICODE_WCHAR	wxString uses wchar_t buffer for internal storage (default under MSW).
wxUSE_UNICODE_UTF8	wxString uses UTF-8 for internal storage (default under Unix and Mac systems).
wxUSE_UTF8_LOCALE_ONLY	Library supports running only under UTF-8 (and C) locale. This eliminates the code necessary for conversions from the other locales and reduces the library size; useful for embedded systems.
wxUSE_GUI	Use the GUI classes; if set to 0 only non-GUI classes are available.

3.4.2 Generic Symbols

wxUSE_ABOUTDLG	Use wxAboutDialogInfo class.
wxUSE_ACCEL	Use wxAcceleratorTable/Entry classes and support for them in wxMenu, wxMenuBar.
wxUSE_AFM_FOR_POSTSCRIPT	In wxPostScriptDC class use AFM (adobe font metrics) file for character widths.
wxUSE_ANIMATIONCTRL	Use wxAnimationCtrl class.
wxUSE_ARTPROVIDER_STD	Use standard low quality icons in wxArtProvider.
wxUSE_ARTPROVIDER_TANGO	Use Tango icons in wxArtProvider.
wxUSE_ANY	Use wxAny class.
wxUSE_APPLE_IEEE	IEEE Extended to/from double routines, see wxDataOutputStream.

wxUSE_ARCHIVE_STREAMS	Enable streams for archive formats.
wxUSE_AUI	Use AUI (dockable windows) library.
wxUSE_BASE64	Enables Base64 support.
wxUSE_BITMAPCOMBOBOX	Use wxBitmapComboBox class.
wxUSE_BMPBUTTON	Use wxBitmapButton class.
wxUSE_BUSYINFO	Use wxBusyInfo class.
wxUSE_BUTTON	Use wxButton class.
wxUSE_CALENDARCTRL	Use wxCalendarCtrl class.
wxUSE_CARET	Use wxCaret class.
wxUSE_CHECKBOX	Use wxCheckBox class.
wxUSE_CHECKLISTBOX	Use wxCheckListBox class.
wxUSE_CHOICE	Use wxChoice class.
wxUSE_CHOICEBOOK	Use wxChoicebook class.
wxUSE_CHOICEDLG	Use wxSingleChoiceDialog, or wxMultiChoiceDialog classes.
wxUSE_CLIPBOARD	Use wxClipboard class.
wxUSE_CMDLINE_PARSER	Use wxCmdLineParser class.
wxUSE_COLLPANE	Use wxCollapsiblePane class.
wxUSE_COLOURDLG	Use wxColourDialog class.
wxUSE_COLOURPICKERCTRL	Use wxColourPickerCtrl class.
wxUSE_COMBOBOX	Use wxComboBox class.
wxUSE_COMBOCTRL	Use wxComboCtrl class.
wxUSE_COMPILER_TLS	Can be set to 0 to prevent using compile thread-specific variables support.
wxUSE_CONFIG	Use wxConfig and related classes.
wxUSE_CONFIG_NATIVE	When enabled use native OS configuration instead of the wxFileConfig class.
wxUSE_CONSOLE_EVENTLOOP	Enable event loop in console programs.
wxUSE_CONSTRAINTS	Use wxLayoutConstraints
wxUSE_CONTROLS	If set to 0, no classes deriving from wxControl can be used.
wxUSE_DATAOBJ	Use wxDataObject and related classes.
wxUSE_DATAVIEWCTRL	Use wxDataViewCtrl class.
wxUSE_DATEPICKCTRL	Use wxDatePickerCtrl class.
wxUSE_DATETIME	Use wxDateTime and related classes.
wxUSE_DBGHELP	Predefine as 0 to avoid using wxDbgHelpDLL and related classes.
wxUSE_DC_TRANSFORM_MATRIX	Use wxDC::SetTransformMatrix() and related methods.
wxUSE_DEBUG_CONTEXT	Use wxDebugContext class.
wxUSE_DEBUG_NEW_ALWAYS	See Debugging
wxUSE_DEBUGREPORT	Use wxDebugReport class.
wxUSE_DIALUP_MANAGER	Use wxDialUpManager and related classes.
wxUSE_DIRDLG	Use wxDirDialog class.
wxUSE_DIRPICKERCTRL	Use wxDirPickerCtrl class.
wxUSE_DISPLAY	Use wxDisplay and related classes.
wxUSE_DOC_VIEW_ARCHITECTURE	Use wxDocument and related classes.
wxUSE_DRAG_AND_DROP	Use Drag and drop classes.

wxUSE_DRAGIMAGE	Use wxDragImage class.
wxUSE_DYNAMIC_LOADER	Use wxPluginManager and related classes. Requires wxDynamicLibrary
wxUSE_DYNLIB_CLASS	Use wxDynamicLibrary
wxUSE_EDITABLELISTBOX	Use wxEditableListBox class.
wxUSE_EXCEPTIONS	Use exception handling.
wxUSE_EXPAT	enable XML support using expat parser.
wxUSE_EXTENDED_RTTI	Use extended RTTI, see also Runtime class information (RTTI)
wxUSE_FFILE	Use wxFFile class.
wxUSE_FILE	Use wxFile class.
wxUSE_FILECONFIG	Use wxFileConfig class.
wxUSE_FILECTRL	Use wxFileCtrl class.
wxUSE_FILEDLG	Use wxFileDialog class.
wxUSE_FILEPICKERCTRL	Use wxFilePickerCtrl class.
wxUSE_FILESYSTEM	Use wxFileSystem and related classes.
wxUSE_FINDREPLDLG	Use wxFindReplaceDialog class.
wxUSE_FONTDLG	Use wxFontDialog class.
wxUSE_FONTENUM	Use wxFontEnumerator class.
wxUSE_FONTMAP	Use wxFontMapper class.
wxUSE_FONTPICKERCTRL	Use wxFontPickerCtrl class.
wxUSE_FS_ARCHIVE	Use virtual archive filesystems like wxArchiveFSHandler in wxFileSystem class.
wxUSE_FS_INET	Use virtual HTTP/FTP filesystems like wxInternetFSHandler in wxFileSystem class.
wxUSE_FS_ZIP	Please use wxUSE_FS_ARCHIVE instead.
wxUSE_FSVOLUME	Use wxFSVolume class.
wxUSE_GAUGE	Use wxGauge class.
wxUSE_GENERIC_DRAGIMAGE	Used in wxDragImage sample.
wxUSE_GENERIC_DRAWELLIPSE	See comment in wx/dc.h file.
wxUSE_GENERIC_MDI_AS_NATIVE	This is not a user-settable symbol, it is only used internally in wx/generic/mdig.h.
wxUSE_GEOMETRY	Use common geometry classes
wxUSE_GIF	Use GIF wxImageHandler
wxUSE_GLCANVAS	Enables OpenGL support.
wxUSE_GLOBAL_MEMORY_OPERATORS	Override global operators `new` and `delete` to use wxWidgets memory leak detection
wxUSE_GRAPHICS_CONTEXT	Use wxGraphicsContext and related classes.
wxUSE_GRID	Use wxGrid and related classes.
wxUSE_HELP	Use wxHelpController and related classes.
wxUSE_HTML	Use wxHtmlWindow and related classes.
wxUSE_HYPERLINKCTRL	Use wxHyperlinkCtrl
wxUSE_ICO_CUR	Support Windows ICO and CUR formats.
wxUSE_IFF	Enables the wxImage handler for Amiga IFF images.
wxUSE_IMAGE	Use wxImage and related classes.
wxUSE_IMAGLIST	Use wxImageList class.
wxUSE_INTL	Use wxLocale and related classes.
wxUSE_IOSTREAMH	Use header "iostream.h" instead of "iostream".

wxUSE_IPC	Use interprocess communication classes.
wxUSE_IPV6	Use experimental wxIPV6address and related classes.
wxUSE_JOYSTICK	Use wxJoystick class.
wxUSE_LIBJPEG	Enables JPEG format support (requires libjpeg).
wxUSE_LIBPNG	Enables PNG format support (requires libpng). Also requires wxUSE_ZLIB.
wxUSE_LIBTIFF	Enables TIFF format support (requires libtiff).
wxUSE_LISTBOOK	Use wxListbook class.
wxUSE_LISTBOX	Use wxListBox class.
wxUSE_LISTCTRL	Use wxListCtrl class.
wxUSE_LOG	Use wxLog and related classes.
wxUSE_LOG_DEBUG	Enabled when wxLog used with **WXDEBUG** defined.
wxUSE_LOG_DIALOG	Use wxLogDialog class.
wxUSE_LOGGUI	Use wxLogGui class.
wxUSE_LOGWINDOW	Use wxLogFrame class.
wxUSE_LONGLONG	Use wxLongLong class.
wxUSE_LONGLONG_NATIVE	Use native `long long` type in wxLongLong implementation.
wxUSE_LONGLONG_WX	Use generic wxLongLong implementation.
wxUSE_MARKUP	Provide wxControl::SetLabelMarkup() method.
wxUSE_MDI	Use wxMDIParentFrame, and wxMDIChildFrame
wxUSE_MDI_ARCHITECTURE	Use MDI-based document-view classes.
wxUSE_MEDIACTRL	Use wxMediaCtrl.
wxUSE_MEMORY_TRACING	Use wxWidgets memory leak detection, not recommended if using another memory debugging tool.
wxUSE_MENUS	Use wxMenu and related classes.
wxUSE_METAFILE	Use wxMetaFile and related classes.
wxUSE_MIMETYPE	Use wxFileType class.
wxUSE_MINIFRAME	Use wxMiniFrame class.
wxUSE_MOUSEWHEEL	Support mouse wheel events.
wxUSE_MSGDLG	Use wxMessageDialog class and wxMessageBox function.
wxUSE_NATIVE_STATUSBAR	Use native wxStatusBar class.
wxUSE_NOTEBOOK	Use wxNotebook and related classes.
wxUSE_NUMBERDLG	Use wxNumberEntryDialog class.
wxUSE_ODCOMBOBOX	Use wxOwnerDrawnComboBox class.
wxUSE_ON_FATAL_EXCEPTION	Catch signals in wxApp::OnFatalException method.
wxUSE_OPENGL	Please use wxUSE_GLCANVAS to test for enabled OpenGL support instead.
wxUSE_OWNER_DRAWN	Use interface for owner-drawn GUI elements.
wxUSE_PALETTE	Use wxPalette and related classes.
wxUSE_PCX	Enables wxImage PCX handler.
wxUSE_PNM	Enables wxImage PNM handler.
wxUSE_POPUPWIN	Use wxPopupWindow class.
wxUSE_POSTSCRIPT	Use wxPostScriptPrinter class.
wxUSE_PRINTF_POS_PARAMS	Use wxVsnprintf which supports positional parameters.

wxUSE_PRINTING_ARCHITECTURE	Enable printer classes.
wxUSE_PROGRESSDLG	Enables progress dialog classes.
wxUSE_PROPGRID	Use wxPropertyGrid library.
wxUSE_PROTOCOL	Use wxProtocol and derived classes.
wxUSE_PROTOCOL_FILE	Use wxFileProto class. (requires wxProtocol)
wxUSE_PROTOCOL_FTP	Use wxFTP class. (requires wxProtocol)
wxUSE_PROTOCOL_HTTP	Use wxHTTP class. (requireswxProtocol)
wxUSE_RADIOBOX	Use wxRadioBox class.
wxUSE_RADIOBTN	Use wxRadioButton class.
wxUSE_REGEX	Use wxRegEx class.
wxUSE_RICHTEXT	Use wxRichTextCtrl class.
wxUSE_RICHTEXT_XML_HANDLER	See src/xrc/xh_richtext.cpp file.
wxUSE_SASH	Use wxSashWindow class.
wxUSE_SCROLLBAR	Use wxScrollBar class.
wxUSE_SEARCHCTRL	Use wxSearchCtrl class.
wxUSE_SELECT_DISPATCHER	Use wxSelectDispatcher class.
wxUSE_SLIDER	Use wxSlider class.
wxUSE_SNGLINST_CHECKER	Use wxSingleInstanceChecker class.
wxUSE_SOCKETS	Enables Network address classes.
wxUSE_SOUND	Use wxSound class.
wxUSE_SPINBTN	Use wxSpinButton class.
wxUSE_SPINCTRL	Use wxSpinCtrl class.
wxUSE_SPLASH	Use wxSplashScreen class.
wxUSE_SPLINES	Provide methods for spline drawing in wxDC.
wxUSE_SPLITTER	Use wxSplitterWindow class.
wxUSE_STACKWALKER	Enables wxStackWalker and related classes.
wxUSE_STARTUP_TIPS	Use startup tips, wxTipProvider class.
wxUSE_STATBMP	Use wxStaticBitmap class.
wxUSE_STATBOX	Use wxStaticBox class.
wxUSE_STATLINE	Use wxStaticLine class.
wxUSE_STATTEXT	Use wxStaticText class.
wxUSE_STATUSBAR	Use wxStatusBar class.
wxUSE_STC	Use wxStyledTextCtrl.
wxUSE_STDPATHS	Use wxStandardPaths class.
wxUSE_STOPWATCH	Use wxStopWatch class.
wxUSE_STREAMS	Enable stream classes.
wxUSE_SVG	Use wxSVGFileDC class.
wxUSE_SYSTEM_OPTIONS	Use wxSystemOptions class.
wxUSE_TAB_DIALOG	Use the obsolete wxTabControl class.
wxUSE_TARSTREAM	Enable Tar files support.
wxUSE_TASKBARICON	Use wxTaskBarIcon class.
wxUSE_TEXTBUFFER	Use wxTextBuffer class.
wxUSE_TEXTCTRL	Use wxTextCtrl class.
wxUSE_TEXTDLG	Use wxTextEntryDialog class.
wxUSE_TEXTFILE	Use wxTextFile class.
wxUSE_TGA	Enable wxImage TGA handler.
wxUSE_THREADS	Use wxThread and related classes.

wxUSE_TIMER	Use wxTimer class.
wxUSE_TIPWINDOW	Use wxTipWindow class.
wxUSE_TOGGLEBTN	Use wxToggleButton class.
wxUSE_TOOLBAR	Use wxToolBar class.
wxUSE_TOOLBAR_NATIVE	Use native wxToolBar class.
wxUSE_TOOLBOOK	Use wxToolbook class.
wxUSE_TOOLTIPS	Use wxToolTip class.
wxUSE_TREEBOOK	Use wxTreebook class.
wxUSE_TREECTRL	Use wxTreeCtrl class.
wxUSE_TREELISTCTRL	Use wxTreeListCtrl class.
wxUSE_TTM_WINDOWFROMPOINT	Obsolete, do not use.
wxUSE_URL	Use wxURL class.
wxUSE_URL_NATIVE	Use native support for some operations with wxURL.
wxUSE_VALIDATORS	Use wxValidator class.
wxUSE_VARIANT	Use wxVariant class.
wxUSE_WEBVIEW	Use wxWebView class.
wxUSE_WIZARDDLG	Use wxWizard class.
wxUSE_WXHTML_HELP	Use wxHtmlHelpController and related classes.
wxUSE_XML	Use XML parsing classes.
wxUSE_XPM	Enable XPM reader for wxImage and wxBitmap classes.
wxUSE_XRC	Use XRC XML-based resource system.
wxUSE_ZIPSTREAM	Enable streams for Zip files.
wxUSE_ZLIB	Use wxZlibInput and wxZlibOutputStream classes, required by wxUSE_LIBPNG.

3.4.3 Unix Platform Symbols

wxUSE_EPOLL_DISPATCHER	Use wxEpollDispatcher class. See also wxUSE_SELECT_DISPATCHER.
wxUSE_GSTREAMER	Use GStreamer library in wxMediaCtrl.
wxUSE_LIBMSPACK	Use libmspack library.
wxUSE_LIBSDL	Use SDL for wxSound implementation.
wxUSE_PLUGINS	See also wxUSE_LIBSDL.
wxUSE_UNIX	Enabled on Unix Platform.

3.4.4 wxX11 Symbols

wxUSE_NANOX	Use NanoX.
wxUSE_UNIV_TEXTCTRL	Use wxUniv's implementation of wxTextCtrl class.

3.4.5 wxGTK Symbols

wxUSE_DETECT_SM	Use code to detect X11 session manager.
wxUSE_GTKPRINT	Use GTK+ printing support.
wxUSE_LIBGNOMEVFS	Use GNOME VFS support. Currently has no effect.
wxUSE_LIBHILDON	Use Hildon framework for Nokia 770. Currently has no effect.

3.4.6 wxMac Symbols

wxUSE_MAC_CRITICAL_REGION_MUTEX	See src/osx/carbon/thread.cpp file.

wxUSE_MAC_PTHREADS_MUTEX	See src/osx/carbon/thread.cpp file.
wxUSE_MAC_SEMAPHORE_MUTEX	See src/osx/carbon/thread.cpp file.
wxUSE_WEBKIT	Use wxWebKitCtrl class.

3.4.7 wxMotif Symbols

wxUSE_GADGETS	Use xmCascadeButtonGadgetClass, xmLabelGadgetClass, xmPushButtonGadgetClass and xmToggleButtonGadgetClass classes.
wxUSE_INVISIBLE_RESIZE	See src/motif/dialog.cpp file.

3.4.8 Cocoa Symbols

wxUSE_OBJC_UNIQUIFYING	Enable Objective-C class name uniquifying.

3.4.9 wxMSW Symbols

wxUSE_ACCESSIBILITY	Enable accessibility support
wxUSE_ACTIVEX	Use wxActiveXContainer and related classes.
wxUSE_COMBOCTRL_POPUP_ANIMATION	See wx/msw/combo.h file.
wxUSE_COMCTL32_SAFELY	See src/msw/treectrl.cpp file.
wxUSE_COMMON_DIALOGS	Enable use of windows common dialogs from header commdlg.h; example PRINTDLG.
wxUSE_CRASHREPORT	Use wxCrashReport class.
wxUSE_DATEPICKCTRL_GENERIC	Use generic wxDatePickerCtrl implementation in addition to the native one.
wxUSE_DC_CACHEING	cache temporary wxDC objects.
wxUSE_DDE_FOR_IPC	See wx/ipc.h file.
wxUSE_ENH_METAFILE	Use wxEnhMetaFile.
wxUSE_HOTKEY	Use wxWindow::RegisterHotKey() and wxWindow::UnregisterHotKey
wxUSE_INKEDIT	Use InkEdit library. Related to Tablet PCs.
wxUSE_MS_HTML_HELP	Use wxCHMHelpController class.
wxUSE_NO_MANIFEST	Can be predefined to disable inclusion of the manifest from wxWidgets RC file. See also wxUSE_RC_MANIFEST.
wxUSE_OLE	Enables OLE helper routines.
wxUSE_OLE_AUTOMATION	Enable OLE automation utilities.
wxUSE_OLE_CLIPBOARD	Use OLE clipboard.
wxUSE_PENWINDOWS	See src/msw/penwin.cpp file.
wxUSE_POSTSCRIPT_ARCHITECTURE_IN_MSW	Use PS printing in wxMSW.
wxUSE_PS_PRINTING	See src/msw/dcprint.cpp file.
wxUSE_RC_MANIFEST	Include manifest for common controls library v6 from wxWidgets RC file. This may be needed to be defined explicitly for MSVC 7 (a.k.a. MSVS 2003) only as later versions of MSVC generate this manifest themselves and the manifest generation is enabled by default for the other compilers. See also wxUSE_NO_MANIFEST.

wxUSE_REGKEY	Use wxRegKey class.
wxUSE_RICHEDIT	Enable use of riched32.dll in wxTextCtrl
wxUSE_RICHEDIT2	Enable use of riched20.dll in wxTextCtrl
wxUSE_VC_CRTDBG	See wx/msw/msvcrt.h file.
wxUSE_UXTHEME	Enable support for XP themes.
wxUSE_WIN_METAFILES_ALWAYS	Use wxMetaFile even when wxUSE_ENH_METAFILE=1.
wxUSE_WXDIB	Use wxDIB class.

3.4.10 wxUniversal Symbols

wxUSE_ALL_THEMES	Use all themes in wxUniversal; See wx/univ/theme.h file.
wxUSE_THEME_GTK	Use GTK+ 1-like theme in wxUniversal
wxUSE_THEME_METAL	Use GTK+ 2-like theme in wxUniversal
wxUSE_THEME_MONO	Use simple monochrome theme in wxUniversal
wxUSE_THEME_WIN32	Use Win32-like theme in wxUniversal

Chapter 4

Copyrights and Licenses

4.1 wxWidgets Copyrights and Licenses

- wxWindows Library Licence
- GNU Library General Public License
- The Open Group and DEC License

4.2 Acknowledgements

The following is the list of the core, active developers of wxWidgets which keep it running and have provided an invaluable, extensive and high-quality amount of changes over the many of years of wxWidgets' life:

- Julian Smart
- Vadim Zeitlin
- Robert Roebling
- Robin Dunn
- Stefan Csomor
- Vaclav Slavik
- Paul Cornett
- Wlodzimierz 'ABX' Skiba
- Chris Elliott
- David Elliott
- Kevin Hock
- Stefan Neis

- Michael Wetherell

We would particularly like to thank the following peoples for their contributions to wxWidgets, and the many others who have been involved in the project over the years. Apologies for any unintentional omissions from this alphabetic list:

Yiorgos Adamopoulos, Jamshid Afshar, Alejandro Aguilar-Sierra, AIAI, Patrick Albert, Karsten Ballueder, Mattia Barbon, Michael Bedward, Kai Bendorf, Yura Bidus, Keith Gary Boyce, Chris Breeze, Pete Britton, Ian Brown, C. Buckley, Marco Cavallini, Dmitri Chubraev, Robin Corbet, Cecil Coupe, Andrew Davison, Gilles Depeyrot, Neil Dudman, Hermann Dunkel, Jos van Eijndhoven, Tom Felici, Thomas Fettig, Matthew Flatt, Pasquale Foggia, Josep Fortiana, Todd Fries, Dominic Gallagher, Guillermo Rodriguez Garcia, Wolfram Gloger, Norbert Grotz, Stefan Gunter, Bill Hale, Patrick Halke, Stefan Hammes, Guillaume Helle, Harco de Hilster, Cord Hockemeyer, Markus Holzem, Olaf Klein, Leif Jensen, Bart Jourquin, Guilhem Lavaux, Ron Lee, Jan Lessner, Nicholas Liebmann, Torsten Liermann, Per Lindqvist, Francesco Montorsi, Thomas Runge, Tatu Männistö, Scott Maxwell, Thomas Myers, Oliver Niedung, Ryan Norton, Hernan Otero, Ian Perrigo, Timothy Peters, Giordano Pezzoli, Harri Pasanen, Thomaso Paoletti, Garrett Potts, Marcel Rasche, Dino Scaringella, Jobst Schmalenbach, Arthur Seaton, Paul Shirley, Stein Somers, Petr Smilauer, Neil Smith, Kari Systä, George Tasker, Arthur Tetzlaff-Deas, Jonathan Tonberg, Jyrki Tuomi, Janos Vegh, Andrea Venturoli, David Webster, Otto Wyss, Xiaokun Zhu, Edward Zimmermann.

Many thanks also to AIAI for being willing to release the original version of wxWidgets into the public domain, and to our patient partners.

'Graphplace', the basis for the wxGraphLayout library, is copyright Dr. Jos T.J. van Eijndhoven of Eindhoven University of Technology. The code has been used in wxGraphLayout (not in wxWidgets anymore) with his permission.

We also acknowledge the author of XFIG, the excellent Unix drawing tool, from the source of which we have borrowed some spline drawing code. His copyright is included below.

XFig2.1 is copyright (c) 1985 by Supoj Sutanthavibul. Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

4.3 wxWindows Library Licence

```
                wxWindows Library Licence, Version 3.1
                ======================================

  Copyright (c) 1998-2005 Julian Smart, Robert Roebling et al

  Everyone is permitted to copy and distribute verbatim copies
  of this licence document, but changing it is not allowed.

                       WXWINDOWS LIBRARY LICENCE
     TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  This library is free software; you can redistribute it and/or modify it
  under the terms of the GNU Library General Public Licence as published by
  the Free Software Foundation; either version 2 of the Licence, or (at
  your option) any later version.

  This library is distributed in the hope that it will be useful, but
  WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Library
  General Public Licence for more details.

  You should have received a copy of the GNU Library General Public Licence
  along with this software, usually in a file named COPYING.LIB.  If not,
  write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
  Boston, MA 02110-1301 USA.

  EXCEPTION NOTICE
```

```
  1. As a special exception, the copyright holders of this library give
  permission for additional uses of the text contained in this release of
  the library as licenced under the wxWindows Library Licence, applying
  either version 3.1 of the Licence, or (at your option) any later version of
  the Licence as published by the copyright holders of version
  3.1 of the Licence document.

  2. The exception is that you may use, copy, link, modify and distribute
  under your own terms, binary object code versions of works based
  on the Library.

  3. If you copy code from files distributed under the terms of the GNU
  General Public Licence or the GNU Library General Public Licence into a
  copy of this library, as this licence permits, the exception does not
  apply to the code that you add in this way.  To avoid misleading anyone as
  to the status of such modified files, you must delete this exception
  notice from such code and/or adjust the licensing conditions notice
  accordingly.

  4. If you write modifications of your own for this library, it is your
  choice whether to permit this exception to apply to your modifications.
  If you do not wish that, you must delete the exception notice from such
  code and/or adjust the licensing conditions notice accordingly.
```

4.4 GNU Library General Public License

```
  GNU LIBRARY GENERAL PUBLIC LICENSE
  ==================================
                Version 2, June 1991

 Copyright (C) 1991 Free Software Foundation, Inc.
                    675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the library GPL.  It is
 numbered 2 because it goes with version 2 of the ordinary GPL.]

                        Preamble

The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General
Public Licenses are intended to guarantee your freedom to share
and change free software--to make sure the software is free for
all its users.

This license, the Library General Public License, applies to
some specially designated Free Software Foundation software, and
to any other libraries whose authors decide to use it.  You can
use it for your libraries, too.

When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure
that you have the freedom to distribute copies of free software
(and charge for this service if you wish), that you receive
source code or can get it if you want it, that you can change
the software or use pieces of it in new free programs; and that
you know you can do these things.

To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the
rights. These restrictions translate to certain responsibilities
for you if you distribute copies of the library, or if you
modify it.

For example, if you distribute copies of the library, whether
gratis or for a fee, you must give the recipients all the rights
that we gave you.  You must make sure that they, too, receive or
can get the source code.  If you link a program with the
```

library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which

```
contains a notice placed by the copyright holder or other
authorized party saying it may be distributed under the terms of
this Library General Public License (also called "this
License").  Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application
programs (which use some of those functions and data) to form
executables.

The "Library", below, refers to any such software library or
work which has been distributed under these terms.  A "work
based on the Library" means either the Library or any derivative
work under copyright law: that is to say, a work containing the
Library or a portion of it, either verbatim or with
modifications and/or translated straightforwardly into another
language.  (Hereinafter, translation is included without
limitation in the term "modification".)

"Source code" for a work means the preferred form of the work
for making modifications to it.  For a library, complete source
code means all the source code for all modules it contains, plus
any associated interface definition files, plus the scripts used
to control compilation and installation of the library.

Activities other than copying, distribution and modification are
not covered by this License; they are outside its scope.  The
act of running a program using the Library is not restricted,
and output from such a program is covered only if its contents
constitute a work based on the Library (independent of the use
of the Library in a tool for writing it).  Whether that is true
depends on what the Library does and what the program that uses
the Library does.

1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided
that you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep
intact all the notices that refer to this License and to the
absence of any warranty; and distribute a copy of this License
along with the Library.

You may charge a fee for the physical act of transferring a
copy, and you may at your option offer warranty protection in
exchange for a fee.

2. You may modify your copy or copies of the Library or any
portion of it, thus forming a work based on the Library, and
copy and distribute such modifications or work under the terms
of Section 1 above, provided that you also meet all of these
conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices
    stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no
    charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a
    table of data to be supplied by an application program that uses
    the facility, other than as an argument passed when the facility
    is invoked, then you must make a good faith effort to ensure that,
    in the event an application does not supply such function or
    table, the facility still operates, and performs whatever part of
    its purpose remains meaningful.

    (For example, a function in a library to compute square roots has
    a purpose that is entirely well-defined independent of the
    application.  Therefore, Subsection 2d requires that any
    application-supplied function or table used by this function must
    be optional: if the application does not supply it, the square
```

```
    root function must still compute square roots.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the
Library, and can be reasonably considered independent and
separate works in themselves, then this License, and its terms,
do not apply to those sections when you distribute them as
separate works.  But when you distribute the same sections as
part of a whole which is a work based on the Library, the
distribution of the whole must be on the terms of this License,
whose permissions for other licensees extend to the entire
whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or
contest your rights to work written entirely by you; rather, the
intent is to exercise the right to control the distribution of
derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the
Library with the Library (or with a work based on the Library)
on a volume of a storage or distribution medium does not bring
the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General
Public License instead of this License to a given copy of the
Library.  To do this, you must alter all the notices that refer
to this License, so that they refer to the ordinary GNU General
Public License, version 2, instead of to this License.  (If a
newer version than version 2 of the ordinary GNU General Public
License has appeared, then you can specify that version instead
if you wish.)  Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for
that copy, so the ordinary GNU General Public License applies to
all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of
the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or
derivative of it, under Section 2) in object code or executable
form under the terms of Sections 1 and 2 above provided that you
accompany it with the complete corresponding machine-readable
source code, which must be distributed under the terms of
Sections 1 and 2 above on a medium customarily used for software
interchange.

If distribution of object code is made by offering access to
copy from a designated place, then offering equivalent access to
copy the source code from the same place satisfies the
requirement to distribute the source code, even though third
parties are not compelled to copy the source along with the
object code.

5. A program that contains no derivative of any portion of the
Library, but is designed to work with the Library by being
compiled or linked with it, is called a "work that uses the
Library".  Such a work, in isolation, is not a derivative work
of the Library, and therefore falls outside the scope of this
License.

However, linking a "work that uses the Library" with the Library
creates an executable that is a derivative of the Library
(because it contains portions of the Library), rather than a
"work that uses the library".  The executable is therefore
covered by this License. Section 6 states terms for distribution
of such executables.

When a "work that uses the Library" uses material from a header
file that is part of the Library, the object code for the work
may be a derivative work of the Library even though the source
code is not. Whether this is true is especially significant if
```

```
the work can be linked without the Library, or if the work is
itself a library.  The threshold for this to be true is not
precisely defined by law.

If such an object file uses only numerical parameters, data
structure layouts and accessors, and small macros and small
inline functions (ten lines or less in length), then the use of
the object file is unrestricted, regardless of whether it is
legally a derivative work.  (Executables containing this object
code plus portions of the Library will still fall under Section
6.)

Otherwise, if the work is a derivative of the Library, you may
distribute the object code for the work under the terms of
Section 6. Any executables containing that work also fall under
Section 6, whether or not they are linked directly with the
Library itself.

6. As an exception to the Sections above, you may also compile
or link a "work that uses the Library" with the Library to
produce a work containing portions of the Library, and
distribute that work under terms of your choice, provided that
the terms permit modification of the work for the customer's own
use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that
the Library is used in it and that the Library and its use are
covered by this License.  You must supply a copy of this
License.  If the work during execution displays copyright
notices, you must include the copyright notice for the Library
among them, as well as a reference directing the user to the
copy of this License.  Also, you must do one of these things:

    a) Accompany the work with the complete corresponding
    machine-readable source code for the Library including whatever
    changes were used in the work (which must be distributed under
    Sections 1 and 2 above); and, if the work is an executable linked
    with the Library, with the complete machine-readable "work that
    uses the Library", as object code and/or source code, so that the
    user can modify the Library and then relink to produce a modified
    executable containing the modified Library.  (It is understood
    that the user who changes the contents of definitions files in the
    Library will not necessarily be able to recompile the application
    to use the modified definitions.)

    b) Accompany the work with a written offer, valid for at
    least three years, to give the same user the materials
    specified in Subsection 6a, above, for a charge no more
    than the cost of performing this distribution.

    c) If distribution of the work is made by offering access to copy
    from a designated place, offer equivalent access to copy the above
    specified materials from the same place.

    d) Verify that the user has already received a copy of these
    materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it.  However, as a special
exception, the source code distributed need not include anything
that is normally distributed (in either source or binary form)
with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that
component itself accompanies the executable.

It may happen that this requirement contradicts the license
restrictions of other proprietary libraries that do not normally
accompany the operating system.  Such a contradiction means you
cannot use both them and the Library together in an executable
that you distribute.

7. You may place library facilities that are a work based on the
```

Library side-by-side in a single library together with other
library facilities not covered by this License, and distribute
such a combined library, provided that the separate distribution
of the work based on the Library and of the other library
facilities is otherwise permitted, and provided that you do
these two things:

    a) Accompany the combined library with a copy of the same work
    based on the Library, uncombined with any other library
    facilities. This must be distributed under the terms of the
    Sections above.

    b) Give prominent notice with the combined library of the fact
    that part of it is a work based on the Library, and explaining
    where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or
distribute the Library except as expressly provided under this
License. Any attempt otherwise to copy, modify, sublicense,
link with, or distribute the Library is void, and will
automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you
under this License will not have their licenses terminated so
long as such parties remain in full compliance.

9. You are not required to accept this License, since you have
not signed it. However, nothing else grants you permission to
modify or distribute the Library or its derivative works. These
actions are prohibited by law if you do not accept this
License. Therefore, by modifying or distributing the Library
(or any work based on the Library), you indicate your acceptance
of this License to do so, and all its terms and conditions for
copying, distributing or modifying the Library or works based on
it.

10. Each time you redistribute the Library (or any work based on
the Library), the recipient automatically receives a license
from the original licensor to copy, distribute, link with or
modify the Library subject to these terms and conditions. You
may not impose any further restrictions on the recipients'
exercise of the rights granted herein. You are not responsible
for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of
patent infringement or for any other reason (not limited to
patent issues), conditions are imposed on you (whether by court
order, agreement or otherwise) that contradict the conditions of
this License, they do not excuse you from the conditions of this
License. If you cannot distribute so as to satisfy
simultaneously your obligations under this License and any other
pertinent obligations, then as a consequence you may not
distribute the Library at all. For example, if a patent license
would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you,
then the only way you could satisfy both it and this License
would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable
under any particular circumstance, the balance of the section is
intended to apply, and the section as a whole is intended to
apply in other circumstances.

It is not the purpose of this section to induce you to infringe
any patents or other property right claims or to contest
validity of any such claims; this section has the sole purpose
of protecting the integrity of the free software distribution
system which is implemented by public license practices. Many
people have made generous contributions to the wide range of
software distributed through that system in reliance on
consistent application of that system; it is up to the
author/donor to decide if he or she is willing to distribute
software through any other system and a licensee cannot impose
that choice.

This section is intended to make thoroughly clear what is
believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted
in certain countries either by patents or by copyrighted
interfaces, the original copyright holder who places the Library
under this License may add an explicit geographical distribution
limitation excluding those countries, so that distribution is
permitted only in or among countries not thus excluded. In such
case, this License incorporates the limitation as if written in
the body of this License.

13. The Free Software Foundation may publish revised and/or new
versions of the Library General Public License from time to
time. Such new versions will be similar in spirit to the present
version, but may differ in detail to address new problems or
concerns.

Each version is given a distinguishing version number. If the
Library specifies a version number of this License which applies
to it and "any later version", you have the option of following
the terms and conditions either of that version or of any later
version published by the Free Software Foundation. If the
Library does not specify a license version number, you may
choose any version ever published by the Free Software
Foundation.

14. If you wish to incorporate parts of the Library into other
free programs whose distribution conditions are incompatible
with these, write to the author to ask for permission. For
software which is copyrighted by the Free Software Foundation,
write to the Free Software Foundation; we sometimes make
exceptions for this. Our decision will be guided by the two
goals of preserving the free status of all derivatives of our
free software and of promoting the sharing and reuse of software
generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR
OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL
DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the
greatest possible use to the public, we recommend making it free
software that everyone can redistribute and change. You can do
so by permitting redistribution under these terms (or,
alternatively, under the terms of the ordinary General Public
License).

To apply these terms, attach the following notices to the
library. It is safest to attach them to the start of each

```
source file to most effectively convey the exclusion of
warranty; and each file should have at least the "copyright"
line and a pointer to where the full notice is found.

    <one line to give the library's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Library General Public
    License as published by the Free Software Foundation; either
    version 2 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Library General Public License for more details.

    You should have received a copy of the GNU Library General Public
    License along with this library; if not, write to the Free
    Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

  <signature of Ty Coon>, 1 April 1990
  Ty Coon, President of Vice

That's all there is to it!
```

4.5 The Open Group and DEC License

```
/************************************************************************

Copyright 1987, 1988, 1998  The Open Group

Permission to use, copy, modify, distribute, and sell this software and its
documentation for any purpose is hereby granted without fee, provided that
the above copyright notice appear in all copies and that both that
copyright notice and this permission notice appear in supporting
documentation.

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
OPEN GROUP BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of The Open Group shall not be
used in advertising or otherwise to promote the sale, use or other dealings
in this Software without prior written authorization from The Open Group.

Copyright 1987 by Digital Equipment Corporation, Maynard, Massachusetts.

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Digital not be
```

```
used in advertising or publicity pertaining to distribution of the
software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL
DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.

******************************************************************/
```

Chapter 5

Cross-Platform Development Tips

This chapter describes some general tips related to cross-platform development.

5.1 Include Files

The main include file is "wx/wx.h"; this includes the most commonly used modules of wxWidgets.

To save on compilation time, include only those header files relevant to the source file. If you are using **precompiled** headers, you should include the following section before any other includes:

```
// For compilers that support precompilation, includes "wx.h".
#include <wx/wxprec.h>

#ifdef __BORLANDC__
#   pragma hdrstop
#endif

#ifndef WX_PRECOMP
    // Include your minimal set of headers here, or wx.h
#   include <wx/wx.h>
#endif

... now your other include files ...
```

The file "wx/wxprec.h" includes "wx/wx.h". Although this incantation may seem quirky, it is in fact the end result of a lot of experimentation, and several Windows compilers to use precompilation which is largely automatic for compilers with necessary support. Currently it is used for Visual C++ (including embedded Visual C++), Borland C++ and newer versions of GCC. Some compilers might need extra work from the application developer to set the build environment up as necessary for the support.

5.2 Libraries

All ports of wxWidgets can create either a **static** library or a **shared** library.

When a program is linked against a *static* library, the machine code from the object files for any external functions used by the program is copied from the library into the final executable.

Shared libraries are handled with a more advanced form of linking, which makes the executable file smaller. They use the extension ".so" (Shared Object) under Linux and ".dll" (Dynamic Link Library) under Windows.

An executable file linked against a shared library contains only a small table of the functions it requires, instead of the complete machine code from the object files for the external functions. Before the executable file starts running, the machine code for the external functions is copied into memory from the shared library file on disk by the operating system - a process referred to as *dynamic* linking.

Dynamic linking makes executable files smaller and saves disk space, because one copy of a library can be shared between multiple programs. Most operating systems also provide a virtual memory mechanism which allows one copy of a shared library in physical memory to be used by all running programs, saving memory as well as disk space.

Furthermore, shared libraries make it possible to update a library without recompiling the programs which use it (provided the interface to the library does not change).

wxWidgets can also be built in **multilib** and **monolithic** variants. See the Library List for more information on these.

5.3 Configuration

When using project files and makefiles directly to build wxWidgets, options are configurable in the file "wx/XX↩X/setup.h" where XXX is the required platform (such as msw, motif, gtk, mac).

Some settings are a matter of taste, some help with platform-specific problems, and others can be set to minimize the size of the library. Please see the "setup.h" file and "install.txt" files for details on configuration.

When using the "configure" script to configure wxWidgets (on Unix and other platforms where configure is available), the corresponding "setup.h" files are generated automatically along with suitable makefiles.

When using the RPM packages (or DEB or other forms of *binaries*) for installing wxWidgets on Linux, a correct "setup.h" is shipped in the package and this must not be changed.

5.4 Makefiles

On Microsoft Windows, wxWidgets has a different set of makefiles for each compiler, because each compiler's 'make' tool is slightly different. Popular Windows compilers that we cater for, and the corresponding makefile extensions, include: Microsoft Visual C++ (.vc), Borland C++ (.bcc) and MinGW/Cygwin (.gcc). Makefiles are provided for the wxWidgets library itself, samples, demos, and utilities.

On Linux, Mac and OS/2, you use the 'configure' command to generate the necessary makefiles. You should also use this method when building with MinGW/Cygwin on Windows.

We also provide project files for some compilers, such as Microsoft VC++. However, we recommend using makefiles to build the wxWidgets library itself, because makefiles can be more powerful and less manual intervention is required.

On Windows using a compiler other than MinGW/Cygwin, you would build the wxWidgets library from the "build/msw" directory which contains the relevant makefiles.

On Windows using MinGW/Cygwin, and on Unix, OS X and OS/2, you invoke 'configure' (found in the top-level of the wxWidgets source hierarchy), from within a suitable empty directory for containing makefiles, object files and libraries.

For details on using makefiles, configure, and project files, please see "docs/xxx/install.txt" in your distribution, where "xxx" is the platform of interest, such as msw, gtk, x11, mac.

All wxWidgets makefiles are generated using Bakefile http://www.bakefile.org/. wxWidgets also provides (in the "build/bakefiles/wxpresets" folder) the wxWidgets bakefile presets. These files allow you to create bakefiles for your own wxWidgets-based applications very easily.

5.5 Windows Resource Files

wxWidgets application compilation under MS Windows requires at least one extra file: a resource file.

The least that must be defined in the Windows resource file (extension RC) is the following statement:

```
#include "wx/msw/wx.rc"
```

which includes essential internal wxWidgets definitions. The resource script may also contain references to icons, cursors, etc., for example:

```
wxicon icon wx.ico
```

The icon can then be referenced by name when creating a frame icon. See the Microsoft Windows SDK documentation.

Note

Include "wx.rc" *after* any ICON statements so programs that search your executable for icons (such as the Program Manager) find your application icon first.

5.6 Allocating and Deleting wxWidgets Objects

In general, classes derived from wxWindow must dynamically allocated with *new* and deleted with *delete*. If you delete a window, all of its children and descendants will be automatically deleted, so you don't need to delete these descendants explicitly.

When deleting a frame or dialog, use **Destroy** rather than **delete** so that the wxWidgets delayed deletion can take effect. This waits until idle time (when all messages have been processed) to actually delete the window, to avoid problems associated with the GUI sending events to deleted windows.

In general wxWindow-derived objects should always be allocated on the heap as wxWidgets will destroy them itself. The only, but important, exception to this rule are the modal dialogs, i.e. wxDialog objects which are shown using wxDialog::ShowModal() method. They may be allocated on the stack and, indeed, usually are local variables to ensure that they are destroyed on scope exit as wxWidgets does not destroy them unlike with all the other windows. So while it is still possible to allocate modal dialogs on the heap, you should still destroy or delete them explicitly in this case instead of relying on wxWidgets doing it.

If you decide to allocate a C++ array of objects (such as wxBitmap) that may be cleaned up by wxWidgets, make sure you delete the array explicitly before wxWidgets has a chance to do so on exit, since calling *delete* on array members will cause memory problems.

wxColour can be created statically: it is not automatically cleaned up and is unlikely to be shared between other objects; it is lightweight enough for copies to be made.

Beware of deleting objects such as a wxPen or wxBitmap if they are still in use. Windows is particularly sensitive to this, so make sure you make calls like wxDC::SetPen(wxNullPen) or wxDC::SelectObject(wxNullBitmap) before deleting a drawing object that may be in use. Code that doesn't do this will probably work fine on some platforms, and then fail under Windows.

5.7 Architecture Dependency

A problem which sometimes arises from writing multi-platform programs is that the basic C types are not defined the same on all platforms. This holds true for both the length in bits of the standard types (such as int and long) as well as their byte order, which might be little endian (typically on Intel computers) or big endian (typically on some Unix workstations). wxWidgets defines types and macros that make it easy to write architecture independent code. The types are:

wxInt32, wxInt16, wxInt8, wxUint32, wxUint16 = wxWord, wxUint8 = wxByte

where wxInt32 stands for a 32-bit signed integer type etc. You can also check which architecture the program is compiled on using the wxBYTE_ORDER define which is either wxBIG_ENDIAN or wxLITTLE_ENDIAN (in the future maybe wxPDP_ENDIAN as well).

The macros handling bit-swapping with respect to the applications endianness are described in the Byte Order section.

5.8 Conditional Compilation

One of the purposes of wxWidgets is to reduce the need for conditional compilation in source code, which can be messy and confusing to follow. However, sometimes it is necessary to incorporate platform-specific features (such as metafile use under MS Windows). The wxUSE Preprocessor Symbols symbols listed in the file `setup.h` may be used for this purpose, along with any user-supplied ones.

5.9 C++ Issues

The following documents some miscellaneous C++ issues.

5.9.1 Templates

wxWidgets does not use templates (except for some advanced features that are switched off by default) since it is a notoriously unportable feature.

5.9.2 Runtime Type Information (RTTI)

wxWidgets does not use C++ run-time type information since wxWidgets provides its own run-time type information system, implemented using macros.

5.9.3 Precompiled Headers

Some compilers, such as Borland C++ and Microsoft C++, support precompiled headers. This can save a great deal of compiling time. The recommended approach is to precompile `"wx.h"`, using this precompiled header for compiling both wxWidgets itself and any wxWidgets applications. For Windows compilers, two dummy source files are provided (one for normal applications and one for creating DLLs) to allow initial creation of the precompiled header.

However, there are several downsides to using precompiled headers. One is that to take advantage of the facility, you often need to include more header files than would normally be the case. This means that changing a header file will cause more recompilations (in the case of wxWidgets, everything needs to be recompiled since everything includes `"wx.h"`).

A related problem is that for compilers that don't have precompiled headers, including a lot of header files slows down compilation considerably. For this reason, you will find (in the common X and Windows parts of the library) conditional compilation that under Unix, includes a minimal set of headers; and when using Visual C++, includes `"wx.h"`. This should help provide the optimal compilation for each compiler, although it is biased towards the precompiled headers facility available in Microsoft C++.

5.10 File Handling

When building an application which may be used under different environments, one difficulty is coping with documents which may be moved to different directories on other machines. Saving a file which has pointers to full pathnames is going to be inherently unportable.

One approach is to store filenames on their own, with no directory information. The application then searches into a list of standard paths (platform-specific) through the use of wxStandardPaths.

Eventually you may want to use also the wxPathList class.

Nowadays the limitations of DOS 8+3 filenames doesn't apply anymore. Most modern operating systems allow at least 255 characters in the filename; the exact maximum length, as well as the characters allowed in the filenames,

are OS-specific so you should try to avoid extremely long ($>$ 255 chars) filenames and/or filenames with non-ANSI characters.

Another thing you need to keep in mind is that all Windows operating systems are case-insensitive, while Unix operating systems (Linux, Mac, etc) are case-sensitive.

Also, for text files, different OSes use different End Of Lines (EOL). Windows uses CR+LF convention, Linux uses LF only, Mac CR only.

The wxTextFile, wxTextInputStream, wxTextOutputStream classes help to abstract from these differences. Of course, there are also 3rd party utilities such as `dos2unix` and `unix2dos` which do the EOL conversions.

See also the Files and Directories section of the reference manual for the description of miscellaneous file handling functions.

5.11 Reducing Programming Errors

5.11.1 Use ASSERT

It is good practice to use ASSERT statements liberally, that check for conditions that should or should not hold, and print out appropriate error messages.

These can be compiled out of a non-debugging version of wxWidgets and your application. Using ASSERT is an example of 'defensive programming': it can alert you to problems later on.

See wxASSERT() for more info.

5.11.2 Use wxString in Preference to Character Arrays

Using wxString can be much safer and more convenient than using `wxChar*`.

You can reduce the possibility of memory leaks substantially, and it is much more convenient to use the overloaded operators than functions such as `strcmp`. wxString won't add a significant overhead to your program; the overhead is compensated for by easier manipulation (which means less code).

The same goes for other data types: use classes wherever possible.

5.12 GUI Design

- **Use Sizers:** Don't use absolute panel item positioning if you can avoid it. Every platform's native controls have very different sizes. Consider using the Sizers Overview instead.
- **Use wxWidgets Resource Files:** Use `XRC` (wxWidgets resource files) where possible, because they can be easily changed independently of source code. See the XML Based Resource System (XRC) for more info.

5.13 Debugging

5.13.1 Positive Thinking

It is common to blow up the problem in one's imagination, so that it seems to threaten weeks, months or even years of work. The problem you face may seem insurmountable: but almost never is. Once you have been programming for some time, you will be able to remember similar incidents that threw you into the depths of despair. But remember, you always solved the problem, somehow!

Perseverance is often the key, even though a seemingly trivial problem can take an apparently inordinate amount of time to solve. In the end, you will probably wonder why you worried so much. That's not to say it isn't painful at the time. Try not to worry – there are many more important things in life.

5.13.2 Simplify the Problem

Reduce the code exhibiting the problem to the smallest program possible that exhibits the problem. If it is not possible to reduce a large and complex program to a very small program, then try to ensure your code doesn't hide the problem (you may have attempted to minimize the problem in some way: but now you want to expose it).

With luck, you can add a small amount of code that causes the program to go from functioning to non-functioning state. This should give a clue to the problem. In some cases though, such as memory leaks or wrong deallocation, this can still give totally spurious results!

5.13.3 Use a Debugger

This sounds like facetious advice, but it is surprising how often people don't use a debugger. Often it is an overhead to install or learn how to use a debugger, but it really is essential for anything but the most trivial programs.

5.13.4 Use Logging Functions

There is a variety of logging functions that you can use in your program: see Logging.

Using tracing statements may be more convenient than using the debugger in some circumstances (such as when your debugger doesn't support a lot of debugging code, or you wish to print a bunch of variables).

5.13.5 Use the wxWidgets Debugging Facilities

You can use wxDebugContext to check for memory leaks and corrupt memory: in fact in debugging mode, wx↩ Widgets will automatically check for memory leaks at the end of the program if wxWidgets is suitably configured. Depending on the operating system and compiler, more or less specific information about the problem will be logged.

You should also use Debugging macros as part of a "defensive programming" strategy, scattering wxASSERT()s liberally to test for problems in your code as early as possible. Forward thinking will save a surprising amount of time in the long run.

See the Debugging for further information.

Chapter 6

Introduction

6.1 What is wxWidgets?

wxWidgets is an open source C++ framework allowing to write cross-platform GUI applications with native look and feel in C++ and other languages.

wxWidgets was originally developed by Julian Smart at the Artificial Intelligence Applications Institute, University of Edinburgh, for internal use, and was first made publicly available in 1992, with a vastly improved version 2 released in 1999. The last major version of the library is 3 and was released in 2013. Currently wxWidgets is developed and maintained by Julian Smart, Vadim Zeitlin, Stefan Csomor, Robert Roebling, Vaclav Slavik and many others.

More information about wxWidgets is available on its web site at http://www.wxwidgets.org.

6.2 Why choose wxWidgets?

Compared to the other similar libraries, wxWidgets is:

1. The only C++ GUI library built by wrapping native GUI widgets which results in the best user experience on each platform.
2. Written using only the standard C++ and doesn't rely on any custom extensions or preprocessing.
3. Open source and free for use in both open source and commercial projects.

wxWidgets provides a simple, easy to learn, yet very rich API. It is also mature and stable, and the applications written using wxWidgets 2.0 pre-releases almost 20 years ago can still be built today with wxWidgets 3 almost unchanged. wxWidgets has a large, active and friendly community of people, including both the users and developers of the library. It is also available now for more than a dozen other languages, including Python, Perl, Ruby, Lua, Haskell, D, Erlang, PHP, in addition to C++.

It is impossible to sum up everything included in wxWidgets in a few paragraphs, but here are some of the benefits:

- Available on all major desktop platforms.
- Free for any use.
- Source is available and easy to read and modify if necessary.
- Over 100 example programs.
- Extensive documentation (almost 200,000 lines of it).
- Straightforward API.
- Simple but powerful layout system.

- Run-time loadable or compile-time embeddable resources.
- Flexible event system.
- All the usual and quite a few of more rare GUI controls.
- And also all the standard dialogs.
- 2D path-based drawing API with full support for transparency.
- Built-in support for many file formats (BMP, PNG, JPEG, GIF, XPM, PNM, PCX, TGA, ...).
- Printing, help, clipboard, drag-and-drop, document/view, ... support.
- Integration with the native platform HTML rendering engine.
- Dockable windows framework.
- Word processor-like widget.
- Powerful text editing widget with syntax highlighting.
- And much, much more...

6.3 wxWidgets Requirements

wxWidgets first-tier "ports", ie implementations of wxWidgets API, are:

- wxMSW: This is the native port for Microsoft Windows systems (from Windows 95 up to Windows 8.1 with at least Windows XP being recommended), either 32 or 64 bits. The primarily supported compilers are Microsoft Visual C++ (versions 6 up to 2013 are supported, at least 2005 is recommended) and GNU g++ (either from the traditional MinGW, TDM-GCC or MinGW-w64 distributions).

- wxGTK: wxGTK2 and wxGTK3 are the ports using GTK+ library version 2.x and 3.x respectively. They are very similar, with wxGTK2 being, however, more mature. Both ports work on almost any Unix system (Linux, FreeBSD, OpenBSD, NetBSD, Solaris, AIX, ...) and require GTK+ 2.6 or later or GTK+ 3.x. The primary supported compiler is GNU g++.

- wxOSX: wxOSX/Cocoa is the primary port for Apple computers, replacing the older and now deprecated wxOSX/Carbon port. wxOSX supports either PowerPC or Intel Macs running OS X 10.5 or higher and can be compiled in either 32 or 64 bits using Apple Developer Tools (both GNU g++ and clang are supported).

Other platforms (iOS, Windows CE, OS/2), compilers (Borland C++ under Windows, Sun CC, HP-UX aCC, IBM xlC or SGI mipsPro under Unix) and ports (wxOSX/Carbon, wxGTK1, wxX11, wxDFB, wxPM...) are also supported but to a lesser extent. Please see the platform details page for more information.

There are no CPU speed requirements but the faster (and more) CPU(s) you have, the faster the library will compile. You do need to have enough RAM, especially under Windows platforms, to avoid running out of memory during link phase. Depending on the compiler used, you may need at least 4GB to be able to link. Under all platforms it's recommended to have large amounts of free hard disk space. The exact amount needed depends on the port, compiler and build configurations but at least 1GB and possibly more is required.

6.4 Where to get wxWidgets and support for it

The download links can be found at http://www.wxwidgets.org. The primary download location is https://sourceforge.net/downloads/wxwindows/ and there is also an FTP mirror at ftp://ftp.wxwidgets.org/pub/. Additionally, the latest version can always be retrieved from our version control system using either Subversion (http://svn.wxwidgets.org/svn/wx/wxWidgets/) or Git (https://github.com/wxWidgets/wxWidgets).

wxWidgets documentation that you are reading is also available online at http://docs.wxwidgets.org/trunk/ and please also visit our wiki at http://wiki.wxwidgets.org/ for user-contributed contents.

And if you have any questions, you can join wxWidgets community using

- Web-based wxForum.
- Mailing lists.
- #wxwidgets IRC channel.
- Or asking questions with wxwidgets tag on http://stackoverflow.com/

6.5 Platform Details

wxWidgets defines a common API across platforms, but uses the native graphical user interface (GUI) on each platform, so your program will take on the native look and feel that users are familiar with.

Unfortunately native toolkits and hardware do not always support the functionality that the wxWidgets API requires. This chapter collects notes about differences among supported platforms and ports.

6.5.1 wxGTK

wxGTK is a port of wxWidgets using the GTK+ library. It makes use of GTK+'s native widgets wherever possible and uses wxWidgets' generic controls when needed. GTK+ itself has been ported to a number of systems, but so far only the original X11 version is supported. Support for other GTK+ backends is planned, such as the new DirectFB backend.

All work is being done on GTK+ version 2.0 and above. Support for GTK+ 1.2 will be deprecated in a later release.

You will need GTK+ 2.6 or higher which is available from:

http://www.gtk.org

The newer version of GTK+ you use, the more native widgets and features will be utilized. We have gone to great lengths to allow compiling wxWidgets applications with the latest version of GTK+, with the resulting binary working on systems even with a much earlier version of GTK+. You will have to ensure that the application is launched with lazy symbol binding for that.

In order to configure wxWidgets to compile wxGTK you will need use the `-with-gtk` argument to the `configure` script. This is the default for many systems.

GTK+ 1.2 can still be used, albeit discouraged. For that you can pass `-with-gtk=1` to the `configure` script.

Support for GTK+ 3 is available starting with wxWidgets 2.9.4, use `configure` option `-with-gtk=3` to enable it.

For further information, please see the files in `docs/gtk` in the distribution.

6.5.2 wxOSX

wxOSX/Cocoa

wxOSX/Cocoa is the currently recommended port of wxWidgets for the Macintosh OS platform. It requires OS X 10.7 or later and, unlike wxOSX/Carbon, fully supports 64 bit builds.

This is the default port when building wxOSX, but in order to select it explicitly you can use

```
configure --with-osx_cocoa
```

For further information, please see the files in `docs/osx` in the distribution.

wxOSX/Carbon

wxOSX/Carbon is an older port of wxWidgets for the Macintosh OS platform. Currently OS X 10.5 or higher are supported. wxOSX/Carbon can be compiled both using Apple's command line developer tools as well as Apple's Xcode IDE. wxOSX/Carbon supports Intel and PowerPC architectures and can be used to produce "universal binaries" in order create application which can run both architecture. Unfortunately, wxOSX/Carbon does not support any 64-bit architecture since Apple decided not to port its Carbon API entirely to 64-bit.

Note

Carbon has been deprecated by Apple as of OS X 10.5 and will likely be removed entirely in a future OS version. It's recommended you look into switching your app over to wxOSX/Cocoa as soon as possible.

To build wxWidgets using wxOSX/Carbon you need to do

```
configure --with-osx_carbon
```

For further information, please see the files in `docs/osx` in the distribution.

6.5.3 wxX11

wxX11 is a port of wxWidgets using X11 (The X Window System) as the underlying graphics backend. wxX11 draws its widgets using the wxUniversal widget set which is now part of wxWidgets. wxX11 is well-suited for a number of special applications such as those running on systems with few resources (PDAs) or for applications which need to use a special themed look.

In order to configure wxWidgets to compile wxX11 you will need to type:

```
configure --with-x11 --with-universal
```

For further information, please see the files in `docs/x11` in the distribution. There is also a page on the use of wxWidgets for embedded applications on the wxWidgets web site.

6.5.4 wxMotif

wxMotif is a port of wxWidgets for X11 systems using Motif libraries. Motif libraries provide a clean and fast user interface at the expense of the beauty and candy of newer interfaces like GTK.

For further information, please see the files in `docs/motif` in the distribution.

6.5.5 wxMSW

wxMSW is a port of wxWidgets for the Windows platforms (Windows XP and later are supported). wxMSW provides native look and feel for each Windows version. This port can be compiled with several compilers including Microsoft Studio VC++ 2003 or later, Borland 5.5, MinGW32, Cygwin as well as cross-compilation with a Linux-hosted MinGW32 tool chain.

For further information, please see the files in docs/msw in the distribution.

Resources and Application Icon

All applications using wxMSW should have a Windows resource file (.rc extension) and this file should include `include/wx/msw/wx.rc` file which defines resources used by wxWidgets itself.

Among other things, `wx.rc` defines some standard icons, all of which have names starting with the "wx" prefix. This normally ensures that any icons defined in the application's own resource file come before them in alphabetical

order which is important because Explorer (Windows shell) selects the first icon in alphabetical order to use as the application icon which is displayed when viewing its file in the file manager. So if all the icons defined in your application start with "x", "y" or "z", they won't be used by Explorer. To avoid this, ensure that the icon which is meant to be used as the main application icon has a name preceding "wxICON" in alphabetical order.

Themed Borders

Starting with wxWidgets 2.8.5, you can specify the `wxBORDER_THEME` style to have wxWidgets use a themed border. Using the default XP theme, this is a thin 1-pixel blue border, with an extra 1-pixel border in the window client background colour (usually white) to separate the client area's scrollbars from the border.

If you don't specify a border style for a wxTextCtrl in rich edit mode, wxWidgets now gives the control themed borders automatically, where previously they would take the Windows 95-style sunken border. Other native controls such as wxTextCtrl in non-rich edit mode, and wxComboBox already paint themed borders where appropriate. To use themed borders on other windows, such as wxPanel, pass the `wxBORDER_THEME` style, or (apart from wxPanel) pass no border style.

In general, specifying `wxBORDER_THEME` will cause a border of some kind to be used, chosen by the platform and control class. To leave the border decision entirely to wxWidgets, pass `wxBORDER_DEFAULT`. This is not to be confused with specifying `wxBORDER_NONE`, which says that there should definitely be *no* border.

Internal Border Implementation

The way that wxMSW decides whether to apply a themed border is as follows. The theming code calls wxWindow::GetBorder() to obtain a border. If no border style has been passed to the window constructor, GetBorder() calls GetDefaultBorder() for this window. If wxBORDER_THEME was passed to the window constructor, GetBorder() calls GetDefaultBorderForControl().

The implementation of wxWindow::GetDefaultBorder() on wxMSW calls wxWindow::CanApplyThemeBorder() which is a virtual function that tells wxWidgets whether a control can have a theme applied explicitly (some native controls already paint a theme in which case we should not apply it ourselves). Note that wxPanel is an exception to this rule because in many cases we wish to create a window with no border (for example, notebook pages). So wxPanel overrides GetDefaultBorder() in order to call the generic wxWindowBase::GetDefaultBorder(), returning wxBORDER_NONE.

wxWinCE

wxWinCE is the name given to wxMSW when compiled on Windows CE devices; most of wxMSW is common to Win32 and Windows CE but there are some simplifications, enhancements, and differences in behaviour.

For building instructions, see docs/msw/wince in the distribution, also the section about Visual Studio 2005 project files below. The rest of this section documents issues you need to be aware of when programming for Windows CE devices.

General Issues for wxWinCE

Mobile applications generally have fewer features and simpler user interfaces. Simply omit whole sizers, static lines and controls in your dialogs, and use comboboxes instead of listboxes where appropriate. You also need to reduce the amount of spacing used by sizers, for which you can use a macro such as this:

```
#if defined(__WXWINCE__)
    #define wxLARGESMALL(large,small) small
#else
    #define wxLARGESMALL(large,small) large
#endif

// Usage
topsizer->Add( CreateTextSizer( message ), 0, wxALL, wxLARGESMALL(10,0) );
```

There is only ever one instance of a Windows CE application running, and wxWidgets will take care of showing the current instance and shutting down the second instance if necessary.

You can test the return value of wxSystemSettings::GetScreenType() for a qualitative assessment of what kind of display is available, or use wxGetDisplaySize() if you need more information.

You can also use wxGetOsVersion to test for a version of Windows CE at run-time (see the next section). However, because different builds are currently required to target different kinds of device, these values are hard-wired according to the build, and you cannot dynamically adapt the same executable for different major Windows CE platforms. This would require a different approach to the way wxWidgets adapts its behaviour (such as for menubars) to suit the style of device.

See the "Life!" example (demos/life) for an example of an application that has been tailored for PocketPC and Smartphone use.

Note

Don't forget to have this line in your .rc file, as for desktop Windows applications:

```
#include "wx/msw/wx.rc"
```

Testing for WinCE SDKs

Use these preprocessor symbols to test for the different types of devices:

- **SMARTPHONE** Generic mobile devices with phone buttons and a small display
- **PDA** Generic mobile devices with no phone
- **HANDHELDPC** Generic mobile device with a keyboard
- **WXWINCE** Microsoft-powered Windows CE devices, whether PocketPC, Smartphone or Standard SDK
- **WIN32_PLATFORM_WFSP** Microsoft-powered smartphone
- **POCKETPC** Microsoft-powered PocketPC devices with touch-screen
- **WINCE_STANDARDSDK** Microsoft-powered Windows CE devices, for generic Windows CE applications
- **WINCE_NET** Microsoft-powered Windows CE .NET devices (_WIN32_WCE is 400 or greater)

wxGetOsVersion() will return these values:

- **wxWINDOWS_POCKETPC** The application is running under PocketPC.
- **wxWINDOWS_SMARTPHONE** The application is running under Smartphone.
- **wxWINDOWS_CE** The application is running under Windows CE (built with the Standard SDK).

Window sizing in wxWinCE

Top level windows (dialogs, frames) are created always full-screen. Fit() of sizers will not rescale top level windows but instead will scale window content.

If the screen orientation changes, the windows will automatically be resized so no further action needs to be taken (unless you want to change the layout according to the orientation, which you could detect in idle time, for example). When input panel (SIP) is shown, top level windows (frames and dialogs) resize accordingly (see wxTopLevelWindow::HandleSettingChange()).

Closing Top-level Windows in wxWinCE

You won't get a wxCloseEvent when the user clicks on the X in the titlebar on Smartphone and PocketPC; the window is simply hidden instead. However the system may send the event to force the application to close down.

Hibernation in wxWinCE

Smartphone and PocketPC will send a wxEVT_HIBERNATE to the application object in low memory conditions. Your application should release memory and close dialogs, and wake up again when the next wxEVT_ACTIVATE or wxEVT_ACTIVATE_APP message is received. (wxEVT_ACTIVATE_APP is generated whenever a wxEVT_ACTIVATE event is received in Smartphone and PocketPC, since these platforms do not support WM_ACTIVATEAPP.)

Hardware Buttons in wxWinCE

Special hardware buttons are sent to a window via the wxEVT_HOTKEY event under Smartphone and PocketPC. You should first register each required button with wxWindow::RegisterHotKey(), and unregister the button when you're done with it. For example:

```
win->RegisterHotKey(0, wxMOD_WIN, WXK_SPECIAL1);
win->UnregisterHotKey(0);
```

You may have to register the buttons in a wxEVT_ACTIVATE event handler since other applications will grab the buttons.

There is currently no method of finding out the names of the special buttons or how many there are.

Dialogs in wxWinCE

PocketPC dialogs have an OK button on the caption, and so you should generally not repeat an OK button on the dialog. You can add a Cancel button if necessary, but some dialogs simply don't offer you the choice (the guidelines recommend you offer an Undo facility to make up for it). When the user clicks on the OK button, your dialog will receive a wxID_OK event by default. If you wish to change this, call wxDialog::SetAffirmativeId() with the required identifier to be used. Or, override wxDialog::DoOK() (return false to have wxWidgets simply call Close to dismiss the dialog).

Smartphone dialogs do *not* have an OK button on the caption, and are closed using one of the two menu buttons. You need to assign these using wxTopLevelWindow::SetLeftMenu and wxTopLevelWindow::SetRightMenu(), for example:

```
#ifdef __SMARTPHONE__
    SetLeftMenu(wxID_OK);
    SetRightMenu(wxID_CANCEL, _("Cancel"));
#elif defined(__POCKETPC__)
    // No OK/Cancel buttons on PocketPC, OK on caption will close
#else
    topsizer->Add( CreateButtonSizer( wxOK|wxCANCEL ), 0, wxEXPAND |
      wxALL, 10 );
#endif
```

For implementing property sheets (flat tabs), use a wxNotebook with wxNB_FLAT|wxNB_BOTTOM and have the notebook left, top and right sides overlap the dialog by about 3 pixels to eliminate spurious borders. You can do this by using a negative spacing in your sizer Add() call. The cross-platform property sheet dialog wxPropertySheetDialog is provided, to show settings in the correct style on PocketPC and on other platforms.

Notifications (bubble HTML text with optional buttons and links) will also be implemented in the future for PocketPC.

Modeless dialogs probably don't make sense for PocketPC and Smartphone, since frames and dialogs are normally full-screen, and a modeless dialog is normally intended to co-exist with the main application frame.

Menubars and Toolbars in PocketPC

On PocketPC, a frame must always have a menubar, even if it's empty. An empty menubar/toolbar is automatically provided for dialogs, to hide any existing menubar for the duration of the dialog.

Menubars and toolbars are implemented using a combined control, but you can use essentially the usual wxWidgets API; wxWidgets will combine the menubar and toolbar. However, there are some restrictions:

- You must create the frame's primary toolbar with wxFrame::CreateToolBar(), because this uses the special wxToolMenuBar class (derived from wxToolBar) to implement the combined toolbar and menubar. Otherwise,

you can create and manage toolbars using the wxToolBar class as usual, for example to implement an optional formatting toolbar above the menubar as Pocket Word does. But don't assign a wxToolBar to a frame using SetToolBar - you should always use CreateToolBar for the main frame toolbar.

- Deleting and adding tools to wxToolMenuBar after Realize is called is not supported.
- For speed, colours are not remapped to the system colours as they are in wxMSW. Provide the tool bitmaps either with the correct system button background, or with transparency (for example, using XPMs).
- Adding controls to wxToolMenuBar is not supported. However, wxToolBar supports controls.

Unlike in all other ports, a wxDialog has a wxToolBar automatically created for you. You may either leave it blank, or access it with wxDialog::GetToolBar() and add buttons, then calling wxToolBar::Realize(). You cannot set or recreate the toolbar.

Menubars and Toolbars in Smartphone

On Smartphone, there are only two menu buttons, so a menubar is simulated using a nested menu on the right menu button. Any toolbars are simply ignored on Smartphone.

Closing Windows in wxWinCE

The guidelines state that applications should not have a Quit menu item, since the user should not have to know whether an application is in memory or not. The close button on a window does not call the window's close handler; it simply hides the window. However, the guidelines say that the Ctrl+Q accelerator can be used to quit the application, so wxWidgets defines this accelerator by default and if your application handles wxID_EXIT, it will do the right thing.

Context Menus in wxWinCE

To enable context menus in PocketPC, you currently need to call wxWindow::EnableContextMenu(), a wxWinCE-only function. Otherwise the context menu event (wxContextMenuEvent) will never be sent. This API is subject to change.

Context menus are not supported in Smartphone.

Control Differences on wxWinCE

These controls and styles are specific to wxWinCE:

- wxTextCtrl The `wxTE_CAPITALIZE` style causes a CAPEDIT control to be created, which capitalizes the first letter.

These controls are missing from wxWinCE:

- MDI classes MDI is not supported under Windows CE.
- wxMiniFrame Not supported under Windows CE.

Tooltips are not currently supported for controls, since on PocketPC controls with tooltips are distinct controls, and it will be hard to add dynamic tooltip support.

Control borders on PocketPC and Smartphone should normally be specified with `wxBORDER_SIMPLE` instead of `wxBORDER_SUNKEN`. Controls will usually adapt appropriately by virtue of their GetDefaultBorder() function, but if you wish to specify a style explicitly you can use `wxDEFAULT_CONTROL_BORDER` which will give a simple border on PocketPC and Smartphone, and the sunken border on other platforms.

Online Help in wxWinCE

You can use the help controller wxWinceHelpController which controls simple .htm files, usually installed in the Windows directory. See the Windows CE reference for how to format the HTML files.

Installing your PocketPC and Smartphone Applications

To install your application, you need to build a CAB file using the parameters defined in a special .inf file. The CabWiz program in your SDK will compile the CAB file from the .inf file and files that it specifies.

For delivery, you can simply ask the user to copy the CAB file to the device and execute the CAB file using File Explorer. Or, you can write a program for the desktop PC that will find the ActiveSync Application Manager and install the CAB file on the device, which is obviously much easier for the user.

Here are some links that may help.

- A setup builder that takes CABs and builds a setup program is at http://www.eskimo.com/~scottlu/win/index.html.
- Sample installation files can be found in `Windows CE Tools/wce420/POCKET PC 2003/Samples/Win32/AppInst`.
- An installer generator using wxPython can be found at http://ppcquicksoft.iespana.es/ppcquicksoft/myinstall.html.
- Miscellaneous Windows CE resources can be found at http://www.orbworks.com/pcce/resources.html.
- Installer creation instructions with a setup.exe for installing to PPC can be found at http://www.pocketpcdn.com/articles/creatingsetup.html.
- Microsoft instructions are at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnppcgen/html/msdn_cabwiz.asp?frame=true
- Troubleshooting WinCE application installations: http://support.microsoft.com/default.aspx?scid=KB;en-us;q181007

You may also check out `demos/life/setup/wince` which contains scripts to create a PocketPC installation for ARM-based devices. In particular, `build.bat` builds the distribution and copies it to a directory called `Deliver`.

wxFileDialog in PocketPC

Allowing the user to access files on memory cards, or on arbitrary parts of the filesystem, is a pain; the standard file dialog only shows folders under My Documents or folders on memory cards (not the system or card root directory, for example). This is a known problem for PocketPC developers.

If you need a file dialog that allows access to all folders, you can use wxGenericFileDialog instead. You will need to include `wx/generic/filedlgg.h`.

Embedded Visual C++ Issues

Run-time type information

If you wish to use runtime type information (RTTI) with eVC++ 4, you need to download an extra library, `ccrtrtti.lib`, and link with it. At the time of writing you can get it from here:

```
http://support.microsoft.com/kb/830482/en-us
```

Otherwise you will get linker errors similar to this:

```
wxwince26d.lib(control.obj) : error LNK2001: unresolved external symbol "const type_info::`vftable'" (??_7type
```

Windows Mobile 5.0 emulator

Note that there is no separate emulator configuration for Windows Mobile 5.0: the emulator runs the ARM code directly.

Visual Studio 2005 project files

Unfortunately, Visual Studio 2005, required to build Windows Mobile 5.0 applications, doesn't do a perfect job of converting the project files from eVC++ format.

When you have converted the wxWidgets workspace, edit the configuration properties for each configuration and in the Librarian, add a relative path

```
..\..\lib
```

to each library path. For example:

```
..\$(PlatformName)\$(ConfigurationName)\wx_mono.lib
```

Then, for a sample you want to compile, edit the configuration properties and make sure

```
..\..\lib\$(PlatformName)\$(ConfigurationName)
```

is in the Linker/General/Additional Library Directories property. Also change the Linker/Input/Additional Dependencies property to something like

```
coredll.lib wx_mono.lib wx_wxjpeg.lib wx_wxpng.lib wx_wxzlib.lib wx_wxexpat.lib commctrl.lib winsock.lib winin
```

since the library names in the wxWidgets workspace were changed by VS 2005.

Alternately, you could edit all the names to be identical to the original eVC++ names, but this will probably be more fiddly.

Remaining Issues

These are some of the remaining problems to be sorted out, and features to be supported.

- **Windows Mobile 5 issues.** It is not possible to get the HMENU for the command bar on Mobile 5, so the menubar functions need to be rewritten to get the individual menus without use of a menubar handle. Also the new Mobile 5 convention of using only two menus (and no bitmap buttons) needs to be considered.
- **Sizer speed.** Particularly for dialogs containing notebooks, layout seems slow. Some analysis is required.
- **Notification boxes.** The balloon-like notification messages, and their icons, should be implemented. This will be quite straightforward.
- **SIP size.** We need to be able to get the area taken up by the SIP (input panel), and the remaining area, by calling SHSipInfo. We also may need to be able to show and hide the SIP programmatically, with SHSip↩ Preference. See also the *Input Dialogs* topic in the *Programming Windows CE* guide for more on this, and how to have dialogs show the SIP automatically using the `WC_SIPREF` control.
- **wxStaticBitmap.** The About box in the "Life!" demo shows a bitmap that is the correct size on the emulator, but too small on a VGA Pocket Loox device.
- **wxStaticLine.** Lines don't show up, and the documentation suggests that missing styles are implemented with `WM_PAINT`.
- **HTML control.** PocketPC has its own HTML control which can be used for showing local pages or navigating the web. We should create a version of wxHtmlWindow that uses this control, or have a separately-named control (wxHtmlCtrl), with a syntax as close as possible to wxHtmlWindow.
- **Tooltip control.** PocketPC uses special TTBUTTON and TTSTATIC controls for adding tooltips, with the tooltip separated from the label with a double tilde. We need to support this using SetToolTip.(Unfortunately it does not seem possible to dynamically remove the tooltip, so an extra style may be required.)
- **Focus.** In the wxPropertySheetDialog demo on Smartphone, it's not possible to navigate between controls. The focus handling in wxWidgets needs investigation. See in particular src/common/containr.cpp, and note that the default OnActivate handler in src/msw/toplevel.cpp sets the focus to the first child of the dialog.

- **OK button.** We should allow the OK button on a dialog to be optional, perhaps by using `wxCLOSE_BOX` to indicate when the OK button should be displayed.
- **Dynamic adaptation.** We should probably be using run-time tests more than preprocessor tests, so that the same WinCE application can run on different versions of the operating system.
- **Modeless dialogs.** When a modeless dialog is hidden with the OK button, it doesn't restore the frame's menubar. See for example the find dialog in the dialogs sample. However, the menubar is restored if pressing Cancel (the window is closed). This reflects the fact that modeless dialogs are not very useful on Windows CE; however, we could perhaps destroy/restore a modeless dialog's menubar on deactivation and activation.
- **Home screen plugins.** Figure out how to make home screen plugins for use with wxWidgets applications (see http://www.codeproject.com/ce/CTodayWindow.asp for inspiration). Although we can't use wxWidgets to create the plugin (too large), we could perhaps write a generic plugin that takes registry information from a given application, with options to display information in a particular way using icons and text from a specified location.
- **Further abstraction.** We should be able to abstract away more of the differences between desktop and mobile applications, in particular for sizer layout.
- **Dialog captions.** The blue, bold captions on dialogs - with optional help button - should be catered for, either by hard-wiring the capability into all dialogs and panels, or by providing a standard component and sizer.

6.5.6 Native Toolkit Documentation

It's sometimes useful to interface directly with the underlying toolkit used by wxWidgets to e.g. use toolkit-specific features. In such case (or when you want to e.g. write a port-specific patch) it can be necessary to use the underlying toolkit API directly:

- wxMSW port uses win32 API: see MSDN docs at http://msdn2.microsoft.com/en-us/library/ms649779.aspx
- wxGTK port uses GTK+ and other lower-level libraries; see
  - GTK+ docs at http://library.gnome.org/devel/gtk/unstable/
  - GDK docs at http://library.gnome.org/devel/gdk/unstable/
  - GLib docs at http://library.gnome.org/devel/glib/unstable/
  - GObject docs at http://library.gnome.org/devel/gobject/unstable/
  - Pango docs at http://library.gnome.org/devel/pango/unstable/
- wxMac port uses the Carbon API: see Carbon docs at http://developer.apple.com/carbon
- wxCocoa port uses the Cocoa API: see Cocoa docs at http://developer.apple.com/cocoa

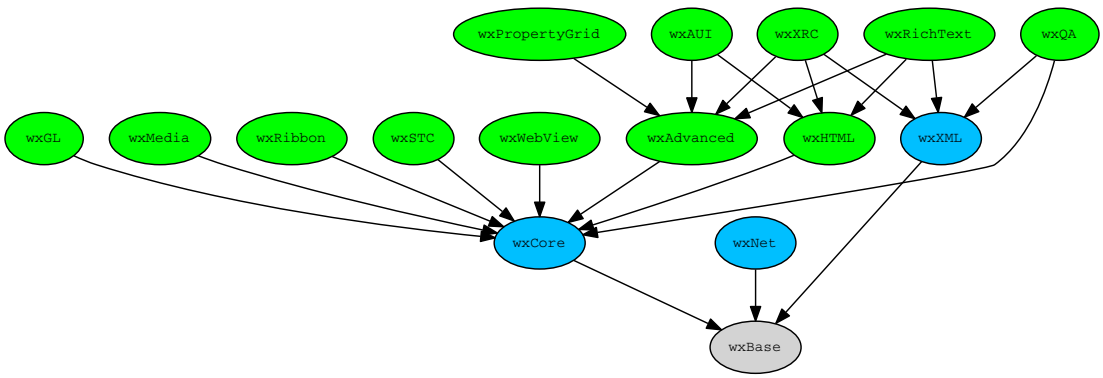
Chapter 7

Library List

wxWidgets can be built either as a single large library (this is called a *monolithic build*) or as several smaller libraries (*multilib build*).

Multilib build is the default.

wxWidgets library is divided into libraries briefly described below. This diagram shows the dependencies between them:



Please note that arrows indicate the "depends from" relation and that all blue libraries depend on the wxBase library (i.e. they are non-GUI libraries), and all green libraries depend on the wxCore library (i.e. they are GUI libraries).

7.1 wxAdvanced

Advanced or rarely used GUI classes:

- wxCalendarCtrl
- wxGrid classes
- wxJoystick
- wxLayoutAlgorithm
- wxSplashScreen
- wxTaskBarIcon

- wxSound
- wxWizard
- wxSashLayoutWindow
- wxSashWindow
- ...others

Requires wxCore and wxBase.

7.2 wxAui

This contains the Advanced User Interface docking library.

Requires wxAdvanced, wxHTML, wxXML, wxCore, wxBase.

7.3 wxBase

Every wxWidgets application must link against this library. It contains mandatory classes that any wxWidgets code depends on (e.g. wxString) and portability classes that abstract differences between platforms. wxBase can be used to develop console mode applications, it does not require any GUI libraries or running X Window System on Unix.

7.4 wxCore

Basic GUI classes such as GDI classes or controls are in this library. All wxWidgets GUI applications must link against this library, only console mode applications don't.

Requires wxBase.

7.5 wxGL

This library contains wxGLCanvas class for integrating OpenGL library with wxWidgets. Unlike all others, this library is **not** part of the monolithic library, it is always built as separate library.

Requires wxCore and wxBase.

7.6 wxHTML

Simple HTML renderer and other wxHTML Overview are contained in this library, as well as wxHtmlHelpController, wxBestHelpController and wxHtmlListBox.

Requires wxCore and wxBase.

7.7 wxMedia

Miscellaneous classes related to multimedia. Currently this library only contains wxMediaCtrl but more classes will be added in the future.

Requires wxCore and wxBase.

7.8 wxNet

Classes for network access:

- wxSocket classes (wxSocketClient, wxSocketServer and related classes)
- wxSocketOutputStream and wxSocketInputStream
- sockets-based IPC classes (wxTCPServer, wxTCPClient and wxTCPConnection)
- wxURL
- wxInternetFSHandler (a wxFileSystem handler)

Requires wxBase.

7.9 wxPropertyGrid

This contains the wxPropertyGrid control.

Requires wxAdvanced, wxCore, wxBase.

7.10 wxQA

This is the library containing extra classes for quality assurance. Currently it only contains wxDebugReport and related classes, but more will be added to it in the future.

Requires wxXML, wxCore, wxBase.

7.11 wxRibbon

This contains the Ribbon User Interface components library.

Requires wxCore, wxBase.

7.12 wxRichText

This contains generic rich text control functionality.

Requires wxAdvanced, wxHTML, wxXML, wxCore, wxBase.

7.13 wxSTC

STC (Styled Text Control) is a wrapper around Scintilla, a syntax-highlighting text editor. See http://www.scintilla.org/ for more info about Scintilla.

Requires wxCore, wxBase.

7.14 wxWebView

The wxWebView library contains the wxWebView control and its associated classes.

Requires wxCore, wxBase.

7.15 wxXML

This library contains simple classes for parsing XML documents.

Requires wxBase.

7.16 wxXRC

This library contains wxXmlResource class that provides access to XML resource files in XRC format.

Requires wxAdvanced, wxHTML, wxXML, wxCore, wxBase.

Chapter 8

Samples Overview

Probably the best way to learn wxWidgets is by reading the source of some 80+ samples provided with it.

Many aspects of wxWidgets programming can be learned from them, but sometimes it is not simple to just choose the right sample to look at. This overview aims at describing what each sample does/demonstrates to make it easier to find the relevant one if a simple grep through all sources didn't help. They also provide some notes about using the samples and what features of wxWidgets are they supposed to test.

There are currently more than 80 different samples as part of wxWidgets: the list in this page is not complete! You should start your tour of wxWidgets with the Minimal Sample which is the wxWidgets version of "Hello, world!". It shows the basic structure of wxWidgets program and is the most commented sample of all - looking at its source code is recommended.

The next most useful sample is Widgets Sample which shows many of wxWidgets controls, such as buttons, text entry zones, list boxes, check boxes, combo boxes etc. It is organized in many different source files, one per each control, which makes it easier to study it, and also allows to change various control styles and call its methods interactively.

Other, more complicated controls, have their own samples. In this category you may find the following samples showing the corresponding controls:

- wxCalendarCtrl: Calendar Sample
- wxListCtrl: List Control Sample
- wxTreeCtrl: wxTreeCtrl Sample
- wxGrid: Grid Sample
- wxDataViewCtrl: wxDataViewCtrl Sample
- wxWebView: wxWebView Sample

Notice that all wxWidgets samples mentioned above can be found in `samples` subdirectory of the library distribution. When a `foobar` sample is mentioned below, its sources can be found in `samples/foobar` directory of your wxWidgets tree. If you installed wxWidgets from a binary package, you might not have this directory. In this case, you may view the samples online at `http://svn.wxwidgets.org/viewvc/wx/wx↩Widgets/trunk/samples/` but you need to download the source distribution in order to be able to build them (highly recommended).

Final advice is to do a search in the entire samples directory if you can't find the sample showing the control you are interested in by name. Most classes contained in wxWidgets occur in at least one of the samples.

Todo Write descriptions for the samples who description started with "This sample demonstrates", they are semi-auto generated.

8.1 Accessibility Sample

This sample shows how you can use the wxAccessible classes in a simple GUI program.

Build Note: You may need to build the wxWidgets library with `wxUSE_ACCESSIBILITY` being set to 1 to be able to make it work, please read comments in `<wx/setup_inc.h>` for more info.

Location: `samples/access` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/access

8.2 Animation Sample

This sample shows how you can use wxAnimationCtrl control and shows concept of a platform-dependent animation encapsulated in wxAnimation.

Location: `samples/animate` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/animate

8.3 Art Provider Sample

This sample shows how you can customize the look of standard wxWidgets dialogs by replacing default bitmaps/icons with your own versions. It also shows how you can use wxArtProvider to get stock bitmaps for use in your application.

Location: `samples/artprov` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/artprov

8.4 Advanced User Interface Sample

This sample demonstrates AUI classes.

Location: `samples/aui` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/aui

8.5 Calendar Sample

This sample shows the calendar control in action. It shows how to configure the control (see the different options in the calendar menu) and also how to process the notifications from it.

Location: `samples/calendar` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/calendar

8.6 Caret Sample

This sample demonstrates wxCaret.

Location: `samples/caret` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/caret

8.7 Collapsible Pane Sample

This sample demonstrates wxCollapsiblePane.

Location: `samples/collpane` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/collpane

8.8 Combo Sample

This sample demonstrates wxComboBox, wxComboCtrl and wxOwnerDrawnComboBox etc.

Location: `samples/combo` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/combo

8.9 Configuration Sample

This sample demonstrates the wxConfig classes in a platform independent way, i.e. it uses text based files to store a given configuration under Unix and uses the Registry under Windows.

See wxConfig Overview for the descriptions of all features of this class.

Location: `samples/config` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/config

8.10 Console Program Sample

This sample demonstrates a console program.

Location: `samples/console` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/console

8.11 Controls Sample

The controls sample is the main test program for most simple controls used in wxWidgets. The sample tests their basic functionality, events, placement, modification in terms of colour and font as well as the possibility to change the controls programmatically, such as adding an item to a list box etc. Apart from that, the sample uses a wxNotebook and tests most features of this special control (using bitmap in the tabs, using wxSizer instances and wxLayoutConstraints within notebook pages, advancing pages programmatically and vetoing a page change by intercepting the wxNotebookEvent.

The various controls tested are listed here:

- wxButton
- wxBitmapButton
- wxCheckBox
- wxChoice
- wxComboBox
- wxGauge
- wxStaticBox
- wxListBox
- wxSpinCtrl
- wxSpinButton

- wxStaticText
- wxStaticBitmap
- wxRadioBox
- wxRadioButton
- wxSlider

Location: `samples/controls` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/controls

8.12 wxDataViewCtrl Sample

This sample demonstrates wxDataViewCtrl.

Location: `samples/dataview` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/dataview

8.13 Clipboard Sample

This sample demonstrates wxClipboard.

Location: `samples/clipboard` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/clipboard

8.14 Debug Reporter Sample

This sample shows how to use wxDebugReport class to generate a debug report in case of a program crash or otherwise. On start up, it proposes to either crash itself (by dereferencing a NULL pointer) or generate debug report without doing it. Next it initializes the debug report with standard information adding a custom file to it (just a timestamp) and allows to view the information gathered using wxDebugReportPreview.

For the report processing part of the sample to work you should make available a Web server accepting form uploads, otherwise wxDebugReportUpload will report an error.

Build Note: You may need to build the wxWidgets library with `wxUSE_DEBUGREPORT` and `wxUSE_ON_FATAL_EXCEPTION` being set to 1 to be able to make it work, please read comments in `<wx/setup_inc.h>` for more info.

Location: `samples/debugrpt` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/debugrpt

8.15 Dialogs Sample

This sample shows how to use the common dialogs available from wxWidgets. These dialogs are described in detail in the Common Dialogs.

In addition to the dialogs accessible from the sample menus, you can also run it with a `-progress=style` command line option to show a wxProgressDialog with the given style (try 0 for the default style) on program startup, before the main window is shown.

Location: `samples/dialogs` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/dialogs

8.16 Dialup Sample

This sample shows the wxDialUpManager class. In the status bar, it displays the information gathered through its interface: in particular, the current connection status (online or offline) and whether the connection is permanent (in which case a string 'LAN' appears in the third status bar field - but note that you may be on a LAN not connected to the Internet, in which case you will not see this) or not.

Using the menu entries, you may also dial or hang up the line if you have a modem attached and (this only makes sense for Windows) list the available connections.

Location: `samples/dialup` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/dialup

8.17 Display Sample

This sample demonstrates wxDisplay.

Location: `samples/display` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/display

8.18 Drag & Drop Sample

This sample shows both clipboard and drag and drop in action. It is quite non trivial and may be safely used as a basis for implementing the clipboard and drag and drop operations in a real-life program.

When you run the sample, its screen is split in several parts. On the top, there are two listboxes which show the standard derivations of wxDropTarget: wxTextDropTarget and wxFileDropTarget.

The middle of the sample window is taken by the log window which shows what is going on (of course, this only works in debug builds) and may be helpful to see the sequence of steps of data transfer.

Finally, the last part is used for dragging text from it to either one of the listboxes (only one will accept it) or another application. The last functionality available from the main frame is to paste a bitmap from the clipboard (or, in the case of the Windows version, also a metafile) - it will be shown in a new frame.

So far, everything we mentioned was implemented with minimal amount of code using standard wxWidgets classes. The more advanced features are demonstrated if you create a shape frame from the main frame menu. A shape is a geometric object which has a position, size and color. It models some application-specific data in this sample. A shape object supports its own private wxDataFormat which means that you may cut and paste it or drag and drop (between one and the same or different shapes) from one sample instance to another (or the same). However, chances are that no other program supports this format and so shapes can also be rendered as bitmaps which allows them to be pasted/dropped in many other applications (and, under Windows, also as metafiles which are supported by most of Windows programs as well - try Write/Wordpad, for example).

Take a look at DnDShapeDataObject class to see how you may use wxDataObject to achieve this.

Location: `samples/dnd` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/dnd

8.19 Document/View Sample

This sample demonstrates Document/View Framework.

Location: `samples/docview` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/docview

See also

MDI Sample

8.20 Drag Image Sample

This sample demonstrates wxDragImage.

Location: `samples/dragimag` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/dragimag`

8.21 Drawing Sample

This sample demonstrates the drawing ability of wxDC.

Location: `samples/drawing` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/drawing`

8.22 Erase Event Sample

This sample demonstrates wxEraseEvent.

Location: `samples/erase` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/erase`

8.23 Event Sample

This sample demonstrates various features of the wxWidgets events. It shows how to dynamic events and connecting/disconnecting the event handlers during run time by using wxEvtHandler::Connect() and wxEvtHandler::Disconnect(), and also how to use wxWindow::PushEventHandler() and wxWindow::PopEventHandler().

Location: `samples/event` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/event`

8.24 Exception Sample

This very simple sample shows how to use C++ exceptions in wxWidgets programs, i.e. where to catch the exception which may be thrown by the program code. It doesn't do anything very exciting by itself, you need to study its code to understand what goes on.

Build Note: You need to build the library with `wxUSE_EXCEPTIONS` being set to `1` and compile your code with C++ exceptions support to be able to build this sample.

Location: `samples/except` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/except`

8.25 External Program Execution Sample

The exec sample demonstrates the wxExecute and wxShell functions. Both of them are used to execute the external programs and the sample shows how to do this synchronously (waiting until the program terminates) or asynchronously (notification will come later).

It also shows how to capture the output of the child process in both synchronous and asynchronous cases and how to kill the processes with wxProcess::Kill() and test for their existence with wxProcess::Exists().

Location: `samples/exec` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/exec`

8.26 Flash Sample

The flash sample demonstrates embedding of Adobe Flash into a wxWidgets program. Currently it only works under Windows as it uses the Flash ActiveX control to achieve this but we hope to be able to extend it to also work under other platforms in the future. The sample also currently requires Microsoft Visual C++ compiler as it uses COM support extensions specific to this compiler.

The sample comes with 2 Flash files (SWF), showing a simple Flash animation which can be controlled using the "Play", "Stop" and "Back"/"Forward" buttons in the sample as well as a Flash form which shows how Flash and wxWidgets program can exchange data: calling "GetText" function without arguments returns the text of the text control defined inside Flash and calling "SetText" with an argument sets the control contents to the given string. Finally clicking on the button generates an event which is caught by the C++ program.

8.27 Font Sample

The font sample demonstrates wxFont, wxFontEnumerator and wxFontMapper classes. It allows you to see the fonts available (to wxWidgets) on the computer and shows all characters of the chosen font as well.

Location: `samples/font` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/font`

8.28 Grid Sample

This sample demonstrates wxGrid.

Location: `samples/grid` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/grid`

8.29 Help Sample

This sample demonstrates wxHelpController.

Location: `samples/help` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/help`

8.30 HTML Sample

Eight HTML samples cover all features of the HTML sub-library.

- **Test** demonstrates how to create wxHtmlWindow and also shows most supported HTML tags.
- **Widget** shows how you can embed ordinary controls or windows within an HTML page. It also nicely explains how to write new tag handlers and extend the library to work with unsupported tags.
- **About** may give you an idea how to write good-looking About boxes.

- **Zip** demonstrates use of virtual file systems in wxHTML. The zip archives handler (ships with wxWidgets) allows you to access HTML pages stored in a compressed archive as if they were ordinary files.

- **Virtual** is yet another virtual file systems demo. This one generates pages at run-time. You may find it useful if you need to display some reports in your application.

- **Printing** explains use of wxHtmlEasyPrinting class which serves as as-simple-as-possible interface for printing HTML documents without much work. In fact, only few function calls are sufficient.

- **Help** and **Helpview** are variations on displaying HTML help (compatible with MS HTML Help Workshop). *Help* shows how to embed wxHtmlHelpController in your application while *Helpview* is a simple tool that only pops up the help window and displays help books given at command line.

Location: `samples/html` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/html`

8.31 HTML List Box Sample

This sample demonstrates wxHtmlListBox.

Location: `samples/htlbox` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/htlbox`

8.32 Image Sample

The image sample demonstrates use of the wxImage class and shows how to download images in a variety of formats, currently PNG, GIF, TIFF, JPEG, BMP, PNM and PCX. The top of the sample shows two rectangles, one of which is drawn directly in the window, the other one is drawn into a wxBitmap, converted to a wxImage, saved as a PNG image and then reloaded from the PNG file again so that conversions between wxImage and wxBitmap as well as loading and saving PNG files are tested.

At the bottom of the main frame there is a test for using a monochrome bitmap by drawing into a wxMemoryDC. The bitmap is then drawn specifying the foreground and background colours with wxDC::SetTextForeground() and wxDC::SetTextBackground() (on the left). The bitmap is then converted to a wxImage and the foreground colour (black) is replaced with red using wxImage::Replace().

This sample also contains the code for testing the image rotation and resizing and using raw bitmap access, see the corresponding menu commands.

Location: `samples/image` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/image`

8.33 Internationalization Sample

The not very clearly named internat sample demonstrates the wxWidgets internationalization (i18n for short from now on) features. To be more precise, it only shows localization support, i.e. support for translating the program messages into another language while true i18n would also involve changing the other aspects of the program's behaviour.

More information about this sample can be found in the `readme.txt` file in its directory. Please also see the Internationalization overview.

Location: `samples/internat` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/internat`

8.34 Connection Sample

This sample demonstrates wxConnection.

Location: `samples/ipc` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/ipc`

8.35 Joystick Sample

This sample demonstrates wxJoystick.

Location: `samples/joytest` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/joytest`

8.36 Key Event Sample

This sample demonstrates wxKeyEvent.

This sample can be used to interactively test the events produced by pressing various keyboard keys. It also shows the interaction between accelerators and the normal keyboard events (which are overridden by any defined accelerators) and finally allows to test that not skipping an event in EVT_KEY_DOWN handler suppresses the subsequent EVT_CHAR event.

Location: `samples/keyboard` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/keyboard`

8.37 Layout Sample

The layout sample demonstrates the two different layout systems offered by wxWidgets. When starting the program, you will see a frame with some controls and some graphics. The controls will change their size whenever you resize the entire frame and the exact behaviour of the size changes is determined using the wxLayoutConstraints class. See also the overview and the wxIndividualLayoutConstraint class for further information.

The menu in this sample offers two more tests, one showing how to use a wxBoxSizer in a simple dialog and the other one showing how to use sizers in connection with a wxNotebook class. See also wxSizer.

Location: `samples/layout` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/layout`

8.38 List Control Sample

This sample shows the wxListCtrl control. Different modes supported by the control (list, icons, small icons, report) may be chosen from the menu.

The sample also provides some timings for adding/deleting/sorting a lot of (several thousands) items into the control.

Location: `samples/listctrl` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/listctrl`

8.39 MDI Sample

This sample demonstrates MDI.

See also

Document/View Sample

Location: `samples/mdi` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/mdi

8.40 Mediaplayer Sample

This sample demonstrates how to use all the features of wxMediaCtrl and play various types of sound, video, and other files.

It replaces the old `dynamic` sample.

Location: `samples/mediaplayer` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/mediaplayer

8.41 Memory Checking Sample

This sample demonstrates memory tracing using wxDebugContext.

Location: `samples/memcheck` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/memcheck

Build Note: You may need to build the wxWidgets library with `wxUSE_MEMORY_TRACING` and `wxUSE_DEBUG_CONTEXT` being set to 1 to be able to make it work, please read comments in `<wx/setup_inc.h>` for more info.

8.42 Menu Sample

This sample demonstrates wxMenu classes.

Location: `samples/menu` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/menu

8.43 MFC Sample

This sample demonstrates how to mix MFC and wxWidgets code. It pops up an initial wxWidgets frame, with a menu item that allows a new MFC window to be created.

For build instructions please read IMPORTANT NOTES in `mfctest.cpp`.

Availability: only available for the wxMSW port.

Location: `samples/mfc` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/mfc

8.44 Minimal Sample

The minimal sample is what most people will know under the term Hello World, i.e. a minimal program that doesn't demonstrate anything apart from what is needed to write a program that will display a "hello" dialog. This is usually a good starting point for learning how to use wxWidgets.

Location: `samples/minimal` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/minimal

8.45 Native Windows Dialog Sample

This sample demonstrates native windows dialog.

Availability: only available for the wxMSW port.

Location: `samples/nativdlg` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/nativdlg`

8.46 Notebook Sample

This samples shows wxBookCtrl family of controls. Although initially it was written to demonstrate wxNotebook only, it can now be also used to see wxListbook, wxChoicebook, wxTreebook and wxToolbook in action. Test each of the controls, their orientation, images and pages using commands through the menu.

Location: `samples/notebook` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/notebook`

8.47 OLE Automation Sample

This sample demonstrates OLE automation using wxAutomationObject.

Availability: only available for the wxMSW port.

Location: `samples/oleauto` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/oleauto`

8.48 OpenGL Sample

This sample demonstrates wxGLCanvas.

- **cube** Draws a cube to demonstrate how to write a basic wxWidgets OpenGL program. Arrow keys rotate the cube. Space bar toggles spinning.
- **isosurf** Draws a surface by reading coordinates from a DAT file.
- **penguin** Draws a rotatable penguin by reading data from a DXF file.

Location: `samples/opengl` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/opengl`

8.49 Owner-drawn Sample

This sample demonstrates owner-drawn wxMenuItem, wxCheckList and wxListBox.

Location: `samples/ownerdrw` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/ownerdrw`

8.50 Popup Transient Window Sample

This sample demonstrates wxPopupTransientWindow.

Location: `samples/popup` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/popup`

8.51 Power Management Sample

This sample demonstrates wxWidgets power management.

See also

wxPowerEvent

Location: `samples/power` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/power

8.52 Printing Sample

This sample demonstrates printing.

See also

Printing Framework Overview, Printing Under Unix (GTK+)

Build Note: You may need to build the wxWidgets library with `wxUSE_PRINTING_ARCHITECTURE` being set to 1 to be able to make it work, please read comments in `<wx/setup_inc.h>` for more info.

Location: `samples/printing` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/printing

8.53 wxPropertyGrid Sample

Sample application has following additional examples of custom properties:

- wxFontDataProperty (edits wxFontData)
- wxPointProperty (edits wxPoint)
- wxSizeProperty (edits wxSize)
- wxAdvImageFileProperty (like wxImageFileProperty, but also has a drop-down for recent image selection)
- wxDirsProperty (edits a wxArrayString consisting of directory strings)
- wxArrayDoubleProperty (edits wxArrayDouble)

This sample demonstrates wxPropertyGrid.

Location: `samples/propgrid` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/propgrid

8.54 Registry Sample

This sample demonstrates wxRegKey.

Availability: only available for the wxMSW port.

Location: `samples/regtest` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/regtest

8.55 Render Sample

This sample shows how to replace the default wxWidgets renderer and also how to write a shared library (DLL) implementing a renderer and load and unload it during the run-time.

Location: `samples/render` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/render

8.56 wxRichTextCtrl Sample

This sample demonstrates wxRichTextCtrl.

Location: `samples/richtext` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/richtext

8.57 Sash Sample

This sample demonstrates wxSashWindow classes.

Location: `samples/sashtest` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/sashtest

8.58 Scroll Window Sample

This sample demonstrates wxScrolledWindow.

This sample demonstrates use of the wxScrolledWindow class including placing subwindows into it and drawing simple graphics. It uses the SetTargetWindow method and thus the effect of scrolling does not show in the scrolled window itself, but in one of its subwindows.

Additionally, this samples demonstrates how to optimize drawing operations in wxWidgets, in particular using the wxWindow::IsExposed() method with the aim to prevent unnecessary drawing in the window and thus reducing or removing flicker on screen.

Location: `samples/scroll` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/scroll

8.59 Shaped Window Sample

This sample demonstrates how to implement a shaped or transparent window, and a window showing/hiding with effect.

See also

wxTopLevelWindow::SetShape(), wxTopLevelWindow::SetTransparent(), wxWindow::ShowWithEffect(), wxWindow::HideWithEffect()

Location: `samples/shaped` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/shaped

8.60 Sockets Sample

The sockets sample demonstrates how to use the communication facilities provided by wxSocket. There are two different applications in this sample: a server, which is implemented using a wxSocketServer object, and a client,

which is implemented as a wxSocketClient.

The server binds to the local address, using TCP port number 3000, sets up an event handler to be notified of incoming connection requests (**wxSOCKET_CONNECTION** events), and sits there, waiting for clients (*listening*, in socket parlance). For each accepted connection, a new wxSocketBase object is created. These socket objects are independent from the server that created them, so they set up their own event handler, and then request to be notified of **wxSOCKET_INPUT** (incoming data) or **wxSOCKET_LOST** (connection closed at the remote end) events. In the sample, the event handler is the same for all connections; to find out which socket the event is addressed to, the GetSocket function is used.

Although it might take some time to get used to the event-oriented system upon which wxSocket is built, the benefits are many. See, for example, that the server application, while being single-threaded (and of course without using fork() or ugly select() loops) can handle an arbitrary number of connections.

The client starts up unconnected, so you can use the Connect... option to specify the address of the server you are going to connect to (the TCP port number is hard-coded as 3000). Once connected, a number of tests are possible. Currently, three tests are implemented. They show how to use the basic IO calls in wxSocketBase, such as wxSocketBase::Read(), wxSocketBase::Write(), wxSocketBase::ReadMsg() and wxSocketBase::WriteMsg(), and how to set up the correct IO flags depending on what you are going to do. See the comments in the code for more information. Note that because both clients and connection objects in the server set up an event handler to catch **wxSOCKET_LOST** events, each one is immediately notified if the other end closes the connection.

There is also a URL test which shows how to use the wxURL class to fetch data from a given URL.

The sockets sample is work in progress. Some things to do:

- More tests for basic socket functionality.
- More tests for protocol classes (wxProtocol and its descendants).
- Tests for the recently added (and still in alpha stage) datagram sockets.
- New samples which actually do something useful (suggestions accepted).

Location: `samples/sockets` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/sockets`

8.61 Sound Sample

The `sound` sample shows how to use wxSound for simple audio output (e.g. notifications).

Location: `samples/sound` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/sound`

8.62 Splash Screen Sample

This sample demonstrates wxSplashScreen.

Location: `samples/splash` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/splash`

8.63 Splitter Window Sample

This sample demonstrates wxSplitterWindow.

Location: `samples/splitter` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/splitter`

8.64 Status Bar Sample

This sample shows how to create and use wxStatusBar. Although most of the samples have a statusbar, they usually only create a default one and only do it once.

Here you can see how to recreate the statusbar (with possibly different number of fields) and how to use it to show icons/bitmaps and/or put arbitrary controls into it.

Location: `samples/statbar` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/statbar`

8.65 wxStyledTextCtrl Sample

This sample demonstrates wxStyledTextCtrl.

Location: `samples/stc` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/stc`

8.66 SVG Sample

This sample demonstrates wxSVGFileDC.

Location: `samples/svg` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/svg`

8.67 Tab Order Sample

This sample allows to test keyboard navigation (mostly done using the `TAB` key, hence the sample name) between different controls. It shows the use of wxWindow::MoveBeforeInTabOrder() and MoveAfterInTabOrder() methods to change the default order of the windows in the navigation chain and of wxWindow::Navigate() for moving focus along this chain.

Location: `samples/taborder` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/taborder`

8.68 Task Bar Icon Sample

This sample demonstrates wxTaskBarIcon.

Location: `samples/taskbar` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/taskbar`

8.69 Text Sample

This sample demonstrates four features: firstly the use and many variants of the wxTextCtrl class (single line, multi line, read only, password, ignoring TAB, ignoring ENTER).

Secondly it shows how to intercept a wxKeyEvent in both the raw form using the `EVT_KEY_UP` and `EVT_KEY_DOWN` macros and the higher level from using the `EVT_CHAR` macro. All characters will be logged in a log window at the bottom of the main window. By pressing some of the function keys, you can test some actions in the text ctrl as well as get statistics on the text ctrls, which is useful for testing if these statistics actually are correct.

Thirdly, on platforms which support it, the sample will offer to copy text to the wxClipboard and to paste text from it. The GTK version will use the so called PRIMARY SELECTION, which is the pseudo clipboard under X and best known from pasting text to the XTerm program.

Last but not least: some of the text controls have tooltips and the sample also shows how tooltips can be centrally disabled and their latency controlled.

Location: `samples/text` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/text

8.70 Thread Sample

This sample demonstrates use of threads in connection with GUI programs.

There are two fundamentally different ways to use threads in GUI programs and either way has to take care of the fact that the GUI library itself usually is not multi-threading safe, i.e. that it might crash if two threads try to access the GUI class simultaneously.

One way to prevent that is have a normal GUI program in the main thread and some worker threads which work in the background. In order to make communication between the main thread and the worker threads possible, wxWidgets offers the wxQueueEvent function and this sample demonstrates its usage.

The other way is to use a wxMutexGuiEnter and wxMutexGuiLeave functions, but this is not currently shown in the sample.

See also Multithreading Overview and wxThread.

Location: `samples/thread` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/thread

8.71 Tool Bar Sample

The toolbar sample shows the wxToolBar class in action.

The following things are demonstrated:

- Creating the toolbar using wxToolBar::AddTool() and wxToolBar::AddControl(): see MyApp::InitToolbar() in the sample.
- Using `EVT_UPDATE_UI` handler for automatically enabling/disabling toolbar buttons without having to explicitly call EnableTool. This is done in MyFrame::OnUpdateCopyAndCut().
- Using wxToolBar::DeleteTool() and wxToolBar::InsertTool() to dynamically update the toolbar.

Some buttons in the main toolbar are check buttons, i.e. they stay checked when pressed. On the platforms which support it, the sample also adds a combobox to the toolbar showing how you can use arbitrary controls and not only buttons in it.

If you toggle another toolbar in the sample (using `Ctrl-A`) you will also see the radio toolbar buttons in action: the first three buttons form a radio group, i.e. checking any of them automatically unchecks the previously checked one.

Location: `samples/toolbar` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/toolbar

8.72 wxTreeCtrl Sample

This sample demonstrates using the wxTreeCtrl class. Here you may see how to process various notification messages sent by this control and also when they occur (by looking at the messages in the text control in the bottom part of the frame).

Adding, inserting and deleting items and branches from the tree as well as sorting (in default alphabetical order as well as in custom one) is demonstrated here as well - try the corresponding menu entries.

Location: `samples/treectrl` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/treectrl

8.73 Types Sample

This sample demonstrates wxWidgets types.

Todo This sample isn't very didactive; it's more than a set of tests rather than a sample and thus should be rewritten with CppUnit and moved under "tests"

Location: `samples/typetest` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/typetest

8.74 wxUIActionSimulator Sample

This sample demonstrates wxUIActionSimulator.

This sample shows some features of wxUIActionSimulator class. When a simulation is run using its menu items, you can see that the button is pressed programmatically and the characters generated by the program appear in the text control.

Location: `samples/uiaction` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/uiaction

8.75 Validator Sample

This sample demonstrates wxValidator.

Location: `samples/validate` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/validate

8.76 VScrolled Window Sample

This sample demonstrates wxVScrolledWindow.

Location: `samples/vscroll` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/vscroll

8.77 wxWebView Sample

The wxWebView sample demonstarates the various capabilities of the wxWebView control. It is set up as a simple single window web browser, but with support for many of the more complex wxWebView features, including browsing through archives.

Location: `samples/webview` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/webview

8.78 Widgets Sample

The widgets sample is the main presentation program for most simple and advanced native controls and complex generic widgets provided by wxWidgets. The sample tests their basic functionality, events, placement, modification in terms of colour and font as well as the possibility to change the controls programmatically, such as adding an item to a list box etc. All widgets are categorized for easy browsing.

Location: `samples/widgets` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/widgets

8.79 Wizard Sample

This sample shows the so-called wizard dialog (implemented using wxWizard and related classes). It shows almost all features supported:

- Using bitmaps with the wizard and changing them depending on the page shown (notice that wxValidationPage in the sample has a different image from the other ones)
- Using TransferDataFromWindow to verify that the data entered is correct before passing to the next page (done in wxValidationPage which forces the user to check a checkbox before continuing).
- Using more elaborated techniques to allow returning to the previous page, but not continuing to the next one or vice versa (in wxRadioboxPage)
- This (wxRadioboxPage) page also shows how the page may process the *Cancel* button itself instead of relying on the wizard parent to do it.
- Normally, the order of the pages in the wizard is known at compile-time, but sometimes it depends on the user choices: wxCheckboxPage shows how to dynamically decide which page to display next (see also wxWizardPage)

Location: `samples/wizard` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/wizard

8.80 wxWrapSizer Sample

This sample demonstrates wxWrapSizer.

Location: `samples/wrapsizer` subdirectory of your wxWidgets installation or http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/wrapsizer

8.81 XRC Sample

This sample shows how to use the various features of the XML Based Resource System (XRC) to create the gui of your program. It starts by loading and showing a frame and other resources. From its menu or toolbar you can then run the following dialogs:

- A non-derived wxDialog
- A derived dialog
- A dialog containing a large number of controls
- An uncentred dialog
- A dialog demonstrating the use of object references and ID ranges

- A dialog that contains a custom class
- A dialog with platform-specific features
- A dialog demonstrating wxArtProvider
- A dialog saying "VARIABLE EXPANSION ISN'T IMPLEMENTED CURRENTLY" :/

Location: `samples/xrc` subdirectory of your wxWidgets installation or `http://svn.wxwidgets.org/viewvc/wx/wxWidgets/trunk/samples/xrc`

Chapter 9

Screenshots of Different Controls

This page contains the screenshots of various controls under the three major platforms: wxMSW in the first column, wxGTK in the second one and wxOSX in the third one.

9.1 Standard Controls

Some common controls:

wxButton wxBitmapButton wxCheckBox wxChoice wxCheckListBox wxComboBox wxGauge wxListBox wxRadio↩ Box wxRadioButton wxScrollBar wxSlider wxSpinButton wxSpinCtrl wxSpinCtrlDouble

9.2 Picker Controls

These controls provide the user with the possibility to choose something (file or directory, font or colour, ...) directly from the window containing them:

wxColourPickerCtrl wxDatePickerCtrl wxFilePickerCtrl wxDirPickerCtrl wxFontPickerCtrl

9.3 Advanced Controls

These controls are considered to be less common and are defined in adv library:

wxAnimationCtrl wxBannerWindow wxBitmapComboBox wxCalendarCtrl wxComboCtrl wxCommandLinkButton wxHyperlinkCtrl wxOwnerDrawnComboBox

9.4 Book Controls

Book controls contain several pages (also called tabs in wxNotebook case) and allow the user to switch between them:

wxChoicebook wxListbook wxNotebook

9.5 Tree and List Controls

Several controls can be used to display items organized in a tree or (multi column) list:

wxDataViewCtrl wxDataViewTreeCtrl wxListCtrl wxPropertyGrid wxSimpleHtmlListBox

9.6 Miscellaneous Other Controls

wxCollapsiblePane wxDirCtrl wxFileCtrl wxRichTextCtrl wxRichToolTip

Chapter 10

Programming Guides

The guides here cover all high level details of a full range of development topics related to building applications with wxWidgets.

10.1 Starting with wxWidgets

- Notes on Using this Reference Manual
- A Quick Guide to Writing Applications
- Hello World Example
- wxPython Overview

10.2 Important wxWidgets Topics

- wxApp Overview
- Unicode Support in wxWidgets
- Internationalization
- Events and Event Handling
- Window Sizing Overview
- Window IDs
- Logging Overview

10.3 Non-GUI Classes

- wxString Overview
- Buffer Classes
- Date and Time
- Container Classes
- File Classes and Functions
- Stream Classes Overview

- Multithreading Overview
- wxConfig Overview
- Persistent Objects Overview
- wxFileSystem Overview
- Regular Expressions
- Archive Formats
- Interprocess Communication

10.4 Drawing Related Classes

- Device Contexts
- Bitmaps and Icons
- wxFont Overview
- Font Encodings
- Printing Framework Overview
- Printing Under Unix (GTK+)

10.5 GUI Classes

- Sizers Overview
- XML Based Resource System (XRC)
- XRC File Format
- Scrolled Windows
- wxDialog Overview
- wxValidator Overview
- wxDataObject Overview
- Drag and Drop Overview

10.6 Individual Controls

- wxHTML Overview
- wxRichTextCtrl Overview
- wxAUI Overview
- wxPropertyGrid Overview
- Common Dialogs
- Toolbar Overview
- wxGrid Overview

- wxTreeCtrl Overview
- wxListCtrl Overview
- wxSplitterWindow Overview
- wxBookCtrl Overview
- wxTipProvider Overview
- Document/View Framework

10.7 Other wxWidgets Programming Overviews

- Backwards Compatibility
- C++ Exceptions
- Runtime Type Information (RTTI)
- Caveats When Not Using C++ RTTI
- Reference Counting
- wxMBConv Overview
- Writing Non-English Applications
- Debugging
- Window Styles
- Window Deletion
- Environment Variables
- Creating a Custom Widget

10.8 Notes on Using this Reference Manual

In the descriptions of the wxWidgets classes and their member functions, note that descriptions of inherited member functions are not duplicated in derived classes unless their behaviour is different.

So in using a class such as wxScrolledWindow, be aware that wxWindow functions may be relevant.

Where not explicitly stated, size and position arguments may usually be given a value of wxDefaultSize and wx↩DefaultPosition, in which case wxWidgets will choose suitable values.

10.9 A Quick Guide to Writing Applications

To set a wxWidgets application going, you will need to derive a wxApp class and override wxApp::OnInit.

An application must have a top-level wxFrame or wxDialog window. Each frame may contain one or more instances of classes such as wxPanel, wxSplitterWindow or other windows and controls.

A frame can have a wxMenuBar, a wxToolBar, a wxStatusBar, and a wxIcon for when the frame is iconized.

A wxPanel is used to place controls (classes derived from wxControl) which are used for user interaction. Examples of controls are wxButton, wxCheckBox, wxChoice, wxListBox, wxRadioBox, and wxSlider.

Instances of wxDialog can also be used for controls and they have the advantage of not requiring a separate frame.

Instead of creating a dialog box and populating it with items, it is possible to choose one of the convenient common dialog classes, such as wxMessageDialog and wxFileDialog.

You never draw directly onto a window - you use a *device context* (DC). wxDC is the base for wxClientDC, wxPaintDC, wxMemoryDC, wxPostScriptDC, wxMemoryDC, wxMetafileDC and wxPrinterDC. If your drawing functions have wxDC as a parameter, you can pass any of these DCs to the function, and thus use the same code to draw to several different devices. You can draw using the member functions of wxDC, such as wxDC::DrawLine and wxDC::DrawText. Control colour on a window (wxColour) with brushes (wxBrush) and pens (wxPen).

To intercept events, you add a DECLARE_EVENT_TABLE macro to the window class declaration, and put a BEGIN_EVENT_TABLE ... END_EVENT_TABLE block in the implementation file. Between these macros, you add event macros which map the event (such as a mouse click) to a member function. These might override predefined event handlers such as for wxKeyEvent and wxMouseEvent.

Most modern applications will have an on-line, hypertext help system; for this, you need wxHelp and the wxHelpController class to control wxHelp.

GUI applications aren't all graphical wizardry. List and hash table needs are catered for by wxList and wxHashMap. You will undoubtedly need some platform-independent Files and Directories, and you may find it handy to maintain and search a list of paths using wxPathList. There's many Miscellaneous of operating system methods and other functions.

See also

Class List by Category

10.10 Hello World Example

This page shows a very simple wxWidgets program that can be used as a skeleton for your own code.

While it does nothing very useful, it introduces a couple of important concepts and explains how to write a working wxWidgets application.

First, you have to include wxWidgets' header files, of course. This can be done on a file by file basis (such as `wx/window.h`) or using one global include (`wx/wx.h`) which includes most of the commonly needed headers (although not all of them as there are simply too many wxWidgets headers to pull in all of them). For the platforms with support for precompiled headers, as indicated by `WX_PRECOMP`, this global header is already included by `wx/wxprec.h` so we only include it for the other ones:

```
// wxWidgets "Hello world" Program

// For compilers that support precompilation, includes "wx/wx.h".
#include <wx/wxprec.h>

#ifndef WX_PRECOMP
    #include <wx/wx.h>
#endif
```

Practically every app should define a new class derived from wxApp. By overriding wxApp's OnInit() virtual method the program can be initialized, e.g. by creating a new main window.

```
class MyApp: public wxApp
{
public:
    virtual bool OnInit();
};
```

The main window is created by deriving a class from wxFrame and giving it a menu and a status bar in its constructor. Also, any class that wishes to respond to any "event" (such as mouse clicks or messages from the menu or a button) must declare an event table using the macro below.

Finally, the way to react to such events must be done in "handlers". In our sample, we react to three menu items, one for our custom menu command and two for the standard "Exit" and "About" commands (any program should normally implement the latter two). Notice that these handlers don't need to be neither virtual nor public.

```
class MyFrame: public wxFrame
{
public:
    MyFrame(const wxString& title, const wxPoint& pos, const
      wxSize& size);

private:
    void OnHello(wxCommandEvent& event);
    void OnExit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

    wxDECLARE_EVENT_TABLE();
};
```

In order to be able to react to a menu command, it must be given a unique identifier which can be defined as a const variable or an enum element. The latter is often used because typically many such constants will be needed:

```
enum
{
    ID_Hello = 1
};
```

Notice that you don't need to define identifiers for the "About" and "Exit". We then proceed to actually implement an event table in which the events are routed to their respective handler functions in the class MyFrame.

There are predefined macros for routing all common events, ranging from the selection of a list box entry to a resize event when a user resizes a window on the screen. If `wxID_ANY` is given as the ID, the given handler will be invoked for any event of the specified type, so that you could add just one entry in the event table for all menu commands or all button commands etc.

The origin of the event can still be distinguished in the event handler as the (only) parameter in an event handler is a reference to a wxEvent object, which holds various information about the event (such as the ID of and a pointer to the class, which emitted the event).

```
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(ID_Hello,   MyFrame::OnHello)
    EVT_MENU(wxID_EXIT,  MyFrame::OnExit)
    EVT_MENU(wxID_ABOUT, MyFrame::OnAbout)
wxEND_EVENT_TABLE()
```

As in all programs there must be a "main" function. Under wxWidgets main is implemented using this macro, which creates an application instance and starts the program.

```
wxIMPLEMENT_APP(MyApp)
```

As mentioned above, wxApp::OnInit() is called upon startup and should be used to initialize the program, maybe showing a "splash screen" and creating the main window (or several). The frame should get a title bar text ("Hello World") and a position and start-up size. One frame can also be declared to be the top window. Returning true indicates a successful initialization.

```
bool MyApp::OnInit()
{
    MyFrame *frame = new MyFrame( "Hello World", wxPoint(50, 50), wxSize(450, 340) );
    frame->Show( true );
    return true;
}
```

In the constructor of the main window (or later on) we create a menu with our menu items as well as a status bar to be shown at the bottom of the main window. Both have to be associated with the frame with respective calls.

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const
      wxSize& size)
        : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    wxMenu *menuFile = new wxMenu;
    menuFile->Append(ID_Hello, "&Hello...\tCtrl-H",
                     "Help string shown in status bar for this menu item");
    menuFile->AppendSeparator();
    menuFile->Append(wxID_EXIT);
```

```
    wxMenu *menuHelp = new wxMenu;
    menuHelp->Append(wxID_ABOUT);

    wxMenuBar *menuBar = new wxMenuBar;
    menuBar->Append( menuFile, "&File" );
    menuBar->Append( menuHelp, "&Help" );

    SetMenuBar( menuBar );

    CreateStatusBar();
    SetStatusText( "Welcome to wxWidgets!" );
}
```

Notice that we don't need to specify the labels for the standard menu items `wxID_ABOUT` and `wxID_EXIT`, they will be given standard (even correctly translated) labels and also standard accelerators correct for the current platform making your program behaviour more native. For this reason you should prefer reusing the standard ids (see Stock Items) if possible.

Here are the standard event handlers implementations. MyFrame::OnExit() closes the main window by calling Close(). The parameter true indicates that other windows have no veto power such as after asking "Do you really want to close?". If there is no other main window left, the application will quit.

```
void MyFrame::OnExit(wxCommandEvent& event)
{
    Close( true );
}
```

MyFrame::OnAbout() will display a small window with some text in it. In this case a typical "About" window with information about the program.

```
void MyFrame::OnAbout(wxCommandEvent& event)
{
    wxMessageBox( "This is a wxWidgets' Hello world sample",
                  "About Hello World", wxOK | wxICON_INFORMATION );
}
```

The implementation of custom menu command handler may perform whatever task your program needs to do, in this case we will simply show a message from it as befits a hello world example:

```
void MyFrame::OnHello(wxCommandEvent& event)
{
    wxLogMessage("Hello world from wxWidgets!");
}
```

Here is the entire program that can be copied and pasted:

```
// wxWidgets "Hello world" Program

// For compilers that support precompilation, includes "wx/wx.h".
#include <wx/wxprec.h>

#ifndef WX_PRECOMP
    #include <wx/wx.h>
#endif

class MyApp: public wxApp
{
public:
    virtual bool OnInit();
};

class MyFrame: public wxFrame
{
public:
    MyFrame(const wxString& title, const wxPoint& pos, const
      wxSize& size);

private:
    void OnHello(wxCommandEvent& event);
    void OnExit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

    wxDECLARE_EVENT_TABLE();
};
```

```
enum
{
    ID_Hello = 1
};

wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(ID_Hello,   MyFrame::OnHello)
    EVT_MENU(wxID_EXIT,  MyFrame::OnExit)
    EVT_MENU(wxID_ABOUT, MyFrame::OnAbout)
wxEND_EVENT_TABLE()

wxIMPLEMENT_APP(MyApp);

bool MyApp::OnInit()
{
    MyFrame *frame = new MyFrame( "Hello World", wxPoint(50, 50), wxSize(450, 340) );
    frame->Show( true );
    return true;
}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const
      wxSize& size)
        : wxFrame(NULL, wxID_ANY, title, pos, size)
{
    wxMenu *menuFile = new wxMenu;
    menuFile->Append(ID_Hello, "&Hello...\tCtrl-H",
                     "Help string shown in status bar for this menu item");
    menuFile->AppendSeparator();
    menuFile->Append(wxID_EXIT);

    wxMenu *menuHelp = new wxMenu;
    menuHelp->Append(wxID_ABOUT);

    wxMenuBar *menuBar = new wxMenuBar;
    menuBar->Append( menuFile, "&File" );
    menuBar->Append( menuHelp, "&Help" );

    SetMenuBar( menuBar );

    CreateStatusBar();
    SetStatusText( "Welcome to wxWidgets!" );
}

void MyFrame::OnExit(wxCommandEvent& event)
{
    Close( true );
}

void MyFrame::OnAbout(wxCommandEvent& event)
{
    wxMessageBox( "This is a wxWidgets' Hello world sample",
                  "About Hello World", wxOK | wxICON_INFORMATION );
}

void MyFrame::OnHello(wxCommandEvent& event)
{
    wxLogMessage("Hello world from wxWidgets!");
}
```

10.11 wxPython Overview

This topic was written by Robin Dunn, author of the wxPython wrapper.

10.11.1 What is wxPython?

wxPython is a blending of the wxWidgets GUI classes and the Python programming language.

Python

So what is Python? Go to http://www.python.org to learn more, but in a nutshell Python is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, and new built-in

modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

Python is copyrighted but freely usable and distributable, even for commercial use.

wxPython

wxPython is a Python package that can be imported at runtime that includes a collection of Python modules and an extension module (native code). It provides a series of Python classes that mirror (or shadow) many of the wxWidgets GUI classes. This extension module attempts to mirror the class hierarchy of wxWidgets as closely as possible. This means that there is a wxFrame class in wxPython that looks, smells, tastes and acts almost the same as the wxFrame class in the C++ version.

wxPython is very versatile. It can be used to create standalone GUI applications, or in situations where Python is embedded in a C++ application as an internal scripting or macro language.

Currently wxPython is available for Win32 platforms and the GTK toolkit (wxGTK) on most Unix/X-windows platforms. See the wxPython website http://wxPython.org/ for details about getting wxPython working for you.

10.11.2 Why Use wxPython?

So why would you want to use wxPython over just C++ and wxWidgets? Personally I prefer using Python for everything. I only use C++ when I absolutely have to eke more performance out of an algorithm, and even then I usually code it as an extension module and leave the majority of the program in Python.

Another good thing to use wxPython for is quick prototyping of your wxWidgets apps. With C++ you have to continuously go though the edit-compile-link-run cycle, which can be quite time consuming. With Python it is only an edit-run cycle. You can easily build an application in a few hours with Python that would normally take a few days or longer with C++. Converting a wxPython app to a C++/wxWidgets app should be a straight forward task.

10.11.3 Other Python GUIs

There are other GUI solutions out there for Python.

Tkinter

Tkinter is the de facto standard GUI for Python. It is available on nearly every platform that Python and Tcl/TK are. Why Tcl/Tk? Well because Tkinter is just a wrapper around Tcl's GUI toolkit, Tk. This has it's upsides and it's downsides...

The upside is that Tk is a pretty versatile toolkit. It can be made to do a lot of things in a lot of different environments. It is fairly easy to create new widgets and use them interchangeably in your programs.

The downside is Tcl. When using Tkinter you actually have two separate language interpreters running, the Python interpreter and the Tcl interpreter for the GUI. Since the guts of Tcl is mostly about string processing, it is fairly slow as well. (Not too bad on a fast Pentium II, but you really notice the difference on slower machines.)

It wasn't until the latest version of Tcl/Tk that native Look and Feel was possible on non-Motif platforms. This is because Tk usually implements its own widgets (controls) even when there are native controls available.

Tkinter is a pretty low-level toolkit. You have to do a lot of work (verbose program code) to do things that would be much simpler with a higher level of abstraction.

PythonWin

PythonWin is an add-on package for Python for the Win32 platform. It includes wrappers for MFC as well as much of the Win32 API. Because of its foundation, it is very familiar for programmers who have experience with MFC and

the Win32 API. It is obviously not compatible with other platforms and toolkits. PythonWin is organized as separate packages and modules so you can use the pieces you need without having to use the GUI portions.

Others

There are quite a few other GUI modules available for Python, some in active use, some that haven't been updated for ages. Most are simple wrappers around some C or C++ toolkit or another, and most are not cross-platform compatible. See this link for a listing of a few of them.

10.11.4 Using wxPython

I'm not going to try and teach the Python language here. You can do that at the Python Tutorial. I'm also going to assume that you know a bit about wxWidgets already, enough to notice the similarities in the classes used.

Take a look at the following wxPython program. You can find a similar program in the wxPython/demo directory, named DialogUnits.py. If your Python and wxPython are properly installed, you should be able to run it by issuing this command:

```
python DialogUnits.py
```

```
01: ## import all of the wxPython GUI package
02: from wxPython.wx import *
03:
04: ## Create a new frame class, derived from the wxPython Frame.
05: class MyFrame(wxFrame):
06:
07:     def __init__(self, parent, id, title):
08:         # First, call the base class' __init__ method to create the frame
09:         wxFrame.__init__(self, parent, id, title,
10:                          wxPoint(100, 100), wxSize(160, 100))
11:
12:         # Associate some events with methods of this class
13:         EVT_SIZE(self, self.OnSize)
14:         EVT_MOVE(self, self.OnMove)
15:
16:         # Add a panel and some controls to display the size and position
17:         panel = wxPanel(self, -1)
18:         wxStaticText(panel, -1, "Size:",
19:                      wxDLG_PNT(panel, wxPoint(4, 4)),  wxDefaultSize)
20:         wxStaticText(panel, -1, "Pos:",
21:                      wxDLG_PNT(panel, wxPoint(4, 14)), wxDefaultSize)
22:         self.sizeCtrl = wxTextCtrl(panel, -1, "",
23:                                    wxDLG_PNT(panel, wxPoint(24, 4)),
24:                                    wxDLG_SZE(panel, wxSize(36, -1)),
25:                                    wxTE_READONLY)
26:         self.posCtrl = wxTextCtrl(panel, -1, "",
27:                                   wxDLG_PNT(panel, wxPoint(24, 14)),
28:                                   wxDLG_SZE(panel, wxSize(36, -1)),
29:                                   wxTE_READONLY)
30:
31:
32:     # This method is called automatically when the CLOSE event is
33:     # sent to this window
34:     def OnCloseWindow(self, event):
35:         # tell the window to kill itself
36:         self.Destroy()
37:
38:     # This method is called by the system when the window is resized,
39:     # because of the association above.
40:     def OnSize(self, event):
41:         size = event.GetSize()
42:         self.sizeCtrl.SetValue("%s, %s" % (size.width, size.height))
43:
44:         # tell the event system to continue looking for an event handler,
45:         # so the default handler will get called.
46:         event.Skip()
47:
48:     # This method is called by the system when the window is moved,
49:     # because of the association above.
50:     def OnMove(self, event):
51:         pos = event.GetPosition()
52:         self.posCtrl.SetValue("%s, %s" % (pos.x, pos.y))
53:
54:
55: # Every wxWidgets application must have a class derived from wxApp
56: class MyApp(wxApp):
```

```
57:
58:     # wxWidgets calls this method to initialize the application
59:     def OnInit(self):
60:
61:         # Create an instance of our customized Frame class
62:         frame = MyFrame(NULL, -1, "This is a test")
63:         frame.Show(true)
64:
67:
68:         # Return a success flag
69:         return true
70:
71:
72: app = MyApp(0)     # Create an instance of the application class
73: app.MainLoop()     # Tell it to start processing events
74:
```

Things to Notice

At line 2 the wxPython classes, constants, and etc. are imported into the current module's namespace. If you prefer to reduce namespace pollution you can use "`from wxPython import wx`" and then access all the wxPython identifiers through the wx module, for example, "`wx.wxFrame`".

At line 13 the frame's sizing and moving events are connected to methods of the class. These helper functions are intended to be like the event table macros that wxWidgets employs. But since static event tables are impossible with wxPython, we use helpers that are named the same to dynamically build the table. The only real difference is that the first argument to the event helpers is always the window that the event table entry should be added to.

Notice the use of `wxDLG_PNT` and `wxDLG_SZE` in lines 19-29 to convert from dialog units to pixels. These helpers are unique to wxPython since Python can't do method overloading like C++.

There is an `OnCloseWindow` method at line 34 but no call to `EVT_CLOSE` to attach the event to the method. Does it really get called? The answer is, yes it does. This is because many of the standard events are attached to windows that have the associated standard method names. I have tried to follow the lead of the C++ classes in this area to determine what is standard but since that changes from time to time I can make no guarantees, nor will it be fully documented. When in doubt, use an `EVT_***` function.

At lines 17 to 21 notice that there are no saved references to the panel or the static text items that are created. Those of you who know Python might be wondering what happens when Python deletes these objects when they go out of scope. Do they disappear from the GUI? They don't. Remember that in wxPython the Python objects are just shadows of the corresponding C++ objects. Once the C++ windows and controls are attached to their parents, the parents manage them and delete them when necessary. For this reason, most wxPython objects do not need to have a **`del`** method that explicitly causes the C++ object to be deleted. If you ever have the need to forcibly delete a window, use the Destroy() method as shown on line 36.

Just like wxWidgets in C++, wxPython apps need to create a class derived from `wxApp` (line 56) that implements a method named `OnInit`, (line 59.) This method should create the application's main window (line 62) and show it.

And finally, at line 72 an instance of the application class is created. At this point wxPython finishes initializing itself, and calls the `OnInit` method to get things started. (The zero parameter here is a flag for functionality that isn't quite implemented yet. Just ignore it for now.) The call to `MainLoop` at line 73 starts the event loop which continues until the application terminates or all the top level windows are closed.

10.11.5 Classes Implemented in wxPython

The following classes are supported in wxPython. Most provide nearly full implementations of the public interfaces specified in the C++ documentation, others are less so. They will all be brought as close as possible to the C++ spec over time.

- wxAcceleratorEntry
- wxAcceleratorTable
- wxActivateEvent
- wxBitmap

- wxBitmapButton
- wxBitmapDataObject
- wxBMPHandler
- wxBoxSizer
- wxBrush
- wxBusyInfo
- wxBusyCursor
- wxButton
- wxCalculateLayoutEvent
- wxCalendarCtrl
- wxCaret
- wxCheckBox
- wxCheckListBox
- wxChoice
- wxClientDC
- wxClipboard
- wxCloseEvent
- wxColourData
- wxColourDialog
- wxColour
- wxComboBox
- wxCommandEvent
- wxConfigBase
- wxControl
- wxCursor
- wxCustomDataObject
- wxDataFormat
- wxDataObject
- wxDataObjectComposite
- wxDataObjectSimple
- wxDateTime
- wxDateSpan
- wxDC
- wxDialog
- wxDirDialog
- wxDragImage

- wxDropFilesEvent
- wxDropSource
- wxDropTarget
- wxEraseEvent
- wxEvent
- wxEvtHandler
- wxFileConfig
- wxFileDataObject
- wxFileDialog
- wxFileDropTarget
- wxFileSystem
- wxFileSystemHandler
- wxFocusEvent
- wxFontData
- wxFontDialog
- wxFont
- wxFrame
- wxFSFile
- wxGauge
- wxGIFHandler
- wxGLCanvas
- wxHtmlCell
- wxHtmlContainerCell
- wxHtmlDCRenderer
- wxHtmlEasyPrinting
- wxHtmlParser
- wxHtmlTagHandler
- wxHtmlTag
- wxHtmlWinParser
- wxHtmlPrintout
- wxHtmlWinTagHandler
- wxHtmlWindow
- wxIconizeEvent
- wxIcon
- wxIdleEvent
- wxImage

- wxImageHandler
- wxImageList
- wxIndividualLayoutConstraint
- wxInitDialogEvent
- wxInputStream
- wxInternetFSHandler
- wxJoystickEvent
- wxJPEGHandler
- wxKeyEvent
- wxLayoutAlgorithm
- wxLayoutConstraints
- wxListBox
- wxListCtrl
- wxListEvent
- wxListItem
- wxMask
- wxMaximizeEvent
- wxMDIChildFrame
- wxMDIClientWindow
- wxMDIParentFrame
- wxMemoryDC
- wxMemoryFSHandler
- wxMenuBar
- wxMenuEvent
- wxMenuItem
- wxMenu
- wxMessageDialog
- wxMetafileDC
- wxMiniFrame
- wxMouseEvent
- wxMoveEvent
- wxNotebookEvent
- wxNotebook
- wxPageSetupDialogData
- wxPageSetupDialog
- wxPaintDC

- wxPaintEvent
- wxPalette
- wxPanel
- wxPen
- wxPNGHandler
- wxPoint
- wxPostScriptDC
- wxPreviewFrame
- wxPrintData
- wxPrintDialogData
- wxPrintDialog
- wxPrinter
- wxPrintPreview
- wxPrinterDC
- wxPrintout
- wxProcess
- wxQueryLayoutInfoEvent
- wxRadioBox
- wxRadioButton
- wxRealPoint
- wxRect
- wxRegionIterator
- wxRegion
- wxSashEvent
- wxSashLayoutWindow
- wxSashWindow
- wxScreenDC
- wxScrollBar
- wxScrollEvent
- wxScrolledWindow
- wxScrollWinEvent
- wxShowEvent
- wxSingleChoiceDialog
- wxSizeEvent
- wxSize
- wxSizer

- wxSizerItem
- wxSlider
- wxSpinButton
- wxSpinEvent
- wxSplitterWindow
- wxStaticBitmap
- wxStaticBox
- wxStaticBoxSizer
- wxStaticLine
- wxStaticText
- wxStatusBar
- wxSysColourChangedEvent
- wxTaskBarIcon
- wxTextCtrl
- wxTextDataObject
- wxTextDropTarget
- wxTextEntryDialog
- wxTimer
- wxTimerEvent
- wxTimeSpan
- wxTipProvider
- wxToolBarTool
- wxToolBar
- wxToolTip
- wxTreeCtrl
- wxTreeEvent
- wxTreeItemData
- wxTreeItemId
- wxUpdateUIEvent
- wxValidator
- wxWindowDC
- wxWindow
- wxZipFSHandler

10.11.6 Where to Go for Help

Since wxPython is a blending of multiple technologies, help comes from multiple sources. See http://wxpython.org/ for details on various sources of help, but probably the best source is the wxPython-users mail list. You can view the archive or subscribe by going to http://wxpython.org/maillist.php

Or you can send mail directly to the list using this address: wxpython-users@lists.wxwidgets.org

10.12 wxApp Overview

A wxWidgets application does not have a *main* procedure; the equivalent is the wxApp::OnInit member defined for a class derived from wxApp.

OnInit will usually create a top window as a bare minimum. Unlike in earlier versions of wxWidgets, OnInit does not return a frame. Instead it returns a boolean value which indicates whether processing should continue (true) or not (false).

Note that the program's command line arguments, represented by *argc* and *argv*, are available from within wxApp member functions.

An application closes by destroying all windows. Because all frames must be destroyed for the application to exit, it is advisable to use parent frames wherever possible when creating new frames, so that deleting the top level frame will automatically delete child frames. The alternative is to explicitly delete child frames in the top-level frame's wxCloseEvent handler.

In emergencies the wxExit function can be called to kill the application however normally the application shuts down automatically, see Application Shutdown.

An example of defining an application follows:

```
class DerivedApp : public wxApp
{
public:
    virtual bool OnInit();
};

IMPLEMENT_APP(DerivedApp)

bool DerivedApp::OnInit()
{
    wxFrame *the_frame = new wxFrame(NULL, ID_MYFRAME, argv[0]);
    ...
    the_frame->Show(true);

    return true;
}
```

Note the use of IMPLEMENT_APP(appClass), which allows wxWidgets to dynamically create an instance of the application object at the appropriate point in wxWidgets initialization. Previous versions of wxWidgets used to rely on the creation of a global application object, but this is no longer recommended, because required global initialization may not have been performed at application object construction time.

You can also use DECLARE_APP(appClass) in a header file to declare the wxGetApp function which returns a reference to the application object. Otherwise you can only use the global `wxTheApp` pointer which is of type `wxApp*`.

10.12.1 Application Shutdown

The application normally shuts down when the last of its top level windows is closed. This is normally the expected behaviour and means that it is enough to call wxWindow::Close() in response to the "Exit" menu command if your program has a single top level window. If this behaviour is not desirable wxApp::SetExitOnFrameDelete can be called to change it.

Note that such logic doesn't apply for the windows shown before the program enters the main loop: in other words, you can safely show a dialog from wxApp::OnInit and not be afraid that your application terminates when this dialog

– which is the last top level window for the moment – is closed.

Another aspect of the application shutdown is wxApp::OnExit which is called when the application exits but *before* wxWidgets cleans up its internal structures. You should delete all wxWidgets object that you created by the time OnExit finishes.

In particular, do **not** destroy them from application class' destructor! For example, this code may crash:

```
class MyApp : public wxApp
{
public:
    wxCHMHelpController m_helpCtrl;
    ...
};
```

The reason for that is that `m_helpCtrl` is a member object and is thus destroyed from MyApp destructor. But MyApp object is deleted after wxWidgets structures that wxCHMHelpController depends on were uninitialized! The solution is to destroy HelpCtrl in *OnExit:*

```
class MyApp : public wxApp
{
public:
    wxCHMHelpController *m_helpCtrl;
    ...
};

bool MyApp::OnInit()
{
    ...
    m_helpCtrl = new wxCHMHelpController;
    ...
}

int MyApp::OnExit()
{
    delete m_helpCtrl;
    return 0;
}
```

10.13 Unicode Support in wxWidgets

This section describes how does wxWidgets support Unicode and how can it affect your programs.

Notice that Unicode support has changed radically in wxWidgets 3.0 and a lot of existing material pertaining to the previous versions of the library is not correct any more. Please see Unicode-related Changes for the details of these changes.

You can skip the first two sections if you're already familiar with Unicode and wish to jump directly in the details of its support in the library.

10.13.1 What is Unicode?

Unicode is a standard for character encoding which addresses the shortcomings of the previous standards (e.g. the ASCII standard), by using 8, 16 or 32 bits for encoding each character. This allows enough code points (see below for the definition) sufficient to encode all of the world languages at once. More details about Unicode may be found at `http://www.unicode.org/`.

From a practical point of view, using Unicode is almost a requirement when writing applications for international audience. Moreover, any application reading files which it didn't produce or receiving data from the network from other services should be ready to deal with Unicode.

10.13.2 Unicode Representations and Terminology

When working with Unicode, it's important to define the meaning of some terms.

A ***glyph*** is a particular image (usually part of a font) that represents a character or part of a character. Any character may have one or more glyph associated; e.g. some of the possible glyphs for the capital letter 'A' are:

Unicode assigns each character of almost any existing alphabet/script a number, which is called ***code point***; it's typically indicated in documentation manuals and in the Unicode website as `U+xxxx` where `xxxx` is an hexadecimal number.

Note that typically one character is assigned exactly one code point, but there are exceptions; the so-called *precomposed characters* (see `http://en.wikipedia.org/wiki/Precomposed_character`) or the *ligatures*. In these cases a single "character" may be mapped to more than one code point or vice versa more than one character may be mapped to a single code point.

The Unicode standard divides the space of all possible code points in ***planes***; a plane is a range of 65,536 (1000016) contiguous Unicode code points. Planes are numbered from 0 to 16, where the first one is the *BMP*, or Basic Multilingual Plane. The BMP contains characters for all modern languages, and a large number of special characters. The other planes in fact contain mainly historic scripts, special-purpose characters or are unused.

Code points are represented in computer memory as a sequence of one or more ***code units***, where a code unit is a unit of memory: 8, 16, or 32 bits. More precisely, a code unit is the minimal bit combination that can represent a unit of encoded text for processing or interchange.

The ***UTF*** or Unicode Transformation Formats are algorithms mapping the Unicode code points to code unit sequences. The simplest of them is **UTF-32** where each code unit is composed by 32 bits (4 bytes) and each code point is always represented by a single code unit (fixed length encoding). (Note that even UTF-32 is still not completely trivial as the mapping is different for little and big-endian architectures). UTF-32 is commonly used under Unix systems for internal representation of Unicode strings.

Another very widespread standard is **UTF-16** which is used by Microsoft Windows: it encodes the first (approximately) 64 thousands of Unicode code points (the BMP plane) using 16-bit code units (2 bytes) and uses a pair of 16-bit code units to encode the characters beyond this. These pairs are called *surrogate*. Thus UTF16 uses a variable number of code units to encode each code point.

Finally, the most widespread encoding used for the external Unicode storage (e.g. files and network protocols) is **UTF-8** which is byte-oriented and so avoids the endianness ambiguities of UTF-16 and UTF-32. UTF-8 uses code units of 8 bits (1 byte); code points beyond the usual english alphabet are represented using a variable number of bytes, which makes it less efficient than UTF-32 for internal representation.

As visual aid to understand the differences between the various concepts described so far, look at the different UTF representations of the same code point:

In this particular case UTF8 requires more space than UTF16 (3 bytes instead of 2).

Note that from the C/C++ programmer perspective the situation is further complicated by the fact that the standard type `wchar_t` which is usually used to represent the Unicode ("wide") strings in C/C++ doesn't have the same size on all platforms. It is 4 bytes under Unix systems, corresponding to the tradition of using UTF-32, but only 2 bytes under Windows which is required by compatibility with the OS which uses UTF-16.

Typically when UTF8 is used, code units are stored into `char` types, since `char` are 8bit wide on almost all systems; when using UTF16 typically code units are stored into `wchar_t` types since `wchar_t` is at least 16bits on all systems. This is also the approach used by wxString. See wxString Overview for more info.

See also `http://unicode.org/glossary/` for the official definitions of the terms reported above.

10.13.3 Unicode Support in wxWidgets

Unicode is Always Used by Default

Since wxWidgets 3.0 Unicode support is always enabled and while building the library without it is still possible, it is not recommended any longer and will cease to be supported in the near future. This means that internally only Unicode strings are used and that, under Microsoft Windows, Unicode system API is used which means that wxWidgets programs require the Microsoft Layer for Unicode to run on Windows 95/98/ME.

However, unlike the Unicode build mode of the previous versions of wxWidgets, this support is mostly transparent: you can still continue to work with the **narrow** (i.e. current locale-encoded `char*`) strings even if **wide** (i.e. UT↩

F16-encoded `wchar_t*` or UTF8-encoded `char*`) strings are also supported. Any wxWidgets function accepts arguments of either type as both kinds of strings are implicitly converted to wxString, so both

```
wxMessageBox("Hello, world!");
```

and the somewhat less usual

```
wxMessageBox(L"Salut \u00E0 toi!"); // U+00E0 is "Latin Small Letter a with Grave"
```

work as expected.

Notice that the narrow strings used with wxWidgets are *always* assumed to be in the current locale encoding, so writing

```
wxMessageBox("Salut à toi!");
```

wouldn't work if the encoding used on the user system is incompatible with ISO-8859-1 (or even if the sources were compiled under different locale in the case of gcc). In particular, the most common encoding used under modern Unix systems is UTF-8 and as the string above is not a valid UTF-8 byte sequence, nothing would be displayed at all in this case. Thus it is important to **never use 8-bit (instead of 7-bit) characters directly in the program source** but use wide strings or, alternatively, write:

```
wxMessageBox(wxString::FromUTF8("Salut \xC3\xA0 toi!"));
    // in UTF8 the character U+00E0 is encoded as 0xC3A0
```

In a similar way, wxString provides access to its contents as either `wchar_t` or `char` character buffer. Of course, the latter only works if the string contains data representable in the current locale encoding. This will always be the case if the string had been initially constructed from a narrow string or if it contains only 7-bit ASCII data but otherwise this conversion is not guaranteed to succeed. And as with wxString::FromUTF8() example above, you can always use wxString::ToUTF8() to retrieve the string contents in UTF-8 encoding – this, unlike converting to `char*` using the current locale, never fails.

For more info about how wxString works, please see the wxString Overview.

To summarize, Unicode support in wxWidgets is mostly **transparent** for the application and if you use wxString objects for storing all the character data in your program there is really nothing special to do. However you should be aware of the potential problems covered by the following section.

Choosing Unicode Representation

wxWidgets uses the system `wchar_t` in wxString implementation by default under all systems. Thus, under Microsoft Windows, UCS-2 (simplified version of UTF-16 without support for surrogate characters) is used as `wchar_t` is 2 bytes on this platform. Under Unix systems, including Mac OS X, UCS-4 (also known as UTF-32) is used by default, however it is also possible to build wxWidgets to use UTF-8 internally by passing `-enable-utf8` option to configure.

The interface provided by wxString is the same independently of the format used internally. However different formats have specific advantages and disadvantages. Notably, under Unix, the underlying graphical toolkit (e.g. GTK+) usually uses UTF-8 encoded strings and using the same representations for the strings in wxWidgets allows to avoid conversion from UTF-32 to UTF-8 and vice versa each time a string is shown in the UI or retrieved from it. The overhead of such conversions is usually negligible for small strings but may be important for some programs. If you believe that it would be advantageous to use UTF-8 for the strings in your particular application, you may rebuild wxWidgets to use UTF-8 as explained above (notice that this is currently not supported under Microsoft Windows and arguably doesn't make much sense there as Windows itself uses UTF-16 and not UTF-8) but be sure to be aware of the performance implications (see Performance Implications of Using UTF-8) of using UTF-8 in wxString before doing this!

Generally speaking you should only use non-default UTF-8 build in specific circumstances e.g. building for resource-constrained systems where the overhead of conversions (and also reduced memory usage of UTF-8 compared to UTF-32 for the European languages) can be important. If the environment in which your program is running is under your control – as is quite often the case in such scenarios – consider ensuring that the system always uses UTF-8 locale and use `-enable-utf8only` configure option to disable support for the other locales and consider all strings to be in UTF-8. This further reduces the code size and removes the need for conversions in more cases.

Unicode Related Preprocessor Symbols

wxUSE_UNICODE is defined as 1 now to indicate Unicode support. It can be explicitly set to 0 in setup.h under MSW or you can use --disable-unicode under Unix but doing this is strongly discouraged. By default, wxUSE_UNICODE_WCHAR is also defined as 1, however in UTF-8 build (described in the previous section), it is set to 0 and wxUSE_UNICODE_UTF8, which is usually 0, is set to 1 instead. In the latter case, wxUSE_UTF8_LOCALE_ONLY can also be set to 1 to indicate that all strings are considered to be in UTF-8.

10.13.4 Potential Unicode Pitfalls

The problems can be separated into three broad classes:

Unicode-Related Compilation Errors

Because of the need to support implicit conversions to both char and wchar_t, wxString implementation is rather involved and many of its operators don't return the types which they could be naively expected to return. For example, the operator[] doesn't return neither a char nor a wchar_t but an object of a helper class wxUniChar or wxUniCharRef which is implicitly convertible to either. Usually you don't need to worry about this as the conversions do their work behind the scenes however in some cases it doesn't work. Here are some examples, using a wxString object s and some integer n:

- Writing

  ```
  switch ( s[n] )
  ```

  doesn't work because the argument of the switch statement must be an integer expression so you need to replace s[n] with

  ```
  s[n].GetValue()
  ```

  . You may also force the conversion to char or wchar_t by using an explicit cast but beware that converting the value to char uses the conversion to current locale and may return 0 if it fails. Finally notice that writing

  ```
  (wxChar)s[n]
  ```

  works both with wxWidgets 3.0 and previous library versions and so should be used for writing code which should be compatible with both 2.8 and 3.0.

- Similarly,

  ```
  &s[n]
  ```

  doesn't yield a pointer to char so you may not pass it to functions expecting char* or wchar_t*. Consider using string iterators instead if possible or replace this expression with

  ```
  s.c_str() + n
  ```

  otherwise.

Another class of problems is related to the fact that the value returned by c_str() itself is also not just a pointer to a buffer but a value of helper class wxCStrData which is implicitly convertible to both narrow and wide strings. Again, this mostly will be unnoticeable but can result in some problems:

- You shouldn't pass c_str() result to vararg functions such as standard printf(). Some compilers (notably g++) warn about this but even if they don't, this

  ```
  printf("Hello, %s", s.c_str())
  ```

is not going to work. It can be corrected in one of the following ways:

- Preferred:

```
wxPrintf("Hello, %s", s)
```

(notice the absence of `c_str()`, it is not needed at all with wxWidgets functions)

- Compatible with wxWidgets 2.8:

```
wxPrintf("Hello, %s", s.c_str())
```

- Using an explicit conversion to narrow, multibyte, string:

```
printf("Hello, %s", (const char *)s.mb_str())
```

- Using a cast to force the issue (listed only for completeness):

```
printf("Hello, %s", (const char *)s.c_str())
```

- The result of `c_str()` cannot be cast to `char*` but only to `const char*`. Of course, modifying the string via the pointer returned by this method has never been possible but unfortunately it was occasionally useful to use a `const_cast` here to pass the value to const-incorrect functions. This can be done either using new wxString::char_str() (and matching wchar_str()) method or by writing a double cast:

```
(char *)(const char *)s.c_str()
```

- One of the unfortunate consequences of the possibility to pass wxString to `wxPrintf()` without using `c_str()` is that it is now impossible to pass the elements of unnamed enumerations to `wxPrintf()` and other similar vararg functions, i.e.

```
enum { Red, Green, Blue };
wxPrintf("Red is %d", Red);
```

doesn't compile. The easiest workaround is to give a name to the enum.

Other unexpected compilation errors may arise but they should happen even more rarely than the above-mentioned ones and the solution should usually be quite simple: just use the explicit methods of wxUniChar and wxCStrData classes instead of relying on their implicit conversions if the compiler can't choose among them.

Data Loss due To Unicode Conversion Errors

wxString API provides implicit conversion of the internal Unicode string contents to narrow, char strings. This can be very convenient and is absolutely necessary for backwards compatibility with the existing code using wxWidgets however it is a rather dangerous operation as it can easily give unexpected results if the string contents isn't convertible to the current locale.

To be precise, the conversion will always succeed if the string was created from a narrow string initially. It will also succeed if the current encoding is UTF-8 as all Unicode strings are representable in this encoding. However initializing the string using wxString::FromUTF8() method and then accessing it as a char string via its wxString::c_str() method is a recipe for disaster as the program may work perfectly well during testing on Unix systems using UTF-8 locale but completely fail under Windows where UTF-8 locales are never used because wxString::c_str() would return an empty string.

The simplest way to ensure that this doesn't happen is to avoid conversions to `char*` completely by using wxString throughout your program. However if the program never manipulates 8 bit strings internally, using `char*` pointers is safe as well. So the existing code needs to be reviewed when upgrading to wxWidgets 3.0 and the new code should be used with this in mind and ideally avoiding implicit conversions to `char*`.

Performance Implications of Using UTF-8

As mentioned above, under Unix systems wxString class can use variable-width UTF-8 encoding for internal representation. In this case it can't guarantee constant-time access to N-th element of the string any longer as to find the position of this character in the string we have to examine all the preceding ones. Usually this doesn't matter much because most algorithms used on the strings examine them sequentially anyhow and because wxString implements a cache for iterating over the string by index but it can have serious consequences for algorithms using random access to string elements as they typically acquire O(N^2) time complexity instead of O(N) where N is the length of the string.

Even despite caching the index, indexed access should be replaced with sequential access using string iterators. For example a typical loop:

```
wxString s("hello");
for ( size_t i = 0; i < s.length(); i++ )
{
    wchar_t ch = s[i];

    // do something with it
}
```

should be rewritten as

```
wxString s("hello");
for ( wxString::const_iterator i = s.begin(); i != s.end(); ++i )
{
    wchar_t ch = *i

    // do something with it
}
```

Another, similar, alternative is to use pointer arithmetic:

```
wxString s("hello");
for ( const wchar_t *p = s.wc_str(); *p; p++ )
{
    wchar_t ch = *i

    // do something with it
}
```

however this doesn't work correctly for strings with embedded `NUL` characters and the use of iterators is generally preferred as they provide some run-time checks (at least in debug build) unlike the raw pointers. But if you do use them, it is better to use `wchar_t` pointers rather than `char` ones to avoid the data loss problems due to conversion as discussed in the previous section.

10.13.5 Unicode and the Outside World

Even though wxWidgets always uses Unicode internally, not all the other libraries and programs do and even those that do use Unicode may use a different encoding of it. So you need to be able to convert the data to various representations and the wxString methods wxString::ToAscii(), wxString::ToUTF8() (or its synonym wxString::utf8_str()), wxString::mb_str(), wxString::c_str() and wxString::wc_str() can be used for this.

The first of them should be only used for the string containing 7-bit ASCII characters only, anything else will be replaced by some substitution character. wxString::mb_str() converts the string to the encoding used by the current locale and so can return an empty string if the string contains characters not representable in it as explained in Data Loss due To Unicode Conversion Errors. The same applies to wxString::c_str() if its result is used as a narrow string. Finally, wxString::ToUTF8() and wxString::wc_str() functions never fail and always return a pointer to char string containing the UTF-8 representation of the string or `wchar_t` string.

wxString also provides two convenience functions: wxString::From8BitData() and wxString::To8BitData(). They can be used to create a wxString from arbitrary binary data without supposing that it is in current locale encoding, and then get it back, again, without any conversion or, rather, undoing the conversion used by wxString::From8BitData(). Because of this you should only use wxString::From8BitData() for the strings created using wxString::To8BitData(). Also notice that in spite of the availability of these functions, wxString is not the ideal class for storing arbitrary binary

data as they can take up to 4 times more space than needed (when using `wchar_t` internal representation on the systems where size of wide characters is 4 bytes) and you should consider using wxMemoryBuffer instead.

Final word of caution: most of these functions may return either directly the pointer to internal string buffer or a temporary wxCharBuffer or wxWCharBuffer object. Such objects are implicitly convertible to `char` and `wchar_t` pointers, respectively, and so the result of, for example, wxString::ToUTF8() can always be passed directly to a function taking `const char*`. However code such as

```
const char *p = s.ToUTF8();
...
puts(p); // or call any other function taking const char *
```

does **not** work because the temporary buffer returned by wxString::ToUTF8() is destroyed and `p` is left pointing nowhere. To correct this you should use

```
const wxScopedCharBuffer p(s.ToUTF8());
puts(p);
```

which does work.

Similarly, wxWX2WCbuf can be used for the return type of wxString::wc_str(). But, once again, none of these cryptic types is really needed if you just pass the return value of any of the functions mentioned in this section to another function directly.

10.14 Internationalization

Although internationalization of an application (i18n for short) involves far more than just translating its text messages to another message - date, time and currency formats need changing too, some languages are written left to right and others right to left, character encoding may differ and many other things may need changing too - it is a necessary first step.

wxWidgets provides facilities for message translation with its wxLocale class and is itself fully translated into several languages. Please consult wxWidgets home page for the most up-to-date translations - and if you translate it into one of the languages not done yet, your translations would be gratefully accepted for inclusion into future versions of the library!

The wxWidgets approach to i18n closely follows the GNU gettext package. wxWidgets uses the message catalogs which are binary compatible with gettext catalogs and this allows to use all of the programs in this package to work with them as well as using any of the tools working with message catalogs in this format such as `Poedit`.

Because of this, you will need to use the gettext package to work with the translations during the program development. However no additional libraries are needed during run-time, so you have only the message catalogs to distribute and nothing else.

There are two kinds of message catalogs: source catalogs which are text files with extension .po and binary catalogs which are created from the source ones with *msgfmt* program (part of gettext package) and have the extension .mo. Only the binary files are needed during program execution.

Translating your application involves several steps:

- Translating the strings in the program text using wxGetTranslation or equivalently the `_()` macro.
- Extracting the strings to be translated from the program: this uses the work done in the previous step because `xgettext` program used for string extraction recognises the standard `_()` as well as (using its `-k` option) our wxGetTranslation and extracts all strings inside the calls to these functions. Alternatively, you may use `-a` option to extract all the strings, but it will usually result in many strings being found which don't have to be translated at all. This will create a text message catalog - a .po file.
- Translating the strings extracted in the previous step to other language(s). It involves editing the .po file.
- Compiling the .po file into .mo file to be used by the program.

- Installing the .mo files with your application in the appropriate location for the target system (see Installing translation catalogs).
- Setting the appropriate locale in your program to use the strings for the given language: see wxLocale.

10.14.1 Installing translation catalogs

The .mo files with compiled catalogs must be included with the application. By default, wxFileTranslationsLoader is used to load them from files installed alongside the application (although you could use wxResourceTranslationsLoader or some custom loader too).

The files are expected to be in the resources directory (as returned by wxStandardPaths::GetLocalizedResourcesDir(wxStandardPaths::ResourceCat_Messages). If the message catalogs are not installed in this default location you may explicitly use wxFileTranslationsLoader::AddCatalogLookupPathPrefix() to still allow wxWidgets to find them, but it is recommended to use the default locations when possible.

Depending on the platform, the default location differs. On Windows, it is alongside the executable. On Unix, translations are expected to be in "$prefix/share/locale". On OS X, application bundle's *Resources* subdirectory is used.

In all cases, translations are searched for in subdirectories named using the languages codes from ISO 639. The .mo file(s) should be located either directly in that directory or in LC_MESSAGES subdirectory. On OS X, ".lproj" extension is used for the per-languages Resources subdirectories.

Here's how an app would typically install the files on Unix:

```
/usr/bin/myapp
/usr/share/locale/de/LC_MESSAGES/myapp.mo
/usr/share/locale/fr/LC_MESSAGES/myapp.mo
```

And on OS X:

```
MyApp.app/Contents/MacOS/MyApp
MyApp.app/Contents/Resources/de.lproj/myapp.mo
MyApp.app/Contents/Resources/fr.lproj/myapp.mo
```

And on Windows:

```
C:\Program Files\MyApp\myapp.exe
C:\Program Files\MyApp\de\myapp.mo
C:\Program Files\MyApp\fr\myapp.mo
```

It is of course possible to use the Unix layout everywhere instead.

10.14.2 Translating Menu Accelerators

If you translate the accelerator modifier names (Ctrl, Alt and Shift) in your menu labels, you may find the accelerators no longer work. In your message catalogs, you need to provide individual translations of these modifiers from their lower case names (ctrl, alt, shift) so that the wxWidgets accelerator code can recognise them even when translated. wxWidgets does not provide translations for all of these currently. wxWidgets does not yet handle translated special key names such as Backspace, End, Insert, etc.

See also

- The gettext Manual: http://www.gnu.org/software/gettext/manual/gettext.html
- Writing Non-English Applications - It focuses on handling charsets related problems.
- Internationalization Sample - Shows you how all this looks in practice.

10.15 Events and Event Handling

Like with all the other GUI frameworks, the control of flow in wxWidgets applications is event-based: the program normally performs most of its actions in response to the events generated by the user.

These events can be triggered by using the input devices (such as keyboard, mouse, joystick) directly or, more commonly, by a standard control which synthesizes such input events into higher level events: for example, a wxButton can generate a click event when the user presses the left mouse button on it and then releases it without pressing `Esc` in the meanwhile. There are also events which don't directly correspond to the user actions, such as wxTimerEvent or wxSocketEvent.

But in all cases wxWidgets represents these events in a uniform way and allows you to handle them in the same way wherever they originate from. And while the events are normally generated by wxWidgets itself, you can also do this, which is especially useful when using custom events (see Custom Event Summary).

To be more precise, each event is described by:

- *Event type*: this is simply a value of type wxEventType which uniquely identifies the type of the event. For example, clicking on a button, selecting an item from a list box and pressing a key on the keyboard all generate events with different event types.
- *Event class* carried by the event: each event has some information associated with it and this data is represented by an object of a class derived from wxEvent. Events of different types can use the same event class, for example both button click and listbox selection events use wxCommandEvent class (as do all the other simple control events), but the key press event uses wxKeyEvent as the information associated with it is different.
- *Event source*: wxEvent stores the object which generated the event and, for windows, its identifier (see Window Identifiers). As it is common to have more than one object generating events of the same type (e.g. a typical window contains several buttons, all generating the same button click event), checking the event source object or its id allows to distinguish between them.

See also

wxEvtHandler, wxWindow, wxEvent

10.15.1 Event Handling

There are two principal ways to handle events in wxWidgets. One of them uses *event table* macros and allows you to define the binding between events and their handlers only statically, i.e., during program compilation. The other one uses wxEvtHandler::Bind<>() call and can be used to bind and unbind, the handlers dynamically, i.e. during run-time depending on some conditions. It also allows the direct binding of events to:

- A handler method in another object.
- An ordinary function like a static method or a global function.
- An arbitrary functor like boost::function<>.

The static event tables can only handle events in the object where they are defined so using Bind<>() is more flexible than using the event tables. On the other hand, event tables are more succinct and centralize all event handler bindings in one place. You can either choose a single approach that you find preferable or freely combine both methods in your program in different classes or even in one and the same class, although this is probably sufficiently confusing to be a bad idea.

Also notice that most of the existing wxWidgets tutorials and discussions use the event tables because they historically preceded the apparition of dynamic event handling in wxWidgets. But this absolutely doesn't mean that using the event tables is the preferred way: handling events dynamically is better in several aspects and you should strongly consider doing it if you are just starting with wxWidgets. On the other hand, you still need to know about the event tables if only because you are going to see them in many samples and examples.

So before you make the choice between static event tables and dynamically connecting the event handlers, let us discuss these two ways in more detail. In the next section we provide a short introduction to handling the events using the event tables. Please see Dynamic Event Handling for the discussion of Bind<>().

Event Handling with Event Tables

To use an *event table* you must first decide in which class you wish to handle the events. The only requirement imposed by wxWidgets is that this class must derive from wxEvtHandler and so, considering that wxWindow derives from it, any classes representing windows can handle events. Simple events such as menu commands are usually processed at the level of a top-level window containing the menu, so let's suppose that you need to handle some events in `MyFrame` class deriving from wxFrame.

First define one or more *event handlers*. They are just simple methods of the class that take as a parameter a reference to an object of a wxEvent-derived class and have no return value (any return information is passed via the argument, which is why it is non-const). You also need to insert a macro

```
wxDECLARE_EVENT_TABLE()
```

somewhere in the class declaration. It doesn't matter where it appears but it's customary to put it at the end because the macro changes the access type internally so it's safest if nothing follows it. The full class declaration might look like this:

```
class MyFrame : public wxFrame
{
public:
    MyFrame(...) : wxFrame(...) { }

    ...

protected:
    int m_whatever;

private:
    // Notice that as the event handlers normally are not called from outside
    // the class, they normally are private. In particular they don't need
    // to be public.
    void OnExit(wxCommandEvent& event);
    void OnButton1(wxCommandEvent& event);
    void OnSize(wxSizeEvent& event);

    // it's common to call the event handlers OnSomething() but there is no
    // obligation to do that; this one is an event handler too:
    void DoTest(wxCommandEvent& event);

    wxDECLARE_EVENT_TABLE()
};
```

Next the event table must be defined and, as with any definition, it must be placed in an implementation file. The event table tells wxWidgets how to map events to member functions and in our example it could look like this:

```
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(wxID_EXIT, MyFrame::OnExit)
    EVT_MENU(DO_TEST, MyFrame::DoTest)
    EVT_SIZE(MyFrame::OnSize)
    EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
wxEND_EVENT_TABLE()
```

Notice that you must mention a method you want to use for the event handling in the event table definition; just defining it in MyFrame class is *not* enough.

Let us now look at the details of this definition: the first line means that we are defining the event table for My↩ Frame class and that its base class is wxFrame, so events not processed by MyFrame will, by default, be handled by wxFrame. The next four lines define bindings of individual events to their handlers: the first two of them map menu commands from the items with the identifiers specified as the first macro parameter to two different member functions. In the next one, `EVT_SIZE` means that any changes in the size of the frame will result in calling OnSize() method. Note that this macro doesn't need a window identifier, since normally you are only interested in the current window's size events.

The EVT_BUTTON macro demonstrates that the originating event does not have to come from the window class implementing the event table – if the event source is a button within a panel within a frame, this will still work, because event tables are searched up through the hierarchy of windows for the command events. (But only command events, so you can't catch mouse move events in a child control in the parent window in the same way because wxMouseEvent doesn't derive from wxCommandEvent. See below for how you can do it.) In this case, the button's event table will be searched, then the parent panel's, then the frame's.

Finally, you need to implement the event handlers. As mentioned before, all event handlers take a wxEvent-derived argument whose exact class differs according to the type of event and the class of the originating window. For size events, wxSizeEvent is used. For menu commands and most control commands (such as button presses), wxCommandEvent is used. When controls get more complicated, more specific wxCommandEvent-derived event classes providing additional control-specific information can be used, such as wxTreeEvent for events from wxTreeCtrl windows.

In the simplest possible case an event handler may not use the event parameter at all. For example,

```
void MyFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    // when the user selects "Exit" from the menu we should close
    Close(true);
}
```

In other cases you may need some information carried by the event argument, as in:

```
void MyFrame::OnSize(wxSizeEvent& event)
{
    wxSize size = event.GetSize();

    ... update the frame using the new size ...
}
```

You will find the details about the event table macros and the corresponding wxEvent-derived classes in the discussion of each control generating these events.

Dynamic Event Handling

See also

Caveats When Not Using C++ RTTI

The possibilities of handling events in this way are rather different. Let us start by looking at the syntax: the first obvious difference is that you need not use wxDECLARE_EVENT_TABLE() nor wxBEGIN_EVENT_TABLE() and the associated macros. Instead, in any place in your code, but usually in the code of the class defining the handler itself (and definitely not in the global scope as with the event tables), call its Bind<>() method like this:

```
MyFrame::MyFrame(...)
{
      Bind(wxEVT_COMMAND_MENU_SELECTED, &MyFrame::OnExit, this, wxID_EXIT);
}
```

Note that this pointer must be specified here.

Now let us describe the semantic differences:

- Event handlers can be bound at any moment. For example, it's possible to do some initialization first and only bind the handlers if and when it succeeds. This can avoid the need to test that the object was properly initialized in the event handlers themselves. With Bind<>() they simply won't be called if it wasn't correctly initialized.
- As a slight extension of the above, the handlers can also be unbound at any time with Unbind<>() (and maybe rebound later). Of course, it's also possible to emulate this behaviour with the classic static (i.e., bound via event tables) handlers by using an internal flag indicating whether the handler is currently enabled and returning from it if it isn't, but using dynamically bind handlers requires less code and is also usually more clear.

- Almost last but very, very far from least is the increased flexibility which allows to bind an event to:
  - A method in another object.
  - An ordinary function like a static method or a global function.
  - An arbitrary functor like boost::function<>.

  This is impossible to do with the event tables because it is not possible to specify these handlers to dispatch the event to, so it necessarily needs to be sent to the same object which generated the event. Not so with Bind<>() which can be used to specify these handlers which will handle the event. To give a quick example, a common question is how to receive the mouse movement events happening when the mouse is in one of the frame children in the frame itself. Doing it in a naive way doesn't work:
  - A `EVT_LEAVE_WINDOW(MyFrame::OnMouseLeave)` line in the frame event table has no effect as mouse move (including entering and leaving) events are not propagated up to the parent window (at least not by default).
  - Putting the same line in a child event table will crash during run-time because the MyFrame method will be called on a wrong object – it's easy to convince oneself that the only object that can be used here is the pointer to the child, as wxWidgets has nothing else. But calling a frame method with the child window pointer instead of the pointer to the frame is, of course, disastrous.

  However writing

  ```
  MyFrame::MyFrame(...)
  {
    m_child->Bind(wxEVT_LEAVE_WINDOW, &MyFrame::OnMouseLeave, this);
  }
  ```

  will work exactly as expected. Note that you can get the object that generated the event – and that is not the same as the frame – via wxEvent::GetEventObject() method of `event` argument passed to the event handler.
- Really last point is the consequence of the previous one: because of increased flexibility of Bind(), it is also safer as it is impossible to accidentally use a method of another class. Instead of run-time crashes you will get compilation errors in this case when using Bind().

Let us now look at more examples of how to use different event handlers using the two overloads of Bind() function: first one for the object methods and the other one for arbitrary functors (callable objects, including simple functions):

In addition to using a method of the object generating the event itself, you can use a method from a completely different object as an event handler:

```
void MyFrameHandler::OnFrameExit( wxCommandEvent & )
{
    // Do something useful.
}

MyFrameHandler myFrameHandler;

MyFrame::MyFrame()
{
      Bind( wxEVT_COMMAND_MENU_SELECTED, &MyFrameHandler::OnFrameExit,
              &myFrameHandler, wxID_EXIT );
}
```

Note that `MyFrameHandler` doesn't need to derive from wxEvtHandler. But keep in mind that then the lifetime of `myFrameHandler` must be greater than that of `MyFrame` object – or at least it needs to be unbound before being destroyed.

To use an ordinary function or a static method as an event handler you would write something like this:

```
void HandleExit( wxCommandEvent & )
{
    // Do something useful
}

MyFrame::MyFrame()
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &HandleExit, wxID_EXIT );
}
```

And finally you can bind to an arbitrary functor and use it as an event handler:

```
struct MyFunctor
{
    void operator()( wxCommandEvent & )
    {
        // Do something useful
    }
};

MyFunctor myFunctor;

MyFrame::MyFrame()
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, myFunctor, wxID_EXIT );
}
```

A common example of a functor is boost::function<>:

```
using namespace boost;

void MyHandler::OnExit( wxCommandEvent & )
{
    // Do something useful
}

MyHandler myHandler;

MyFrame::MyFrame()
{
    function< void ( wxCommandEvent & ) > exitHandler( bind( &MyHandler::OnExit, &myHandler, _1 ));

    Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
}
```

With the aid of boost::bind<>() you can even use methods or functions which don't quite have the correct signature:

```
void MyHandler::OnExit( int exitCode, wxCommandEvent &, wxString goodByeMessage )
{
    // Do something useful
}

MyHandler myHandler;

MyFrame::MyFrame()
{
    function< void ( wxCommandEvent & ) > exitHandler(
            bind( &MyHandler::OnExit, &myHandler, EXIT_FAILURE, _1, "Bye" ));

    Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
}
```

To summarize, using Bind<>() requires slightly more typing but is much more flexible than using static event tables so don't hesitate to use it when you need this extra power. On the other hand, event tables are still perfectly fine in simple situations where this extra flexibility is not needed.

10.15.2 How Events are Processed

The previous sections explain how to define event handlers but don't address the question of how exactly wxWidgets finds the handler to call for the given event. This section describes the algorithm used in detail. Notice that you may want to run the Event Sample while reading this section and look at its code and the output when the button which can be used to test the event handlers execution order is clicked to understand it better.

When an event is received from the windowing system, wxWidgets calls wxEvtHandler::ProcessEvent() on the first event handler object belonging to the window generating the event. The normal order of event table searching by ProcessEvent() is as follows, with the event processing stopping as soon as a handler is found (unless the handler calls wxEvent::Skip() in which case it doesn't count as having handled the event and the search continues):

1. Before anything else happens, wxApp::FilterEvent() is called. If it returns anything but -1 (default), the event handling stops immediately.

2. If this event handler is disabled via a call to wxEvtHandler::SetEvtHandlerEnabled() the next three steps are skipped and the event handler resumes at step (5).
3. If the object is a wxWindow and has an associated validator, wxValidator gets a chance to process the event.
4. The list of dynamically bound event handlers, i.e., those for which Bind<>() was called, is consulted. Notice that this is done before checking the static event table entries, so if both a dynamic and a static event handler match the same event, the static one is never going to be used unless wxEvent::Skip() is called in the dynamic one. Also note that the dynamically bound handlers are searched in order of their registration during program run-time, i.e. later bound handlers take priority over the previously bound ones.
5. The event table containing all the handlers defined using the event table macros in this class and its base classes is examined. The search in an event table respects the order of the event macros appearance in the source code, i.e. earlier occurring entries take precedence over later occurring ones. Notice that this means that any event handler defined in a base class will be executed at this step.
6. The event is passed to the next event handler, if any, in the event handler chain, i.e., the steps (1) to (4) are done for it. Usually there is no next event handler so the control passes to the next step but see Event Handlers Chain for how the next handler may be defined.
7. If the object is a wxWindow and the event is set to propagate (by default only wxCommandEvent-derived events are set to propagate), then the processing restarts from the step (1) (and excluding the step (7)) for the parent window. If this object is not a window but the next handler exists, the event is passed to its parent if it is a window. This ensures that in a common case of (possibly several) non-window event handlers pushed on top of a window, the event eventually reaches the window parent.
8. Finally, i.e., if the event is still not processed, the wxApp object itself (which derives from wxEvtHandler) gets a last chance to process it.

Please pay close attention to step 6! People often overlook or get confused by this powerful feature of the wxWidgets event processing system. The details of event propagation up the window hierarchy are described in the next section.

Also please notice that there are additional steps in the event handling for the windows-making part of wxWidgets document-view framework, i.e., wxDocParentFrame, wxDocChildFrame and their MDI equivalents wxDocMDIParentFrame and wxDocMDIChildFrame. The parent frame classes modify step (2) above to send the events received by them to wxDocManager object first. This object, in turn, sends the event to the current view and the view itself lets its associated document process the event first. The child frame classes send the event directly to the associated view which still forwards it to its document object. Notice that to avoid remembering the exact order in which the events are processed in the document-view frame, the simplest, and recommended, solution is to only handle the events at the view classes level, and not in the document or document manager classes

How Events Propagate Upwards

As mentioned above, the events of the classes deriving from wxCommandEvent are propagated by default to the parent window if they are not processed in this window itself. But although by default only the command events are propagated like this, other events can be propagated as well because the event handling code uses wxEvent::ShouldPropagate() to check whether an event should be propagated. It is also possible to propagate the event only a limited number of times and not until it is processed (or a top level parent window is reached).

Finally, there is another additional complication (which, in fact, simplifies life of wxWidgets programmers significantly): when propagating the command events up to the parent window, the event propagation stops when it reaches the parent dialog, if any. This means that you don't risk getting unexpected events from the dialog controls (which might be left unprocessed by the dialog itself because it doesn't care about them) when a modal dialog is popped up. The events do propagate beyond the frames, however. The rationale for this choice is that there are only a few frames in a typical application and their parent-child relation are well understood by the programmer while it may be difficult, if not impossible, to track down all the dialogs that may be popped up in a complex program (remember that some are created automatically by wxWidgets). If you need to specify a different behaviour for some reason, you can use `wxWindow::SetExtraStyle(wxWS_EX_BLOCK_EVENTS)` explicitly to prevent the events from being propagated beyond the given window or unset this flag for the dialogs that have it on by default.

Typically events that deal with a window as a window (size, motion, paint, mouse, keyboard, etc.) are sent only to the window. Events that have a higher level of meaning or are generated by the window itself (button click, menu select, tree expand, etc.) are command events and are sent up to the parent to see if it is interested in the event. More precisely, as said above, all event classes **not** deriving from wxCommandEvent (see the wxEvent inheritance map) do **not** propagate upward.

In some cases, it might be desired by the programmer to get a certain number of system events in a parent window, for example all key events sent to, but not used by, the native controls in a dialog. In this case, a special event handler will have to be written that will override ProcessEvent() in order to pass all events (or any selection of them) to the parent window.

Event Handlers Chain

The step 4 of the event propagation algorithm checks for the next handler in the event handler chain. This chain can be formed using wxEvtHandler::SetNextHandler(): (referring to the image, if `A->ProcessEvent` is called and it doesn't handle the event, `B->ProcessEvent` will be called and so on...).

Additionally, in the case of wxWindow you can build a stack (implemented using wxEvtHandler double-linked list) using wxWindow::PushEventHandler(): (referring to the image, if `W->ProcessEvent` is called, it immediately calls `A->ProcessEvent`; if nor `A` nor `B` handle the event, then the wxWindow itself is used – i.e. the dynamically bind event handlers and static event table entries of wxWindow are looked as the last possibility, after all pushed event handlers were tested).

By default the chain is empty, i.e. there is no next handler.

10.15.3 Custom Event Summary

General approach

As each event is uniquely defined by its event type, defining a custom event starts with defining a new event type for it. This is done using wxDEFINE_EVENT() macro. As an event type is a variable, it can also be declared using wxDECLARE_EVENT() if necessary.

The next thing to do is to decide whether you need to define a custom event class for events of this type or if you can reuse an existing class, typically either wxEvent (which doesn't provide any extra information) or wxCommandEvent (which contains several extra fields and also propagates upwards by default). Both strategies are described in details below. See also the Event Sample for a complete example of code defining and working with the custom event types.

Finally, you will need to generate and post your custom events. Generation is as simple as instancing your custom event class and initializing its internal fields. For posting events to a certain event handler there are two possibilities: using wxEvtHandler::AddPendingEvent or using wxEvtHandler::QueueEvent. Basically you will need to use the latter when doing inter-thread communication; when you use only the main thread you can also safely use the former. Last, note that there are also two simple global wrapper functions associated to the two wxEvtHandler mentioned functions: wxPostEvent() and wxQueueEvent().

Using Existing Event Classes

If you just want to use a wxCommandEvent with a new event type, use one of the generic event table macros listed below, without having to define a new event class yourself.

Example:

```
// this is typically in a header: it just declares MY_EVENT event type
wxDECLARE_EVENT(MY_EVENT, wxCommandEvent);

// this is a definition so can't be in a header
wxDEFINE_EVENT(MY_EVENT, wxCommandEvent);

// example of code handling the event with event tables
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU     (wxID_EXIT, MyFrame::OnExit)
```

```
    ...
    EVT_COMMAND (ID_MY_WINDOW, MY_EVENT, MyFrame::OnMyEvent)
wxEND_EVENT_TABLE()

void MyFrame::OnMyEvent(wxCommandEvent& event)
{
    // do something
    wxString text = event.GetString();
}

// example of code handling the event with Bind<>():
MyFrame::MyFrame()
{
    Bind(MY_EVENT, &MyFrame::OnMyEvent, this, ID_MY_WINDOW);
}

// example of code generating the event
void MyWindow::SendEvent()
{
    wxCommandEvent event(MY_EVENT, GetId());
    event.SetEventObject(this);

    // Give it some contents
    event.SetString("Hello");

    // Do send it
    ProcessWindowEvent(event);
}
```

Defining Your Own Event Class

Under certain circumstances, you must define your own event class e.g., for sending more complex data from one place to another. Apart from defining your event class, you also need to define your own event table macro if you want to use event tables for handling events of this type.

Here is an example:

```
// define a new event class
class MyPlotEvent: public wxEvent
{
public:
    MyPlotEvent(wxEventType eventType, int winid, const wxPoint& pos)
        : wxEvent(winid, eventType),
          m_pos(pos)
    {
    }

    // accessors
    wxPoint GetPoint() const { return m_pos; }

    // implement the base class pure virtual
    virtual wxEvent *Clone() const { return new MyPlotEvent(*this); }

private:
    const wxPoint m_pos;
};

// we define a single MY_PLOT_CLICKED event type associated with the class
// above but typically you are going to have more than one event type, e.g. you
// could also have MY_PLOT_ZOOMED or MY_PLOT_PANNED &c -- in which case you
// would just add more similar lines here
wxDEFINE_EVENT(MY_PLOT_CLICKED, MyPlotEvent);


// if you want to support old compilers you need to use some ugly macros:
typedef void (wxEvtHandler::*MyPlotEventFunction)(MyPlotEvent&);
#define MyPlotEventHandler(func) wxEVENT_HANDLER_CAST(MyPlotEventFunction, func)

// if your code is only built using reasonably modern compilers, you could just
// do this instead:
#define MyPlotEventHandler(func) (&func)

// finally define a macro for creating the event table entries for the new
// event type
//
// remember that you don't need this at all if you only use Bind<>() and that
// you can replace MyPlotEventHandler(func) with just &func unless you use a
// really old compiler
#define MY_EVT_PLOT_CLICK(id, func) \
    wx__DECLARE_EVT1(MY_PLOT_CLICKED, id, MyPlotEventHandler(func))
```

```
// example of code handling the event (you will use one of these methods, not
// both, of course):
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_PLOT(ID_MY_WINDOW, MyFrame::OnPlot)
wxEND_EVENT_TABLE()

MyFrame::MyFrame()
{
    Bind(MY_PLOT_CLICKED, &MyFrame::OnPlot, this, ID_MY_WINDOW);
}

void MyFrame::OnPlot(MyPlotEvent& event)
{
    ... do something with event.GetPoint() ...
}


// example of code generating the event:
void MyWindow::SendEvent()
{
    MyPlotEvent event(MY_PLOT_CLICKED, GetId(), wxPoint(...));
    event.SetEventObject(this);
    ProcessWindowEvent(event);
}
```

10.15.4 Miscellaneous Notes

Event Handlers vs Virtual Methods

It may be noted that wxWidgets' event processing system implements something close to virtual methods in normal C++ in spirit: both of these mechanisms allow you to alter the behaviour of the base class by defining the event handling functions in the derived classes.

There is however an important difference between the two mechanisms when you want to invoke the default behaviour, as implemented by the base class, from a derived class handler. With the virtual functions, you need to call the base class function directly and you can do it either in the beginning of the derived class handler function (to post-process the event) or at its end (to pre-process the event). With the event handlers, you only have the option of pre-processing the events and in order to still let the default behaviour happen you must call wxEvent::Skip() and *not* call the base class event handler directly. In fact, the event handler probably doesn't even exist in the base class as the default behaviour is often implemented in platform-specific code by the underlying toolkit or OS itself. But even if it does exist at wxWidgets level, it should never be called directly as the event handlers are not part of wxWidgets API and should never be called directly.

User Generated Events vs Programmatically Generated Events

While generically wxEvents can be generated both by user actions (e.g., resize of a wxWindow) and by calls to functions (e.g., wxWindow::SetSize), wxWidgets controls normally send wxCommandEvent-derived events only for the user-generated events. The only **exceptions** to this rule are:

- wxNotebook::AddPage: No event-free alternatives
- wxNotebook::AdvanceSelection: No event-free alternatives
- wxNotebook::DeletePage: No event-free alternatives
- wxNotebook::SetSelection: Use wxNotebook::ChangeSelection instead, as wxNotebook::SetSelection is deprecated
- wxTreeCtrl::Delete: No event-free alternatives
- wxTreeCtrl::DeleteAllItems: No event-free alternatives
- wxTreeCtrl::EditLabel: No event-free alternatives
- All wxTextCtrl methods

wxTextCtrl::ChangeValue can be used instead of wxTextCtrl::SetValue but the other functions, such as wxTextCtrl::Replace or wxTextCtrl::WriteText don't have event-free equivalents.

Pluggable Event Handlers

TODO: Probably deprecated, Bind() provides a better way to do this

In fact, you don't have to derive a new class from a window class if you don't want to. You can derive a new class from wxEvtHandler instead, defining the appropriate event table, and then call wxWindow::SetEventHandler (or, preferably, wxWindow::PushEventHandler) to make this event handler the object that responds to events. This way, you can avoid a lot of class derivation, and use instances of the same event handler class (but different objects as the same event handler object shouldn't be used more than once) to handle events from instances of different widget classes.

If you ever have to call a window's event handler manually, use the GetEventHandler function to retrieve the window's event handler and use that to call the member function. By default, GetEventHandler returns a pointer to the window itself unless an application has redirected event handling using SetEventHandler or PushEventHandler.

One use of PushEventHandler is to temporarily or permanently change the behaviour of the GUI. For example, you might want to invoke a dialog editor in your application that changes aspects of dialog boxes. You can grab all the input for an existing dialog box, and edit it 'in situ', before restoring its behaviour to normal. So even if the application has derived new classes to customize behaviour, your utility can indulge in a spot of body-snatching. It could be a useful technique for on-line tutorials, too, where you take a user through a serious of steps and don't want them to diverge from the lesson. Here, you can examine the events coming from buttons and windows, and if acceptable, pass them through to the original event handler. Use PushEventHandler/PopEventHandler to form a chain of event handlers, where each handler processes a different range of events independently from the other handlers.

Window Identifiers

Window identifiers are integers, and are used to uniquely determine window identity in the event system (though you can use it for other purposes). In fact, identifiers do not need to be unique across your entire application as long they are unique within the particular context you're interested in, such as a frame and its children. You may use the `wxID_OK` identifier, for example, on any number of dialogs as long as you don't have several within the same dialog.

If you pass `wxID_ANY` to a window constructor, an identifier will be generated for you automatically by wxWidgets. This is useful when you don't care about the exact identifier either because you're not going to process the events from the control being created or because you process the events from all controls in one place (in which case you should specify `wxID_ANY` in the event table or wxEvtHandler::Bind call as well). The automatically generated identifiers are always negative and so will never conflict with the user-specified identifiers which must be always positive.

See Standard Event Identifiers for the list of standard identifiers available. You can use wxID_HIGHEST to determine the number above which it is safe to define your own identifiers. Or, you can use identifiers below wxID_LOWEST. Finally, you can allocate identifiers dynamically using wxNewId() function too. If you use wxNewId() consistently in your application, you can be sure that your identifiers don't conflict accidentally.

Generic Event Table Macros

EVT_CUSTOM(event, id, func)	Allows you to add a custom event table entry by specifying the event identifier (such as wxEVT_SIZE), the window identifier, and a member function to call.
EVT_CUSTOM_RANGE(event, id1, id2, func)	The same as EVT_CUSTOM, but responds to a range of window identifiers.
EVT_COMMAND(id, event, func)	The same as EVT_CUSTOM, but expects a member function with a wxCommandEvent argument.
EVT_COMMAND_RANGE(id1, id2, event, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxCommandEvent argument.

EVT_NOTIFY(event, id, func)	The same as EVT_CUSTOM, but expects a member function with a wxNotifyEvent argument.
EVT_NOTIFY_RANGE(event, id1, id2, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxNotifyEvent argument.

List of wxWidgets Events

For the full list of event classes, please see the event classes group page.

10.16 Window Sizing Overview

It can sometimes be confusing to keep track of the various size-related attributes of a wxWindow, how they relate to each other, and how they interact with sizers.

This document will attempt to clear the fog a little, and give some simple explanations of things.

10.16.1 Glossary

- **"Size"**: this is the current size of the window and it can be explicitly set or fetched with the wxWindow::SetSize() or wxWindow::GetSize() methods. This size value is the size that the widget is currently using on screen and is the way to change the size of something that is not being managed by a sizer.

- **"Client Size"**: the client size represents the widget's area inside of any borders belonging to the widget and is the area that can be drawn upon in a `EVT_PAINT` event. For wxFrame, the client size also excludes the frame menu, tool and status bars, if any. If a window doesn't have any border (and is not a wxFrame with some bars) then its client size is the same as its size.

- **"Best Size"**: the best size of a widget depends on what kind of widget it is, and usually also on the contents of the widget. For example a wxListBox's best size will be calculated based on how many items it has, up to a certain limit, or a wxButton's best size will be calculated based on its label size, but normally won't be smaller than the platform default button size (unless a style flag overrides that). There is a special virtual method in the C++ window classes called wxWindow::DoGetBestSize() that a class can override if it wants to calculate its own best size based on its content, however notice that usually it is more convenient to override DoGetBestClientSize(), see below.

- **"Best Client Size"**: this is simply the client size corresponding to the best window size. When the fitting size for the given contents is computed, it will usually be the client size and the size of the borders needs to be added to obtain the full best size. For this reason, it's preferable to override DoGetBestClientSize() and let DoGetBestSize() compute the full best size.

- **"Minimal Size"**: the minimal size of a widget is a size that is normally explicitly set by the programmer either with the wxWindow::SetMinSize() method or with the wxWindow::SetSizeHints() method. Most controls will also set the minimal size to the size given in the control's constructor if a non-default value is passed. Top-level windows such as wxFrame will not allow the user to resize the frame below the minimal size.

- **"Maximum Size"**: just like for the minimal size, the maximum size is normally explicitly set by the programmer with the wxWindow::SetMaxSize() method or with wxWindow::SetSizeHints(). Top-level windows such as wxFrame will not allow the user to resize the frame above the maximum size.

- **"Initial Size"**: the initial size of a widget is the size given to the constructor of the widget, if any. As mentioned above most controls will also set this size value as the control's minimal size. If the size passed to the constructor is the default wxDefaultSize, or if the size is not fully specified (such as wxSize(150,-1)) then most controls will fill in the missing size components using the best size and will set the initial size of the control to the resulting size.

- **"Virtual Size"**: the virtual size is the size of the potentially viewable area of the widget. The virtual size of a widget may be larger than its actual size and in this case scrollbars will appear to the let the user 'explore' the full contents of the widget. See wxScrolled for more info.

10.16.2 Functions related to sizing

- wxWindow::GetEffectiveMinSize(): returns a blending of the widget's minimal size and best size, giving precedence to the minimal size. For example, if a widget's min size is set to (150, -1) and the best size is (80, 22) then the best fitting size is (150, 22). If the min size is (50, 20) then the best fitting size is (50, 20). This method is what is called by the sizers when determining what the requirements of each item in the sizer is, and is used for calculating the overall minimum needs of the sizer.

- wxWindow::SetInitialSize(): this is a little different than the typical size setters. Rather than just setting an "initial size" attribute it actually sets the minimal size to the value passed in, blends that value with the best size, and then sets the size of the widget to be the result. So you can consider this method to be a "Smart SetSize". This method is what is called by the constructor of most controls to set the minimal size and the initial size of the control.

- wxWindow::Fit(): this method sets the size of a window to fit around its children. If it has no children then nothing is done, if it does have children then the size of the window is set to the window's best size.

- wxSizer::Fit(): this sets the size of the window to be large enough to accommodate the minimum size needed by the sizer, (along with a few other constraints...). If the sizer is the one that is assigned to the window then this should be equivalent to wxWindow::Fit().

- wxSizer::Layout(): recalculates the minimum space needed by each item in the sizer, and then lays out the items within the space currently allotted to the sizer.

- wxWindow::Layout(): if the window has a sizer then it sets the space given to the sizer to the current size of the window, which results in a call to wxSizer::Layout(). If the window has layout constraints instead of a sizer then the constraints algorithm is run. The `Layout()` method is what is called by the default `EVT_SIZE` handler for container windows.

10.17 Window IDs

Various controls and other parts of wxWidgets need an ID.

Sometimes the ID may be directly provided by the user or have a predefined value, such as `wxID_OPEN`. Often, however, the value of the ID is unimportant and is created automatically by calling wxWindow::NewControlId or by passing `wxID_ANY` as the ID of an object.

There are two ways to generate an ID. One way is to start at a negative number, and for each new ID, return the next smallest number. This is fine for systems that can use the full range of negative numbers for IDs, as this provides more than enough IDs and it would take a very very long time to run out and wrap around. However, some systems cannot use the full range of the ID value. Windows, for example, can only use 16 bit IDs, and only has about 32000 possible automatic IDs that can be generated by wxWindow::NewControlId. If the program runs long enough, depending on the program itself, using this first method would cause the IDs to wrap around into the positive ID range and cause possible clashes with any directly specified ID values.

The other way is to keep track of the IDs returned by wxWindow::NewControlId and don't return them again until the ID is completely free and not being used by any other objects. This will make sure that the ID values do not clash with one another. This is accomplished by keeping a reference count for each of the IDs that can possibly be returned by wxWindow::NewControlId. Other IDs are not reference counted.

See also

wxIdManager, wxWindow::NewControlId(), wxWindow::UnreserveControlId()

10.17.1 Data Types

A wxWindowID is just the integer type for a window ID. It should be used almost everywhere. To help keep track of the count for the automatically generated IDs, a new type, wxWindowIDRef exists, that can take the place of wxWindowID where needed. When an ID is first created, it is marked as reserved. When assigning it to a wxWindowIDRef, the usage count of the ID is increased, or set to 1 if it is currently reserved. Assigning the same ID to several wxWindowIDRefs will keep track of the count. As the wxWindowIDRef gets destroyed or its value changes, it will decrease the count of the used ID. When there are no more wxWindowIDRef types with the created ID, the ID is considered free and can then be used again by wxWindow::NewControlId.

If a created ID is not assigned to a wxWindowIDRef, then it remains reserved until it is unreserved manually with wxWindow::UnreserveControlId. However, if it is assigned to a wxWindowIDRef, then it will be unreserved automatically and will be considered free when the count is 0, and should NOT be manually unreserved.

wxWindowIDRef can store both automatic IDs from wxWindow::NewControlId and normal IDs. Reference counting is only done for the automatic IDs. Also, wxWindowIDRef has conversion operators that allow it to be treated just like a wxWindowID.

10.17.2 Using wxWindowIDRef

A wxWindowIDRef should be used in place of a wxWindowID where you want to make sure the ID is not created again by wxWindow::NewControlId at least until the wxWindowIDRef is destroyed, usually when the associated object is destroyed. This is done already for windows, menu items, and tool bar items. It should only be used in the main thread, as it is not thread safe.

10.18 Logging Overview

This is a general overview of logging classes provided by wxWidgets.

The word logging here has a broad sense, including all of the program output, not only non-interactive messages. The logging facilities included in wxWidgets provide the base wxLog class which defines the standard interface for a *log* target as well as several standard implementations of it and a family of functions to use with them.

First of all, no knowledge of wxLog classes is needed to use them. For this, you should only know about wxLogXXX() functions. All of them have the same syntax as *printf()* or *vprintf()* , i.e. they take the format string as the first argument and respectively a variable number of arguments or a variable argument list pointer. Here are all of them:

- wxLogFatalError() which is like wxLogError(), but also terminates the program with the exit code 3 (using *abort()* standard function). Unlike for all the other logging functions, this function can't be overridden by a log target.
- wxLogError() is the function to use for error messages, i.e. the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.
- wxLogWarning() for warnings. They are also normally shown to the user, but don't interrupt the program work.
- wxLogMessage() is for all normal, informational messages. They also appear in a message box by default (but it can be changed, see below).
- wxLogVerbose() is for verbose output. Normally, it is suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function is wxLogInfo).
- wxLogStatus() is for status messages. They will go into the status bar of the active or specified (as the first argument) wxFrame if it has one.
- wxLogSysError() is mostly used by wxWidgets itself, but might be handy for logging errors after system call (API function) failure. It logs the specified message text as well as the last system error code (*errno* or Windows' *GetLastError()* depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.

- wxLogDebug() is **the** right function for debug output. It only does anything at all in the debug mode (when the preprocessor symbol **WXDEBUG** is defined) and expands to nothing in release mode (otherwise). Note that under Windows, you must either run the program under debugger or use a 3rd party program such as DebugView (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx) to actually see the debug output.
- wxLogTrace() as wxLogDebug() only does something in debug build. The reason for making it a separate function from it is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages which would be flooded in them. Moreover, the second version of this function takes a trace mask as the first argument which allows to further restrict the amount of messages generated.

See also

Logging Functions and Macros

The usage of these functions should be fairly straightforward, however it may be asked why not use the other logging facilities, such as C standard stdio functions or C++ streams. The short answer is that they're all very good generic mechanisms, but are not really adapted for wxWidgets, while the log classes are. Some of advantages in using wxWidgets log functions are:

- **Portability:** It is a common practice to use *printf()* statements or cout/cerr C++ streams for writing out some (debug or otherwise) information. Although it works just fine under Unix, these messages go strictly nowhere under Windows where the stdout of GUI programs is not assigned to anything. Thus, you might view wxLogMessage() as a simple substitute for *printf()*. You can also redirect the *wxLogXXX* calls to *cout* by just writing:

```
wxLog* logger = new wxLogStream(&cout);
wxLog::SetActiveTarget(logger);
```

  Finally, there is also a possibility to redirect the output sent to *cout* to a wxTextCtrl by using the wxStreamToTextRedirector class.
- **Flexibility:** The output of wxLog functions can be redirected or suppressed entirely based on their importance, which is either impossible or difficult to do with traditional methods. For example, only error messages, or only error messages and warnings might be logged, filtering out all informational messages.
- **Completeness:** Usually, an error message should be presented to the user when some operation fails. Let's take a quite simple but common case of a file error: suppose that you're writing your data file on disk and there is not enough space. The actual error might have been detected inside wxWidgets code (say, in wxFile::Write), so the calling function doesn't really know the exact reason of the failure, it only knows that the data file couldn't be written to the disk. However, as wxWidgets uses wxLogError() in this situation, the exact error code (and the corresponding error message) will be given to the user together with "high level" message about data file writing error.

10.18.1 Log Messages Selection

By default, most log messages are enabled. In particular, this means that errors logged by wxWidgets code itself (e.g. when it fails to perform some operation, for instance wxFile::Open() logs an error when it fails to open a file) will be processed and shown to the user. To disable the logging entirely you can use wxLog::EnableLogging() method or, more usually, wxLogNull class which temporarily disables logging and restores it back to the original setting when it is destroyed.

To limit logging to important messages only, you may use wxLog::SetLogLevel() with e.g. wxLOG_Warning value – this will completely disable all logging messages with the severity less than warnings, so wxLogMessage() output won't be shown to the user any more.

Moreover, the log level can be set separately for different log components. Before showing how this can be useful, let us explain what log components are: they are simply arbitrary strings identifying the component, or module, which generated the message. They are hierarchical in the sense that "foo/bar/baz" component is supposed to be a child of "foo". And all components are children of the unnamed root component.

By default, all messages logged by wxWidgets originate from "wx" component or one of its subcomponents such as "wx/net/ftp", while the messages logged by your own code are assigned empty log component. To change this, you need to define `wxLOG_COMPONENT` to a string uniquely identifying each component, e.g. you could give it the value "MyProgram" by default and re-define it as "MyProgram/DB" in the module working with the database and "MyProgram/DB/Trans" in its part managing the transactions. Then you could use wxLog::SetComponentLevel() in the following ways:

```
// disable all database error messages, everybody knows databases never
// fail anyhow
wxLog::SetComponentLevel("MyProgram/DB",
      wxLOG_FatalError);

// but enable tracing for the transactions as somehow our changes don't
// get committed sometimes
wxLog::SetComponentLevel("MyProgram/DB/Trans",
      wxLOG_Trace);

// also enable tracing messages from wxWidgets dynamic module loading
// mechanism
wxLog::SetComponentLevel("wx/base/module", wxLOG_Trace);
```

Notice that the log level set explicitly for the transactions code overrides the log level of the parent component but that all other database code subcomponents inherit its setting by default and so won't generate any log messages at all.

10.18.2 Log Targets

After having enumerated all the functions which are normally used to log the messages, and why would you want to use them, we now describe how all this works.

wxWidgets has the notion of a *log target*: it is just a class deriving from wxLog. As such, it implements the virtual functions of the base class which are called when a message is logged. Only one log target is *active* at any moment, this is the one used by wxLogXXX() functions. The normal usage of a log object (i.e. object of a class derived from wxLog) is to install it as the active target with a call to *SetActiveTarget()* and it will be used automatically by all subsequent calls to wxLogXXX() functions.

To create a new log target class you only need to derive it from wxLog and override one or several of wxLog::DoLogRecord(), wxLog::DoLogTextAtLevel() and wxLog::DoLogText() in it. The first one is the most flexible and allows you to change the formatting of the messages, dynamically filter and redirect them and so on – all log messages, except for those generated by wxLogFatalError(), pass by this function. wxLog::DoLogTextAtLevel() should be overridden if you simply want to redirect the log messages somewhere else, without changing their formatting. Finally, it is enough to override wxLog::DoLogText() if you only want to redirect the log messages and the destination doesn't depend on the message log level.

There are some predefined classes deriving from wxLog and which might be helpful to see how you can create a new log target class and, of course, may also be used without any change. There are:

- wxLogStderr: This class logs messages to a `FILE *`, using stderr by default as its name suggests.
- wxLogStream: This class has the same functionality as wxLogStderr, but uses *ostream* and cerr instead of `FILE *` and stderr.
- wxLogGui: This is the standard log target for wxWidgets applications (it is used by default if you don't do anything) and provides the most reasonable handling of all types of messages for given platform.
- wxLogWindow: This log target provides a "log console" which collects all messages generated by the application and also passes them to the previous active log target. The log window frame has a menu allowing user to clear the log, close it completely or save all messages to file.
- wxLogBuffer: This target collects all the logged messages in an internal buffer allowing to show them later to the user all at once.
- wxLogNull: The last log class is quite particular: it doesn't do anything. The objects of this class may be instantiated to (temporarily) suppress output of *wxLogXXX()* functions. As an example, trying to open a non-existing file will usually provoke an error message, but if for some reasons it is unwanted, just use this construction:

```
wxFile file;

// wxFile.Open() normally complains if file can't be opened, we don't want it
{
    wxLogNull logNo;
    if ( !file.Open("bar") )
    {
        // ... process error ourselves ...
    }
} // ~wxLogNull called, old log sink restored

wxLogMessage("..."); // ok
```

See also

Logging Classes

The log targets can also be combined: for example you may wish to redirect the messages somewhere else (for example, to a log file) but also process them as normally. For this the wxLogChain, wxLogInterposer, and wxLogInterposerTemp can be used.

10.18.3 Logging in Multi-Threaded Applications

Starting with wxWidgets 2.9.1, logging functions can be safely called from any thread. Messages logged from threads other than the main one will be buffered until wxLog::Flush() is called in the main thread (which usually happens during idle time, i.e. after processing all pending events) and will be really output only then. Notice that the default GUI logger already only output the messages when it is flushed, so by default messages from the other threads will be shown more or less at the same moment as usual. However if you define a custom log target, messages may be logged out of order, e.g. messages from the main thread with later timestamp may appear before messages with earlier timestamp logged from other threads. wxLog does however guarantee that messages logged by each thread will appear in order in which they were logged.

Also notice that wxLog::EnableLogging() and wxLogNull class which uses it only affect the current thread, i.e. logging messages may still be generated by the other threads after a call to `EnableLogging(false)`.

10.18.4 Logging Customization

To completely change the logging behaviour you may define a custom log target. For example, you could define a class inheriting from wxLog which shows all the log messages in some part of your main application window reserved for the message output without interrupting the user work flow with modal message boxes.

To use your custom log target you may either call wxLog::SetActiveTarget() with your custom log object or create a wxAppTraits-derived class and override wxAppTraits::CreateLogTarget() virtual method in it and also override wxApp::CreateTraits() to return an instance of your custom traits object. Notice that in the latter case you should be prepared for logging messages early during the program startup and also during program shutdown so you shouldn't rely on existence of the main application window, for example. You can however safely assume that GUI is (already/still) available when your log target as used as wxWidgets automatically switches to using wxLogStderr if it isn't.

There are several methods which may be overridden in the derived class to customize log messages handling: wxLog::DoLogRecord(), wxLog::DoLogTextAtLevel() and wxLog::DoLogText().

The last method is the simplest one: you should override it if you simply want to redirect the log output elsewhere, without taking into account the level of the message. If you do want to handle messages of different levels differently, then you should override wxLog::DoLogTextAtLevel().

Additionally, you can customize the way full log messages are constructed from the components (such as time stamp, source file information, logging thread ID and so on). This task is performed by wxLogFormatter class so you need to derive a custom class from it and override its Format() method to build the log messages in desired way. Notice that if you just need to modify (or suppress) the time stamp display, overriding FormatTime() is enough.

Finally, if even more control over the output format is needed, then DoLogRecord() can be overridden as it allows to construct custom messages depending on the log level or even do completely different things depending on the

message severity (for example, throw away all messages except warnings and errors, show warnings on the screen and forward the error messages to the user's (or programmer's) cell phone – maybe depending on whether the timestamp tells us if it is day or night in the current time zone).

The *dialog* sample illustrates this approach by defining a custom log target customizing the dialog used by wxLogGui for the single messages.

10.18.5 Using Trace Masks

Notice that the use of log trace masks is hardly necessary any longer in current wxWidgets version as the same effect can be achieved by using different log components for different log statements of any level. Please see Log Messages Selection for more information about the log components.

The functions below allow some limited customization of wxLog behaviour without writing a new log target class (which, aside from being a matter of several minutes, allows you to do anything you want). The verbose messages are the trace messages which are not disabled in the release mode and are generated by wxLogVerbose(). They are not normally shown to the user because they present little interest, but may be activated, for example, in order to help the user find some program problem.

As for the (real) trace messages, their handling depends on the currently enabled trace masks: if wxLog::AddTraceMask() was called for the mask of the given message, it will be logged, otherwise nothing happens.

For example,

```
wxLogTrace( wxTRACE_OleCalls, "IFoo::Bar() called" );
```

will log the message if it was preceded by:

```
wxLog::AddTraceMask( wxTRACE_OleCalls );
```

The standard trace masks are given in wxLogTrace() documentation.

10.19 wxString Overview

wxString is a class which represents a Unicode string of arbitrary length and containing arbitrary Unicode characters.

This class has all the standard operations you can expect to find in a string class: dynamic memory management (string extends to accommodate new characters), construction from other strings, compatibility with C strings and wide character C strings, assignment operators, access to individual characters, string concatenation and comparison, substring extraction, case conversion, trimming and padding (with spaces), searching and replacing and both C-like `printf` (wxString::Printf) and stream-like insertion functions as well as much more - see wxString for a list of all functions.

The wxString class has been completely rewritten for wxWidgets 3.0 but much work has been done to make existing code using ANSI string literals work as it did in previous versions.

10.19.1 Internal wxString Encoding

Since wxWidgets 3.0 wxString may use any of `UTF-16` (under Windows, using the native 16 bit `wchar_t`), `UTF-32` (under Unix, using the native 32 bit `wchar_t`) or `UTF-8` (under both Windows and Unix) to store its content. By default, `wchar_t` is used under all platforms, but wxWidgets can be compiled with `wxUSE_UNICODE_UTF8=1` to use UTF-8.

For simplicity of implementation, wxString uses *per code unit indexing* instead of *per code point indexing* when using UTF-16, i.e. in the default `wxUSE_UNICODE_WCHAR==1` build under Windows and doesn't know anything about surrogate pairs. In other words it always considers code points to be composed by 1 code unit, while this is really true only for characters in the *BMP* (Basic Multilingual Plane), as explained in more details in the Unicode Representations and Terminology section. Thus when iterating over a UTF-16 string stored in a wxString under

Windows, the user code has to take care of *surrogate pairs* himself. (Note however that Windows itself has built-in support for surrogate pairs in UTF-16, such as for drawing strings on screen.)

Remarks

Note that while the behaviour of wxString when `wxUSE_UNICODE_WCHAR==1` resembles UCS-2 encoding, it's not completely correct to refer to wxString as UCS-2 encoded since you can encode code points outside the *BMP* in a wxString as two code units (i.e. as a surrogate pair; as already mentioned however wxString will "see" them as two different code points)

In `wxUSE_UNICODE_UTF8==1` case, wxString handles UTF-8 multi-bytes sequences just fine also for characters outside the BMP (it implements *per code point indexing*), so that you can use UTF-8 in a completely transparent way:

Example:

```
// first test, using exotic characters outside of the Unicode BMP:

wxString test = wxString::FromUTF8("\xF0\x90\x8C\x80");
    // U+10300 is "OLD ITALIC LETTER A" and is part of Unicode Plane 1
    // in UTF8 it's encoded as 0xF0 0x90 0x8C 0x80

// it's a single Unicode code-point encoded as:
// - a UTF16 surrogate pair under Windows
// - a UTF8 multiple-bytes sequence under Linux
// (without considering the final NULL)

wxPrintf("wxString reports a length of %d character(s)", test.length());
    // prints "wxString reports a length of 1 character(s)" on Linux
    // prints "wxString reports a length of 2 character(s)" on Windows
    // since wxString on Windows doesn't have surrogate pairs support!


// second test, this time using characters part of the Unicode BMP:

wxString test2 = wxString::FromUTF8("\x41\xC3\xA0\xE2\x82\xAC");
    // this is the UTF8 encoding of capital letter A followed by
    // 'small case letter a with grave' followed by the 'euro sign'

// they are 3 Unicode code-points encoded as:
// - 3 UTF16 code units under Windows
// - 6 UTF8 code units under Linux
// (without considering the final NULL)

wxPrintf("wxString reports a length of %d character(s)", test2.length());
    // prints "wxString reports a length of 3 character(s)" on Linux
    // prints "wxString reports a length of 3 character(s)" on Windows
```

To better explain what stated above, consider the second string of the example above; it's composed by 3 characters and the final `NULL`:

As you can see, UTF16 encoding is straightforward (for characters in the *BMP*) and in this example the UTF16-encoded wxString takes 8 bytes. UTF8 encoding is more elaborated and in this example takes 7 bytes.

In general, for strings containing many latin characters UTF8 provides a big advantage with regards to the memory footprint respect UTF16, but requires some more processing for common operations like e.g. length calculation.

Finally, note that the type used by wxString to store Unicode code units (`wchar_t` or `char`) is always `typedef-ined` to be wxStringCharType.

10.19.2 Using wxString to store binary data

wxString can be used to store binary data (even if it contains `NULs`) using the functions wxString::To8BitData and wxString::From8BitData.

Beware that even if `NUL` character is allowed, in the current string implementation some methods might not work correctly with them.

Note however that other classes like wxMemoryBuffer are more suited to this task. For handling binary data you may also want to look at the wxStreamBuffer, wxMemoryOutputStream, wxMemoryInputStream classes.

10.19.3 Comparison to Other String Classes

The advantages of using a special string class instead of working directly with C strings are so obvious that there is a huge number of such classes available. The most important advantage is the need to always remember to allocate/free memory for C strings; working with fixed size buffers almost inevitably leads to buffer overflows. At last, C++ has a standard string class (`std::string`). So why the need for wxString? There are several advantages:

- **Efficiency:** Since wxWidgets 3.0 wxString uses `std::string` (in UTF8 mode under Linux, Unix and OS X) or `std::wstring` (in UTF16 mode under Windows) internally by default to store its contents. wxString will therefore inherit the performance characteristics from `std::string`.
- **Compatibility:** This class tries to combine almost full compatibility with the old wxWidgets 1.xx wxString class, some reminiscence of MFC's CString class and 90% of the functionality of `std::string` class.
- **Rich set of functions:** Some of the functions present in wxString are very useful but don't exist in most of other string classes: for example, wxString::AfterFirst, wxString::BeforeLast, wxString::Printf. Of course, all the standard string operations are supported as well.
- **wxString is Unicode friendly:** it allows to easily convert to and from ANSI and Unicode strings (see Unicode Support in wxWidgets for more details) and maps to `std::wstring` transparently.
- **Used by wxWidgets:** And, of course, this class is used everywhere inside wxWidgets so there is no performance loss which would result from conversions of objects of any other string class (including `std::string`) to wxString internally by wxWidgets.

However, there are several problems as well. The most important one is probably that there are often several functions to do exactly the same thing: for example, to get the length of the string either one of wxString::length(), wxString::Len() or wxString::Length() may be used. The first function, as almost all the other functions in lowercase, is `std::string` compatible. The second one is the "native" wxString version and the last one is the wxWidgets 1.xx way.

So which is better to use? The usage of the `std::string` compatible functions is strongly advised! It will both make your code more familiar to other C++ programmers (who are supposed to have knowledge of `std::string` but not of wxString), let you reuse the same code in both wxWidgets and other programs (by just typedefing wxString as `std::string` when used outside wxWidgets) and by staying compatible with future versions of wxWidgets which will probably start using `std::string` sooner or later too.

In the situations where there is no corresponding `std::string` function, please try to use the new wxString methods and not the old wxWidgets 1.xx variants which are deprecated and may disappear in future versions.

10.19.4 Advice About Using wxString

Implicit conversions

Probably the main trap with using this class is the implicit conversion operator to `const char*`. It is advised that you use wxString::c_str() instead to clearly indicate when the conversion is done. Specifically, the danger of this implicit conversion may be seen in the following code fragment:

```
// this function converts the input string to uppercase,
// output it to the screen and returns the result
const char *SayHELLO(const wxString& input)
{
    wxString output = input.Upper();
    printf("Hello, %s!\n", output);
    return output;
}
```

There are two nasty bugs in these three lines. The first is in the call to the `printf()` function. Although the implicit conversion to C strings is applied automatically by the compiler in the case of

```
puts(output);
```

because the argument of `puts()` is known to be of the type `const char*`, this is **not** done for `printf()` which is a function with variable number of arguments (and whose arguments are of unknown types). So this call may do any number of things (including displaying the correct string on screen), although the most likely result is a program crash. The solution is to use wxString::c_str(). Just replace this line with this:

```
printf("Hello, %s!\n", output.c_str());
```

The second bug is that returning `output` doesn't work. The implicit cast is used again, so the code compiles, but as it returns a pointer to a buffer belonging to a local variable which is deleted as soon as the function exits, its contents are completely arbitrary. The solution to this problem is also easy, just make the function return wxString instead of a C string.

This leads us to the following general advice: all functions taking string arguments should take `const wx↩String&` (this makes assignment to the strings inside the function faster) and all functions returning strings should return wxString - this makes it safe to return local variables.

Finally note that wxString uses the current locale encoding to convert any C string literal to Unicode. The same is done for converting to and from `std::string` and for the return value of c_str(). For this conversion, the *wxConvLibc* class instance is used. See wxCSConv and wxMBConv.

Iterating wxString Characters

As previously described, when `wxUSE_UNICODE_UTF8==1`, wxString internally uses the variable-length UTF8 encoding. Accessing a UTF-8 string by index can be very **inefficient** because a single character is represented by a variable number of bytes so that the entire string has to be parsed in order to find the character. Since iterating over a string by index is a common programming technique and was also possible and encouraged by wxString using the access operator[]() wxString implements caching of the last used index so that iterating over a string is a linear operation even in UTF-8 mode.

It is nonetheless recommended to use **iterators** (instead of index based access) like this:

```
wxString s = "hello";
wxString::const_iterator i;
for (i = s.begin(); i != s.end(); ++i)
{
    wxUniChar uni_ch = *i;
    // do something with it
}
```

10.19.5 String Related Functions and Classes

As most programs use character strings, the standard C library provides quite a few functions to work with them. Unfortunately, some of them have rather counter-intuitive behaviour (like `strncpy()` which doesn't always terminate the resulting string with a NULL) and are in general not very safe (passing NULL to them will probably lead to program crash). Moreover, some very useful functions are not standard at all. This is why in addition to all wxString functions, there are also a few global string functions which try to correct these problems: wxIsEmpty() verifies whether the string is empty (returning true for NULL pointers), wxStrlen() also handles NULL correctly and returns 0 for them and wxStricmp() is just a platform-independent version of case-insensitive string comparison function known either as `stricmp()` or `strcasecmp()` on different platforms.

The `<wx/string.h>` header also defines wxSnprintf() and wxVsnprintf() functions which should be used instead of the inherently dangerous standard `sprintf()` and which use `snprintf()` instead which does buffer size checks whenever possible. Of course, you may also use wxString::Printf which is also safe.

There is another class which might be useful when working with wxString: wxStringTokenizer. It is helpful when a string must be broken into tokens and replaces the standard C library `strtok()` function.

And the very last string-related class is wxArrayString: it is just a version of the "template" dynamic array class which is specialized to work with strings. Please note that this class is specially optimized (using its knowledge of the internal structure of wxString) for storing strings and so it is vastly better from a performance point of view than a wxObjectArray of wxStrings.

10.19.6 Tuning wxString for Your Application

Note

This section is strictly about performance issues and is absolutely not necessary to read for using wxString class. Please skip it unless you feel familiar with profilers and relative tools.

For the performance reasons wxString doesn't allocate exactly the amount of memory needed for each string. Instead, it adds a small amount of space to each allocated block which allows it to not reallocate memory (a relatively expensive operation) too often as when, for example, a string is constructed by subsequently adding one character at a time to it, as for example in:

```
// delete all vowels from the string
wxString DeleteAllVowels(const wxString& original)
{
    wxString vowels( "aeuioAEIOU" );
    wxString result;
    wxString::const_iterator i;
    for ( i = original.begin(); i != original.end(); ++i )
    {
        if (vowels.Find( *i ) == wxNOT_FOUND)
            result += *i;
    }

    return result;
}
```

This is quite a common situation and not allocating extra memory at all would lead to very bad performance in this case because there would be as many memory (re)allocations as there are consonants in the original string. Allocating too much extra memory would help to improve the speed in this situation, but due to a great number of wxString objects typically used in a program would also increase the memory consumption too much.

The very best solution in precisely this case would be to use wxString::Alloc() function to preallocate, for example, len bytes from the beginning - this will lead to exactly one memory allocation being performed (because the result is at most as long as the original string).

However, using wxString::Alloc() is tedious and so wxString tries to do its best. The default algorithm assumes that memory allocation is done in granularity of at least 16 bytes (which is the case on almost all of wide-spread platforms) and so nothing is lost if the amount of memory to allocate is rounded up to the next multiple of 16. Like this, no memory is lost and 15 iterations from 16 in the example above won't allocate memory but use the already allocated pool.

The default approach is quite conservative. Allocating more memory may bring important performance benefits for programs using (relatively) few very long strings. The amount of memory allocated is configured by the setting of `EXTRA_ALLOC` in the file string.cpp during compilation (be sure to understand why its default value is what it is before modifying it!). You may try setting it to greater amount (say twice nLen) or to 0 (to see performance degradation which will follow) and analyse the impact of it on your program. If you do it, you will probably find it helpful to also define `WXSTRING_STATISTICS` symbol which tells the wxString class to collect performance statistics and to show them on stderr on program termination. This will show you the average length of strings your program manipulates, their average initial length and also the percent of times when memory wasn't reallocated when string concatenation was done but the already preallocated memory was used (this value should be about 98% for the default allocation policy, if it is less than 90% you should really consider fine tuning wxString for your application).

It goes without saying that a profiler should be used to measure the precise difference the change to `EXTRA_AL↩LOC` makes to your program.

10.19.7 wxString Related Compilation Settings

The main option affecting wxString is `wxUSE_UNICODE` which is now always defined as `1` by default to indicate Unicode support. You may set it to 0 to disable Unicode support in wxString and elsewhere in wxWidgets but this is *strongly* not recommended.

Another option affecting wxWidgets is `wxUSE_UNICODE_WCHAR` which is also 1 by default. You may want to set it to 0 and set `wxUSE_UNICODE_UTF8` to 1 instead to use UTF-8 internally. wxString still provides the same API in

this case, but using UTF-8 has performance implications as explained in Performance Implications of Using UTF-8, so it probably shouldn't be enabled for legacy code which might contain a lot of index-using loops.

See also Most Important Symbols for a few other options affecting wxString.

10.20 Buffer Classes

wxWidgets uses two classes of classes for dealing with buffers in memory.

The first is one for dealing with character buffers, namely wxCharBuffer for char pointer or multi-byte c strings and wxWCharBuffer for wchar_t pointer or wide character c strings.

Secondly, wxWidgets uses, although only rarely currently, wxMemoryBuffer for dealing with raw buffers in memory.

10.20.1 wxXCharBuffer

General Usage

As mentioned, wxCharBuffer and its wide character variant wxWCharBuffer deal with c strings in memory. They have two constructors, one in which you pass the c string you want them to have a copy of, and another where you specify the size of the buffer in memory in characters you want.

wxCharBuffer and its variant only contain the c string as a member, so they can be used safely to c functions with variable arguments such as printf. They also contain standard assignment, character access operators and a copy constructor.

Destruction

It should be noted that on destruction wxCharBuffer and its wide character variant delete the c string that hold onto. If you want to get the pointer to the buffer and don't want wxCharBuffer to delete it on destruction, use the member function release to do so.

10.21 Date and Time

wxWidgets provides a set of powerful classes to work with dates and times.

Some of the supported features of wxDateTime class are:

- Wide range: the range of supported dates goes from about 4714 B.C. to some 480 million years in the future.
- Precision: not using floating point calculations anywhere ensures that the date calculations don't suffer from rounding errors.
- Many features: not only all usual calculations with dates are supported, but also more exotic week and year day calculations, work day testing, standard astronomical functions, conversion to and from strings in either strict or free format.
- Efficiency: objects of wxDateTime are small (8 bytes) and working with them is fast

There are 3 main classes declared in `wx/datetime.h`: except wxDateTime itself which represents an absolute moment in time, there are also two classes - wxTimeSpan and wxDateSpan - which represent the intervals of time.

There are also helper classes which are used together with wxDateTime: wxDateTimeHolidayAuthority which is used to determine whether a given date is a holiday or not and wxDateTimeWorkDays which is a derivation of this class for which (only) Saturdays and Sundays are the holidays. See more about these classes in the discussion of the holidays (see wxDateTime and Holidays).

Finally, in other parts of this manual you may find mentions of wxDate and wxTime classes. Compatibility are obsolete and superseded by wxDateTime.

10.21.1 wxDateTime Characteristics

wxDateTime stores the time as a signed number of milliseconds since the Epoch which is fixed, by convention, to Jan 1, 1970 - however this is not visible to the class users (in particular, dates prior to the Epoch are handled just as well (or as bad) as the dates after it). But it does mean that the best resolution which can be achieved with this class is 1 millisecond.

The size of wxDateTime object is 8 bytes because it is represented as a 64 bit integer. The resulting range of supported dates is thus approximatively 580 million years, but due to the current limitations in the Gregorian calendar support, only dates from Nov 24, 4714BC are supported (this is subject to change if there is sufficient interest in doing it).

Finally, the internal representation is time zone independent (always in GMT) and the time zones only come into play when a date is broken into year/month/day components. See more about timezones below (see Time Zone Considerations).

Currently, the only supported calendar is Gregorian one (which is used even for the dates prior to the historic introduction of this calendar which was first done on Oct 15, 1582 but is, generally speaking, country, and even region, dependent). Future versions will probably have Julian calendar support as well and support for other calendars (Maya, Hebrew, Chinese...) is not ruled out.

10.21.2 wxDateSpan and wxTimeSpan

While there is only one logical way to represent an absolute moment in the time (and hence only one wxDateTime class), there are at least two methods to describe a time interval.

First, there is the direct and self-explaining way implemented by wxTimeSpan: it is just a difference in milliseconds between two moments in time. Adding or subtracting such an interval to wxDateTime is always well-defined and is a fast operation.

But in the daily life other, calendar-dependent time interval specifications are used. For example, 'one month later' is commonly used. However, it is clear that this is not the same as wxTimeSpan of $60*60*24*31$ seconds because 'one month later' Feb 15 is Mar 15 and not Mar 17 or Mar 16 (depending on whether the year is leap or not).

This is why there is another class for representing such intervals called wxDateSpan. It handles these sort of operations in the most natural way possible, but note that manipulating with intervals of this kind is not always well-defined. Consider, for example, Jan 31 + '1 month': this will give Feb 28 (or 29), i.e. the last day of February and not the non-existent Feb 31. Of course, this is what is usually wanted, but you still might be surprised to notice that now subtracting back the same interval from Feb 28 will result in Jan 28 and **not** Jan 31 we started with!

So, unless you plan to implement some kind of natural language parsing in the program, you should probably use wxTimeSpan instead of wxDateSpan (which is also more efficient). However, wxDateSpan may be very useful in situations when you do need to understand what 'in a month' means (of course, it is just `wxDateTime::Now()` + wxDateSpan::Month()).

10.21.3 Date Arithmetics

Many different operations may be performed with the dates, however not all of them make sense. For example, multiplying a date by a number is an invalid operation, even though multiplying either of the time span classes by a number is perfectly valid.

Here is what can be done:

- **Addition:** a wxTimeSpan or wxDateSpan can be added to wxDateTime resulting in a new wxDateTime object and also 2 objects of the same span class can be added together giving another object of the same class.
- **Subtraction:** the same types of operations as above are allowed and, additionally, a difference between two wxDateTime objects can be taken and this will yield wxTimeSpan.
- **Multiplication:** a wxTimeSpan or wxDateSpan object can be multiplied by an integer number resulting in an object of the same type.

- **Unary minus**: a wxTimeSpan or wxDateSpan object may finally be negated giving an interval of the same magnitude but of opposite time direction.

For all these operations there are corresponding global (overloaded) operators and also member functions which are synonyms for them: Add(), Subtract() and Multiply(). Unary minus as well as composite assignment operations (like +=) are only implemented as members and Neg() is the synonym for unary minus.

10.21.4 Time Zone Considerations

Although the time is always stored internally in GMT, you will usually work in the local time zone. Because of this, all wxDateTime constructors and setters which take the broken down date assume that these values are for the local time zone. Thus, `wxDateTime(1, wxDateTime::Jan, 1970)` will not correspond to the wxDateTime Epoch unless you happen to live in the UK. All methods returning the date components (year, month, day, hour, minute, second...) will also return the correct values for the local time zone by default, so, generally, doing the natural things will lead to natural and correct results.

If you only want to do this, you may safely skip the rest of this section. However, if you want to work with different time zones, you should read it to the end.

In this (rare) case, you are still limited to the local time zone when constructing wxDateTime objects, i.e. there is no way to construct a wxDateTime corresponding to the given date in, say, Pacific Standard Time. To do it, you will need to call wxDateTime::ToTimezone or wxDateTime::MakeTimezone methods to adjust the date for the target time zone. There are also special versions of these functions wxDateTime::ToUTC and wxDateTime::MakeUTC for the most common case - when the date should be constructed in UTC.

You also can just retrieve the value for some time zone without converting the object to it first. For this you may pass TimeZone argument to any of the methods which are affected by the time zone (all methods getting date components and the date formatting ones, for example). In particular, the Format() family of methods accepts a TimeZone parameter and this allows to simply print time in any time zone.

To see how to do it, the last issue to address is how to construct a TimeZone object which must be passed to all these methods. First of all, you may construct it manually by specifying the time zone offset in seconds from GMT, but usually you will just use one of the Date and Time and let the conversion constructor do the job.

I.e. you would just write

```
wxDateTime dt(...whatever...);
printf("The time is %s in local time zone", dt.FormatTime().c_str());
printf("The time is %s in GMT", dt.FormatTime(wxDateTime::GMT).c_str());
```

10.21.5 Daylight Saving Time (DST)

DST (a.k.a. 'summer time') handling is always a delicate task which is better left to the operating system which is supposed to be configured by the administrator to behave correctly. Unfortunately, when doing calculations with date outside of the range supported by the standard library, we are forced to deal with these issues ourselves.

Several functions are provided to calculate the beginning and end of DST in the given year and to determine whether it is in effect at the given moment or not, but they should not be considered as absolutely correct because, first of all, they only work more or less correctly for only a handful of countries (any information about other ones appreciated!) and even for them the rules may perfectly well change in the future.

The time zone handling methods (see Time Zone Considerations) use these functions too, so they are subject to the same limitations.

10.21.6 wxDateTime and Holidays

Todo WRITE THIS DOC PARAGRAPH.

10.21.7 Compatibility

The old classes for date/time manipulations ported from wxWidgets version 1.xx are still included but are reimplemented in terms of wxDateTime. However, using them is strongly discouraged because they have a few quirks/bugs and were not 'Y2K' compatible.

10.22 Container Classes

For historical reasons, wxWidgets uses custom container classes internally.

This was unfortunately unavoidable during a long time when the standard library wasn't widely available and can't be easily changed even now that it is for compatibility reasons. If you are building your own version of the library and don't care about compatibility nor slight (less than 5%) size penalty imposed by the use of STL classes, you may choose to use the "STL" build of wxWidgets in which these custom classes are replaced with their standard counterparts and only read the section STL Build explaining how to do it.

Otherwise you will need to know about the custom wxWidgets container classes such as wxList<T> and wxArray<T> if only to use wxWidgets functions that work with them, e.g. wxWindow::GetChildren(), and you should find the information about using these classes below useful.

Notice that we recommend that you use standard classes directly in your own code instead of the container classes provided by wxWidgets in any case as the standard classes are easier to use and may also be safer because of extra run-time checks they may perform as well as more efficient.

Finally notice that recent versions of wxWidgets also provide standard-like classes such as wxVector<T>, wxStack<T> or wxDList which can be used exactly like the std::vector<T>, std::stack<T> and std::list<T*>, respectively, and actually are just typedefs for the corresponding types if wxWidgets is compiled in STL mode. These classes could be useful if you wish to avoid the use of the standard library in your code for some reason.

To summarize, you should use the standard container classes such as std::vector<T> and std::list<T> if possible and wxVector<T> or wxDList<T> if it isn't and only use legacy wxWidgets containers such as wxArray<T> and wxList<T> when you must, i.e. when you use a wxWidgets function taking or returning a container of such type.

See also

Containers

10.22.1 Legacy Classes

The list classes in wxWidgets are doubly-linked lists which may either own the objects they contain (meaning that the list deletes the object when it is removed from the list or the list itself is destroyed) or just store the pointers depending on whether or not you called wxList<T>::DeleteContents() method.

Dynamic arrays resemble C arrays but with two important differences: they provide run-time range checking in debug builds and they automatically expand the allocated memory when there is no more space for new items. They come in two sorts: the "plain" arrays which store either built-in types such as "char", "int" or "bool" or the pointers to arbitrary objects, or "object arrays" which own the object pointers to which they store.

For the same portability reasons, the container classes implementation in wxWidgets don't use templates, but are rather based on C preprocessor i.e. are implemented using the macros: WX_DECLARE_LIST() and WX_DEFINE_LIST() for the linked lists and WX_DECLARE_ARRAY(), WX_DECLARE_OBJARRAY() and WX_DEFINE_OBJARRAY() for the dynamic arrays.

The "DECLARE" macro declares a new container class containing the elements of given type and is needed for all three types of container classes: lists, arrays and objarrays. The "DEFINE" classes must be inserted in your program in a place where the *full* declaration of container element class is in scope (i.e. not just forward declaration), otherwise destructors of the container elements will not be called!

As array classes never delete the items they contain anyhow, there is no WX_DEFINE_ARRAY() macro for them.

Examples of usage of these macros may be found in wxList<T> and wxArray<T> documentation.

Finally, wxWidgets predefines several commonly used container classes. wxList is defined for compatibility with previous versions as a list containing wxObjects and wxStringList as a list of C-style strings (char *), both of these classes are deprecated and should not be used in new programs. The following array classes are defined: wxArrayInt, wxArrayLong, wxArrayPtrVoid and wxArrayString. The first three store elements of corresponding types, but wxArrayString is somewhat special: it is an optimized version of wxArray which uses its knowledge about wxString reference counting schema.

10.22.2 STL Build

To build wxWidgets with the standard containers you need to set wxUSE_STD_CONTAINERS option to 1 in `wx/msw/setup.h` for wxMSW builds or specify `-enable-std_containers` option to configure (which is also implicitly enabled by `-enable-stl` option) in Unix builds.

The standard container build is mostly, but not quite, compatible with the default one. Here are the most important differences:

- wxList::compatibility_iterator must be used instead of wxList::Node* when iterating over the list contents. The compatibility_iterator class has the same semantics as a Node pointer but it is an object and not a pointer, so you need to write

```
for ( wxWindowList::compatibility_iterator it = list.GetFirst();
      it;
      it = it->GetNext() )
    ...
```

  instead of the old

```
for ( wxWindowList::Node *n = list.GetFirst(); n; n = n->GetNext() )
    ...
```

- wxSortedArrayString and wxArrayString are separate classes now and the former doesn't derive from the latter. If you need to convert a sorted array to a normal one, you must copy all the elements. Alternatively, you may avoid the use of wxSortedArrayString by using a normal array and calling its Sort() method when needed.
- WX_DEFINE_ARRAY_INT(bool) cannot be used because of the differences in std::vector<bool> specialization compared with the generic std::vector<> class. Please either use std::vector<bool> directly or use an integer array instead.

10.23 File Classes and Functions

wxWidgets provides some functions and classes to facilitate working with files.

As usual, the accent is put on cross-platform features which explains, for example, the wxTextFile class which may be used to convert between different types of text files (DOS/Unix/Mac).

wxFile may be used for low-level IO. It contains all the usual functions to work with files (opening/closing, reading/writing, seeking, and so on) but compared with using standard C functions, has error checking (in case of an error a message is logged using wxLog facilities) and closes the file automatically in the destructor which may be quite convenient.

wxTempFile is a very small file designed to make replacing the files contents safer - see its documentation for more details.

wxTextFile is a general purpose class for working with small text files on line by line basis. It is especially well suited for working with configuration files and program source files. It can be also used to work with files with "non native" line termination characters and write them as "native" files if needed (in fact, the files may be written in any format).

wxDir is a helper class for enumerating the files or subdirectories of a directory. It may be used to enumerate all files, only files satisfying the given template mask or only non-hidden files.

See also

wxFile, wxDir, wxTempFile, wxTextFile, Files and Directories

10.24 Stream Classes Overview

wxWidgets provides its own set of stream classes in order to support platforms not providing standard C++ streams implementation and also to make it possible to provide binary versions of wxWidgets application not depending on any particular standard library version.

The wxWidgets stream classes also provide some functionality not available in the standard library such as support for several compression formats and possibility to work with sockets or text controls (for output only in the latter case).

Nevertheless wxWidgets programs can also use standard stream classes and are encouraged to do so if the above considerations don't apply. Moreover, wxStdInputStream and wxStdOutputStream classes are provided to provide a degree of interoperability between the two and make it possible to use any wxWidgets stream as a standard stream (the converse possibility to use a standard stream as a wxWidgets stream is planned for a future release).

10.24.1 Stream Classes

wxStream classes are divided in two main groups:

- The core: wxStreamBase, wxStreamBuffer, wxInputStream, wxOutputStream, wxFilterInputStream, wxFilterOutputStream
- The "IO" classes: wxSocketInputStream, wxSocketOutputStream, wxFileInputStream, wxFileOutputStream, ...
- Classes for reading text or binary data from a particular stream such as wxTextInputStream, wxTextOutputStream, wxDataInputStream and wxDataOutputStream

wxStreamBase is the base definition of a stream. It defines, for example, the API of OnSysRead(), OnSysWrite(), OnSysSeek() and OnSysTell(). These functions are really implemented by the "IO" classes. wxInputStream and wxOutputStream classes inherit from wxStreamBase and provide specialized methods for input and output.

wxStreamBuffer is a cache manager for wxStreamBase: it manages a stream buffer linked to a stream. One stream can have multiple stream buffers but one stream has always one autoinitialized stream buffer.

wxInputStream is the base class for read-only streams. It implements Read(), SeekI() (I for Input), and all read or IO generic related functions. wxOutputStream does the same thing for write-only streams.

wxFilterInputStream and wxFileterOutputStream are the base class definitions for stream filtering. Stream filtering means a stream which does no syscall but filters data which are passed to it and then pass them to another stream. For example, wxZLibInputStream is an inline stream decompressor.

The "IO" classes implements the specific parts of the stream. This could be nothing in the case of wxMemoryInputStream and wxMemoryOutputStream which base themselves on wxStreamBuffer. This could also be a simple link to the true syscall (for example read(...), write(...)).

10.24.2 Example

Usage is simple. We can take the example of wxFileInputStream and here is some sample code:

```
...
// The constructor initializes the stream buffer and open the file descriptor
// associated to the name of the file.
wxFileInputStream in_stream("the_file_to_be_read");

// Ok, read some bytes ... nb_datas is expressed in bytes.
in_stream.Read(data, nb_datas);
```

```
if (in_stream.LastError() != wxSTREAM_NOERROR) {
    // Oh oh, something bad happens.
    // For a complete list, look into the documentation at wxStreamBase.
}

// You can also inline all like this.
if (in_stream.Read(data, nb_datas).LastError() != wxSTREAM_NOERROR) {
    // Do something.
}

// You can also get the last number of bytes REALLY put into the buffer.
size_t really_read = in_stream.LastRead();

// Ok, moves to the beginning of the stream. SeekI returns the last position
// in the stream counted from the beginning.
off_t old_position = in_stream.SeekI(0, wxFromBeginning);

// What is my current position ?
off_t position = in_stream.TellI();

// wxFileInputStream will close the file descriptor on destruction.
```

10.25 Multithreading Overview

wxWidgets provides a complete set of classes encapsulating objects necessary in multi-threaded (MT) applications: the wxThread class itself and different synchronization objects: mutexes (see wxMutex) and critical sections (see wxCriticalSection) with conditions (see wxCondition).

The thread API in wxWidgets resembles to POSIX1.c threads API (a.k.a. pthreads), although several functions have different names and some features inspired by Win32 thread API are there as well.

These classes hopefully make writing MT programs easier and they also provide some extra error checking (compared to the native - be it Win32 or Posix - thread API), however it is still a non-trivial undertaking especially for large projects. Before starting an MT application (or starting to add MT features to an existing one) it is worth asking oneself if there is no easier and safer way to implement the same functionality. Of course, in some situations threads really make sense (classical example is a server application which launches a new thread for each new client), but in others it might be an overkill. On the other hand, the recent evolution of the computer hardware shows an important trend towards multi-core systems, which are better exploited using multiple threads (e.g. you may want to split a long task among as many threads as many CPU (cores) the system reports; see wxThread::GetCPUCount).

To implement non-blocking operations *without* using multiple threads you have two possible implementation choices:

- use wxIdleEvent (e.g. to perform a long calculation while updating a progress dialog)
- do everything at once but call wxWindow::Update() or wxApp::YieldFor(wxEVT_CATEGORY_UI) periodically to update the screen.

If instead you choose to use threads in your application, please read the following section of this overview.

See also

wxThread, wxThreadHelper, wxMutex, wxCriticalSection, wxCondition, wxSemaphore

10.25.1 Important Notes for Multi-threaded Applications

When writing a multi-threaded application, it is strongly recommended that **no secondary threads call GUI functions**. The design which uses one GUI thread and several worker threads which communicate with the main one using **events** is much more robust and will undoubtedly save you countless problems (example: under Win32 a thread can only access GDI objects such as pens, brushes, device contexts created by itself and not by the other threads).

For communication between secondary threads and the main thread, you may use wxEvtHandler::QueueEvent or its short version wxQueueEvent. These functions have a thread-safe implementation so that they can be used as they are for sending events from one thread to another. However there is no built in method to send messages to the

worker threads and you will need to use the available synchronization classes to implement the solution which suits your needs yourself. In particular, please note that it is not enough to derive your class from wxThread and wxEvtHandler to send messages to it: in fact, this does not work at all. You're instead encouraged to use wxThreadHelper as it greatly simplifies the communication and the sharing of resources.

You should also look at the wxThread docs for important notes about secondary threads and their deletion.

Last, remember that if wxEventLoopBase::YieldFor() is used directly or indirectly (e.g. through wxProgressDialog) in your code, then you may have both re-entrancy problems and also problems caused by the processing of events out of order. To resolve the last problem wxThreadEvent can be used: thanks to its implementation of the wxThreadEvent::GetEventCategory function wxThreadEvent classes in fact do not get processed by wxEventLoopBase::YieldFor() unless you specify the `wxEVT_CATEGORY_THREAD` flag.

See also the Thread Sample for a sample showing some simple interactions between the main and secondary threads.

10.26 wxConfig Overview

Classes: wxConfigBase

This overview briefly describes what the config classes are and what they are for. All the details about how to use them may be found in the description of the wxConfigBase class and the documentation of the file, registry and INI file based implementations mentions all the features/limitations specific to each one of these versions.

The config classes provide a way to store some application configuration information. They were especially designed for this usage and, although may probably be used for many other things as well, should be limited to it. It means that this information should be:

- Typed, i.e. strings or numbers for the moment. You cannot store binary data, for example.
- Small. For instance, it is not recommended to use the Windows registry for amounts of data more than a couple of kilobytes.
- Not performance critical, neither from speed nor from a memory consumption point of view.

On the other hand, the features provided make them very useful for storing all kinds of small to medium volumes of hierarchically-organized, heterogeneous data. In short, this is a place where you can conveniently stuff all your data (numbers and strings) organizing it in a tree where you use the filesystem-like paths to specify the location of a piece of data. In particular, these classes were designed to be as easy to use as possible.

From another point of view, they provide an interface which hides the differences between the Windows registry and the standard Unix text format configuration files. Other (future) implementations of wxConfigBase might also understand GTK resource files or their analogues on the KDE side.

In any case, each implementation of wxConfigBase does its best to make the data look the same way everywhere. Due to limitations of the underlying physical storage, it may not implement 100% of the base class functionality.

There are groups of entries and the entries themselves. Each entry contains either a string or a number (or a boolean value; support for other types of data such as dates or timestamps is planned) and is identified by the full path to it: something like `/MyApp/UserPreferences/Colors/Foreground`.

The previous elements in the path are the group names, and each name may contain an arbitrary number of entries and subgroups.

The path components are *always* separated with a slash, even though some implementations use the backslash internally. Further details (including how to read/write these entries) may be found in the documentation for wxConfigBase.

10.27 Persistent Objects Overview

Persistent objects are simply the objects which automatically save their state when they are destroyed and restore it when they are recreated, even during another program invocation.

Most often, persistent objects are, in fact, persistent windows as it is especially convenient to automatically restore the UI state when the program is restarted but an object of any class can be made persistent. Moreover, persistence is implemented in a non-intrusive way so that the original object class doesn't need to be modified at all in order to add support for saving and restoring its properties.

The persistence framework includes the following components:

- wxPersistenceManager which all persistent objects register themselves with. This class handles actual saving and restoring of persistent data as well as various global aspects of persistence, e.g. it can be used to disable restoring the saved data.
- wxPersistentObject is the base class for all persistent objects or, rather, adaptors for the persistent objects as this class main purpose is to provide the bridge between the original class – which has no special persistence support – and wxPersistenceManager,
- wxPersistentWindow<> which derives from wxPersistentObject and implements some of its methods using wxWindow-specific functionality. Notably, wxPersistenceManager handles the destruction of persistent windows automatically implicitly while it has to be done explicitly for the arbitrary persistent objects.
- wxCreatePersistentObject() function which is used to create the appropriate persistence adapter for the object.

10.27.1 Using Persistent Windows

wxWidgets has built-in support for a (constantly growing) number of controls. Currently the following classes are supported:

- wxTopLevelWindow (and hence wxFrame and wxDialog)
- wxBookCtrlBase (i.e. wxNotebook, wxListbook, wxToolbook and wxChoicebook)
- wxTreebook

To automatically save and restore the properties of the windows of classes listed above you need to:

1. Set a unique name for the window using wxWindow::SetName(): this step is important as the name is used in the configuration file and so must be unique among all windows of the same class.
2. Call wxPersistenceManager::Register() at any moment after creating the window and then wxPersistenceManager::Restore() when the settings may be restored (which can't be always done immediately, e.g. often the window needs to be populated first). If settings can be restored immediately after the window creation, as is often the case for wxTopLevelWindow, for example, then wxPersistenceManager::RegisterAndRestore() can be used to do both at once.
3. If you do not want the settings for the window to be saved (for example the changes to the dialog size are usually not saved if the dialog was cancelled), you need to call wxPersistenceManager::Unregister() manually. Otherwise the settings will be automatically saved when the control itself is destroyed.

Example of using a notebook control which automatically remembers the last open page:

```
wxNotebook *book = new wxNotebook(parent, wxID_ANY);
book->SetName("MyBook"); // do not use the default name
book->AddPage(...);
book->AddPage(...);
book->AddPage(...);
if ( !wxPersistenceManager::RegisterAndRestore(book) )
{
    // nothing was restored, so choose the default page ourselves
    book->SetSelection(0);
}
```

10.27.2 Defining Custom Persistent Windows

User-defined classes can be easily integrated with wxPersistenceManager. To add support for your custom class `MyWidget` you just need to:

1. Define a new `MyPersistentWidget` class inheriting from wxPersistentWindow<MyWidget>.
2. Implement its pure virtual GetKind() method returning a unique string identifying all `MyWidget` objects, typically something like `"widget"`
3. Implement its pure virtual Save() and Restore() methods to actually save and restore the widget settings using wxPersistentObject::SaveValue() and wxPersistentObject::RestoreValue() methods.
4. Define wxCreatePersistentObject() overload taking `MyWidget *` and returning a new `MyPersistentWidget` object.

If you want to add persistence support for a class not deriving from wxWindow, you need to derive `MyPersistentWidget` directly from wxPersistentObject and so implement its pure virtual wxPersistentObject::GetName() method too. Additionally, you must ensure that wxPersistenceManager::SaveAndUnregister() is called when your object is destroyed as this can be only done automatically for windows.

10.28 wxFileSystem Overview

The wxHTML library uses a **virtual** file system mechanism similar to the one used in Midnight Commander, Dos Navigator, FAR or almost any modern file manager.

It allows the user to access data stored in archives as if they were ordinary files. On-the-fly generated files that exist only in memory are also supported.

10.28.1 Classes

Three classes are used in order to provide virtual file systems mechanism:

- The wxFSFile class provides information about opened file (name, input stream, mime type and anchor).
- The wxFileSystem class is the interface. Its main methods are ChangePathTo() and OpenFile(). This class is most often used by the end user.
- The wxFileSystemHandler is the core of virtual file systems mechanism. You can derive your own handler and pass it to the VFS mechanism. You can derive your own handler and pass it to wxFileSystem's AddHandler() method. In the new handler you only need to override the OpenFile() and CanOpen() methods.

10.28.2 Locations

Locations (aka filenames aka addresses) are constructed from four parts:

- **protocol** - handler can recognize if it is able to open a file by checking its protocol. Examples are "http", "file" or "ftp".
- **right location** - is the name of file within the protocol. In "http://www.wxwidgets.org/index.html" the right location is "//www.wxwidgets.org/index.html".
- **anchor** - an anchor is optional and is usually not present. In "index.htm#chapter2" the anchor is "chapter2".
- **left location** - this is usually an empty string. It is used by 'local' protocols such as ZIP. See Combined Protocols paragraph for details.

10.28.3 Combined Protocols

The left location precedes the protocol in the URL string.

It is not used by global protocols like HTTP but it becomes handy when nesting protocols - for example you may want to access files in a ZIP archive: `file:archives/cpp_doc.zip#zip:reference/fopen.htm#syntax` In this example, the protocol is "zip", right location is "reference/fopen.htm", anchor is "syntax" and left location is "file:archives/cpp_doc.zip".

There are **two** protocols used in this example: "zip" and "file".

10.28.4 File Systems Included in wxHTML

The following virtual file system handlers are part of wxWidgets so far:

- **wxArchiveFSHandler:** A handler for archives such as zip and tar. Include file is wx/fs_arc.h. URLs examples: "archive.zip#zip:filename", "archive.tar.gz#gzip:#tar:filename".
- **wxFilterFSHandler:** A handler for compression schemes such as gzip. Header is wx/fs_filter.h. URLs are in the form, e.g.: "document.ps.gz#gzip:".
- **wxInternetFSHandler:** A handler for accessing documents via HTTP or FTP protocols. Include file is wx/fs_inet.h.
- **wxMemoryFSHandler:** This handler allows you to access data stored in memory (such as bitmaps) as if they were regular files. See wxMemoryFSHandler for details. Include file is wx/fs_mem.h. URL is prefixed with memory:, e.g. "memory:myfile.htm"

In addition, wxFileSystem itself can access local files.

10.28.5 Initializing file system handlers

Use wxFileSystem::AddHandler to initialize a handler, for example:

```
#include <wx/fs_mem.h>

...

bool MyApp::OnInit()
{
    wxFileSystem::AddHandler(new wxMemoryFSHandler);
...
}
```

10.29 Regular Expressions

A *regular expression* describes strings of characters.

It's a pattern that matches certain strings and doesn't match others.

See also

wxRegEx

10.29.1 Different Flavors of Regular Expressions

Regular expressions (RE), as defined by POSIX, come in two flavors: *extended regular expressions* (ERE) and *basic regular expressions* (BRE). EREs are roughly those of the traditional *egrep*, while BREs are roughly those of

the traditional *ed*. This implementation adds a third flavor: *advanced regular expressions* (ARE), basically EREs with some significant extensions.

This manual page primarily describes AREs. BREs mostly exist for backward compatibility in some old programs. POSIX EREs are almost an exact subset of AREs. Features of AREs that are not present in EREs will be indicated.

10.29.2 Regular Expression Syntax

These regular expressions are implemented using the package written by Henry Spencer, based on the 1003.2 spec and some (not quite all) of the Perl5 extensions (thanks, Henry!). Much of the description of regular expressions below is copied verbatim from his manual entry.

An ARE is one or more *branches*, separated by "|", matching anything that matches any of the branches.

A branch is zero or more *constraints* or *quantified* atoms, concatenated. It matches a match for the first, followed by a match for the second, etc; an empty branch matches the empty string.

A quantified atom is an *atom* possibly followed by a single *quantifier*. Without a quantifier, it matches a match for the atom. The quantifiers, and what a so-quantified atom matches, are:

`*`	A sequence of 0 or more matches of the atom.
`+`	A sequence of 1 or more matches of the atom.
`?`	A sequence of 0 or 1 matches of the atom.
`{m}`	A sequence of exactly *m* matches of the atom.
`{m,}`	A sequence of *m* or more matches of the atom.
`{m,n}`	A sequence of *m* through *n* (inclusive) matches of the atom; *m* may not exceed *n*.
`*? +? ?? {m}? {m,}? {m,n}?`	*Non-greedy* quantifiers, which match the same possibilities, but prefer the smallest number rather than the largest number of matches (see Matching).

The forms using **{** and **}** are known as *bounds*. The numbers *m* and *n* are unsigned decimal integers with permissible values from 0 to 255 inclusive. An atom is one of:

`(re)`	Where *re* is any regular expression, matches for *re*, with the match captured for possible reporting.
`(?:re)`	As previous, but does no reporting (a "non-capturing" set of parentheses).
`()`	Matches an empty string, captured for possible reporting.
`(?:)`	Matches an empty string, without reporting.
`[chars]`	A *bracket expression*, matching any one of the *chars* (see Bracket Expressions for more details).
`.`	Matches any single character.
`\k`	Where *k* is a non-alphanumeric character, matches that character taken as an ordinary character, e.g. \\ matches a backslash character.
`\c`	Where *c* is alphanumeric (possibly followed by other characters), an *escape* (AREs only), see Escapes below.
`{`	When followed by a character other than a digit, matches the left-brace character "{"; when followed by a digit, it is the beginning of a *bound* (see above).
`x`	Where *x* is a single character with no other significance, matches that character.

A *constraint* matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier. The simple constraints are as follows; some more constraints are described later, under Escapes.

^	Matches at the beginning of a line.
$	Matches at the end of a line.
(?=re)	*Positive* lookahead (AREs only), matches at any point where a substring matching *re* begins.
(?!re)	*Negative* lookahead (AREs only), matches at any point where no substring matching *re* begins.

The lookahead constraints may not contain back references (see later), and all parentheses within them are considered non-capturing. A RE may not end with "\".

10.29.3 Bracket Expressions

A *bracket expression* is a list of characters enclosed in `[]`. It normally matches any single character from the list (but see below). If the list begins with ^, it matches any single character (but see below) *not* from the rest of the list.

If two characters in the list are separated by `-`, this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. `[0-9]` in ASCII matches any decimal digit. Two ranges may not share an endpoint, so e.g. `a-c-e` is illegal. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal `]` or `-` in the list, the simplest method is to enclose it in `[.` and `.]` to make it a collating element (see below). Alternatively, make it the first character (following a possible ^), or (AREs only) precede it with `\`. Alternatively, for `-`, make it the last character, or the second endpoint of a range. To use a literal `-` as the first endpoint of a range, make it a collating element or (AREs only) precede it with `\`. With the exception of these, some combinations using `[` (see next paragraphs), and escapes, all other special characters lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in `[.` and `.]` stands for the sequence of characters of that collating element.

wxWidgets: Currently no multi-character collating elements are defined. So in `[.X.]`, `X` can either be a single character literal or the name of a character. For example, the following are both identical: `[[.0.]-[.9.]]` and `[[.zero.]-[.nine.]]` and mean the same as `[0-9]`. See Regular Expression Character Names.

Within a bracket expression, a collating element enclosed in `[=` and `=]` is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. An equivalence class may not be an endpoint of a range.

wxWidgets: Currently no equivalence classes are defined, so `[=X=]` stands for just the single character `X`. `X` can either be a single character literal or the name of a character, see Regular Expression Character Names.

Within a bracket expression, the name of a *character* class enclosed in `[:` and `:]` stands for the list of all characters (not all collating elements!) belonging to that class. Standard character classes are:

alpha	A letter.
upper	An upper-case letter.
lower	A lower-case letter.
digit	A decimal digit.
xdigit	A hexadecimal digit.
alnum	An alphanumeric (letter or digit).
print	An alphanumeric (same as alnum).
blank	A space or tab character.
space	A character producing white space in displayed text.
punct	A punctuation character.
graph	A character with a visible representation.
cntrl	A control character.

A character class may not be used as an endpoint of a range.

wxWidgets: In a non-Unicode build, these character classifications depend on the current locale, and correspond to the values return by the ANSI C "is" functions: `isalpha`, `isupper`, etc. In Unicode mode they are based on Unicode classifications, and are not affected by the current locale.

There are two special cases of bracket expressions: the bracket expressions `[[:<:]]` and `[[:>:]]` are constraints, matching empty strings at the beginning and end of a word respectively. A word is defined as a sequence of word characters that is neither preceded nor followed by word characters. A word character is an *alnum* character or an underscore (_). These special bracket expressions are deprecated; users of AREs should use constraint escapes instead (see escapes below).

10.29.4 Escapes

Escapes (AREs only), which begin with a \ followed by an alphanumeric character, come in several varieties: character entry, class shorthands, constraint escapes, and back references. A \ followed by an alphanumeric character but not constituting a valid escape is illegal in AREs. In EREs, there are no escapes: outside a bracket expression, a \ followed by an alphanumeric character merely stands for that character as an ordinary character, and inside a bracket expression, \ is an ordinary character. (The latter is the one actual incompatibility between EREs and AREs.)

Character-entry escapes (AREs only) exist to make it easier to specify non-printing and otherwise inconvenient characters in REs:

`\a`	Alert (bell) character, as in C.
`\b`	Backspace, as in C.
`\B`	Synonym for \ to help reduce backslash doubling in some applications where there are multiple levels of backslash processing.
`\cX`	The character whose low-order 5 bits are the same as those of *X*, and whose other bits are all zero, where *X* is any character.
`\e`	The character whose collating-sequence name is `ESC`, or failing that, the character with octal value 033.
`\f`	Formfeed, as in C.
`\n`	Newline, as in C.
`\r`	Carriage return, as in C.
`\t`	Horizontal tab, as in C.
`\uwxyz`	The Unicode character `U+wxyz` in the local byte ordering, where *wxyz* is exactly four hexadecimal digits.
`\Ustuvwxyz`	Reserved for a somewhat-hypothetical Unicode extension to 32 bits, where *stuvwxyz* is exactly eight hexadecimal digits.
`\v`	Vertical tab, as in C are all available.
`\xhhh`	The single character whose hexadecimal value is *0xhhh*, where *hhh* is any sequence of hexadecimal digits.
`\0`	The character whose value is 0.
`\xy`	The character whose octal value is *0xy*, where *xy* is exactly two octal digits, and is not a *back reference* (see below).
`\xyz`	The character whose octal value is *0xyz*, where *xyz* is exactly three octal digits, and is not a *back reference* (see below).

Hexadecimal digits are 0-9, a-f, and A-F. Octal digits are 0-7.

The character-entry escapes are always taken as ordinary characters. For example, `\135` is `]` in ASCII, but `\135` does not terminate a bracket expression. Beware, however, that some applications (e.g., C compilers) interpret such sequences themselves before the regular-expression package gets to see them, which may require doubling (quadrupling, etc.) the '\'.

Class-shorthand escapes (AREs only) provide shorthands for certain commonly-used character classes:

\d	[[:digit:]]
\s	[[:space:]]
\w	[[:alnum:]_] (note underscore)
\D	[^[:digit:]]
\S	[^[:space:]]
\W	[^[:alnum:]_] (note underscore)

Within bracket expressions, \d, \s, and \w lose their outer brackets, and \D, \S, \W are illegal. So, for example, [a-c\d] is equivalent to [a-c[:digit:]]. Also, [a-c\D], which is equivalent to [a-c^[:digit:]], is illegal.

A constraint escape (AREs only) is a constraint, matching the empty string if specific conditions are met, written as an escape:

\A	Matches only at the beginning of the string, see Matching for how this differs from ^.
\m	Matches only at the beginning of a word.
\M	Matches only at the end of a word.
\y	Matches only at the beginning or end of a word.
\Y	Matches only at a point that is not the beginning or end of a word.
\Z	Matches only at the end of the string, see Matching for how this differs from $.
\m	A *back reference*, where *m* is a non-zero digit. See below.
\mnn	A *back reference*, where *m* is a nonzero digit, and *nn* is some more digits, and the decimal value *mnn* is not greater than the number of closing capturing parentheses seen so far. See below.

A word is defined as in the specification of [[:<:]] and [[:>:]] above. Constraint escapes are illegal within bracket expressions.

A back reference (AREs only) matches the same string matched by the parenthesized subexpression specified by the number. For example, "([bc])\1" matches "bb" or "cc" but not "bc". The subexpression must entirely precede the back reference in the RE.Subexpressions are numbered in the order of their leading parentheses. Non-capturing parentheses do not define subexpressions.

There is an inherent historical ambiguity between octal character-entry escapes and back references, which is resolved by heuristics, as hinted at above. A leading zero always indicates an octal escape. A single non-zero digit, not followed by another digit, is always taken as a back reference. A multi-digit sequence not starting with a zero is taken as a back reference if it comes after a suitable subexpression (i.e. the number is in the legal range for a back reference), and otherwise is taken as octal.

10.29.5 Metasyntax

In addition to the main syntax described above, there are some special forms and miscellaneous syntactic facilities available.

Normally the flavor of RE being used is specified by application-dependent means. However, this can be overridden by a *director*. If an RE of any flavor begins with ***:, the rest of the RE is an ARE. If an RE of any flavor begins with ***=, the rest of the RE is taken to be a literal string, with all characters considered ordinary characters.

An ARE may begin with *embedded options*: a sequence (?xyz) (where *xyz* is one or more alphabetic characters) specifies options affecting the rest of the RE. These supplement, and can override, any options specified by the application. The available option letters are:

b	Rest of RE is a BRE.
c	Case-sensitive matching (usual default).
e	Rest of RE is an ERE.
i	Case-insensitive matching (see Matching, below).
m	Historical synonym for *n*.
n	Newline-sensitive matching (see Matching, below).
p	Partial newline-sensitive matching (see Matching, below).
q	Rest of RE is a literal ("quoted") string, all ordinary characters.
s	Non-newline-sensitive matching (usual default).
t	Tight syntax (usual default; see below).
w	Inverse partial newline-sensitive ("weird") matching (see Matching, below).
x	Expanded syntax (see below).

Embedded options take effect at the `)` terminating the sequence. They are available only at the start of an ARE, and may not be used later within it.

In addition to the usual (*tight*) RE syntax, in which all characters are significant, there is an *expanded* syntax, available in AREs with the embedded x option. In the expanded syntax, white-space characters are ignored and all characters between a `#` and the following newline (or the end of the RE) are ignored, permitting paragraphing and commenting a complex RE. There are three exceptions to that basic rule:

- A white-space character or `#` preceded by `\` is retained.
- White space or `#` within a bracket expression is retained.
- White space and comments are illegal within multi-character symbols like the ARE `(?:` or the BRE `(.`

Expanded-syntax white-space characters are blank, tab, newline, and any character that belongs to the *space* character class.

Finally, in an ARE, outside bracket expressions, the sequence `(?#ttt)` (where *ttt* is any text not containing a `)`) is a comment, completely ignored. Again, this is not allowed between the characters of multi-character symbols like `(?:`. Such comments are more a historical artifact than a useful facility, and their use is deprecated; use the expanded syntax instead.

None of these metasyntax extensions is available if the application (or an initial `***=` director) has specified that the user's input be treated as a literal string rather than as an RE.

10.29.6 Matching

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, the choice is determined by it's *preference:* either the longest substring, or the shortest.

Most atoms, and all constraints, have no preference. A parenthesized RE has the same preference (possibly none) as the RE. A quantified atom with quantifier `{m}` or `{m}?` has the same preference (possibly none) as the atom itself. A quantified atom with other normal quantifiers (including `{m,n}` with *m* equal to *n*) prefers longest match. A quantified atom with other non-greedy quantifiers (including `{m,n}?` with *m* equal to *n*) prefers shortest match. A branch has the same preference as the first quantified atom in it which has a preference. An RE consisting of two or more branches connected by the `|` operator prefers longest match.

Subject to the constraints imposed by the rules for matching the whole RE, subexpressions also match the longest or shortest possible substrings, based on their preferences, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that outer subexpressions thus take priority over their component subexpressions.

Note that the quantifiers `{1,1}` and `{1,1}?` can be used to force longest and shortest preference, respectively, on a subexpression or a whole RE.

Match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all. For example, `bb*` matches the three middle characters of "abbbc", `(week|wee)(night|knights)` matches all ten characters of "weeknights", when `(.*).*` is matched against "abc" the parenthesized subexpression matches all three characters, and when `(a*)*` is matched against "bc" both the whole RE and the parenthesized subexpression match an empty string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, so that `x` becomes [xX]. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that [x] becomes [xX] and [^x] becomes [^xX].

If newline-sensitive matching is specified, "." and bracket expressions using "^" will never match the newline character (so that matches will never cross newlines unless the RE explicitly arranges it) and "^" and "$" will match the empty string after and before a newline respectively, in addition to matching at beginning and end of string respectively. ARE `\A` and `\Z` continue to match beginning or end of string *only*.

If partial newline-sensitive matching is specified, this affects "." and bracket expressions as with newline-sensitive matching, but not "^" and "$".

If inverse partial newline-sensitive matching is specified, this affects "^" and "$" as with newline-sensitive matching, but not "." and bracket expressions. This isn't very useful but is provided for symmetry.

10.29.7 Limits and Compatibility

No particular limit is imposed on the length of REs. Programs intended to be highly portable should not employ REs longer than 256 bytes, as a POSIX-compliant implementation can refuse to accept such REs.

The only feature of AREs that is actually incompatible with POSIX EREs is that `\` does not lose its special significance inside bracket expressions. All other ARE features use syntax which is illegal or has undefined or unspecified effects in POSIX EREs; the `***` syntax of directors likewise is outside the POSIX syntax for both BREs and EREs.

Many of the ARE extensions are borrowed from Perl, but some have been changed to clean them up, and a few Perl extensions are not present. Incompatibilities of note include `\b`, `\B`, the lack of special treatment for a trailing newline, the addition of complemented bracket expressions to the things affected by newline-sensitive matching, the restrictions on parentheses and back references in lookahead constraints, and the longest/shortest-match (rather than first-match) matching semantics.

The matching rules for REs containing both normal and non-greedy quantifiers have changed since early beta-test versions of this package. The new rules are much simpler and cleaner, but don't work as hard at guessing the user's real intentions.

Henry Spencer's original 1986 *regexp* package, still in widespread use, implemented an early version of today's E↩ REs. There are four incompatibilities between *regexp's* near-EREs (RREs for short) and AREs. In roughly increasing order of significance:

- In AREs, `\` followed by an alphanumeric character is either an escape or an error, while in RREs, it was just another way of writing the alphanumeric. This should not be a problem because there was no reason to write such a sequence in RREs.
- `{` followed by a digit in an ARE is the beginning of a bound, while in RREs, `{` was always an ordinary character. Such sequences should be rare, and will often result in an error because following characters will not look like a valid bound.
- In AREs, `\` remains a special character within [], so a literal `\` within [] must be written as `\\`. `\\` also gives a literal `\` within [] in RREs, but only truly paranoid programmers routinely doubled the backslash.
- AREs report the longest/shortest match for the RE, rather than the first found in a specified search order. This may affect some RREs which were written in the expectation that the first match would be reported. The careful crafting of RREs to optimize the search order for fast matching is obsolete (AREs examine all possible matches in parallel, and their performance is largely insensitive to their complexity) but cases where the search order was exploited to deliberately find a match which was *not* the longest/shortest will need rewriting.

10.29.8 Basic Regular Expressions

BREs differ from EREs in several respects. |, +, and ? are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are \{ and \}, with { and } by themselves ordinary characters. The parentheses for nested subexpressions are @(and @), with (and) by themselves ordinary characters. ^ is an ordinary character except at the beginning of the RE or the beginning of a parenthesized subexpression, $ is an ordinary character except at the end of the RE or the end of a parenthesized subexpression, and * is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading ^). Finally, single-digit back references are available, and \< and \> are synonyms for [[:<:]] and [[:>:]] respectively; no other escapes are available.

10.29.9 Regular Expression Character Names

Note that the character names are case sensitive.

NUL	\0
SOH	\001
STX	\002
ETX	\003
EOT	\004
ENQ	\005
ACK	\006
BEL	\007
alert	\007
BS	\010
backspace	\b
HT	\011
tab	\t
LF	\012
newline	\n
VT	\013
vertical-tab	\v
FF	\014
form-feed	\f

CR	\015
carriage-return	\r
SO	\016
SI	\017
DLE	\020
DC1	\021
DC2	\022
DC3	\023
DC4	\024
NAK	\025
SYN	\026
ETB	\027
CAN	\030
EM	\031
SUB	\032
ESC	\033
IS4	\034
FS	\034
IS3	\035

GS	\035
IS2	\036
RS	\036
IS1	\037
US	\037
space	" " (space)
exclamation-mark	!
quotation-mark	"
number-sign	#
dollar-sign	$
percent-sign	%
ampersand	&
apostrophe	'
left-parenthesis	(
right-parenthesis	)
asterisk	*
plus-sign	+
comma	,
hyphen	-

hyphen-minus	-
period	.
full-stop	.
slash	/
solidus	/
zero	0
one	1
two	2
three	3
four	4
five	5
six	6
seven	7
eight	8
nine	9
colon	:
semicolon	;
less-than-sign	<
equals-sign	=

greater-than-sign	>
question-mark	?
commercial-at	@
left-square-bracket	[
backslash	\
reverse-solidus	\
right-square-bracket	]
circumflex	^
circumflex-accent	^
underscore	_
low-line	_
grave-accent	'
left-brace	{
left-curly-bracket	{
vertical-line	|
right-brace	}
right-curly-bracket	}
tilde	~
DEL	\177

10.30 Archive Formats

The archive classes handle archive formats such as zip, tar, rar and cab.

Currently wxZip, wxTar and wxZlib classes are included.

For each archive type, there are the following classes (using zip here as an example):

- wxZipInputStream: Input stream
- wxZipOutputStream: Output stream
- wxZipEntry: Holds meta-data for an entry (e.g. filename, timestamp, etc.)

There are also abstract wxArchive classes that can be used to write code that can handle any of the archive types, see Generic Archive Programming.

Also see wxFileSystem for a higher level interface that can handle archive files in a generic way.

The classes are designed to handle archives on both seekable streams such as disk files, or non-seekable streams such as pipes and sockets (see Archives on Non-Seekable Streams).

10.30.1 Creating an Archive

Call wxArchiveOutputStream::PutNextEntry() to create each new entry in the archive, then write the entry's data. Another call to PutNextEntry() closes the current entry and begins the next. For example:

```
wxFFileOutputStream out(wxT("test.zip"));
wxZipOutputStream zip(out);
wxTextOutputStream txt(zip);
wxString sep(wxFileName::GetPathSeparator());

zip.PutNextEntry(wxT("entry1.txt"));
txt << wxT("Some text for entry1.txt\n");

zip.PutNextEntry(wxT("subdir") + sep + wxT("entry2.txt"));
txt << wxT("Some text for subdir/entry2.txt\n");
```

The name of each entry can be a full path, which makes it possible to store entries in subdirectories.

10.30.2 Extracting an Archive

wxArchiveInputStream::GetNextEntry() returns a pointer to entry object containing the meta-data for the next entry in the archive (and gives away ownership).

Reading from the input stream then returns the entry's data. Eof() becomes true after an attempt has been made to read past the end of the entry's data.

When there are no more entries, GetNextEntry() returns NULL and sets Eof().

```
auto_ptr<wxZipEntry> entry;

wxFFileInputStream in(wxT("test.zip"));
wxZipInputStream zip(in);

while (entry.reset(zip.GetNextEntry()), entry.get() != NULL)
{
    // access meta-data
    wxString name = entry->GetName();
    // read 'zip' to access the entry's data
}
```

10.30.3 Modifying an Archive

To modify an existing archive, write a new copy of the archive to a new file, making any necessary changes along the way and transferring any unchanged entries using wxArchiveOutputStream::CopyEntry().

For archive types which compress entry data, CopyEntry() is likely to be much more efficient than transferring the data using Read() and Write() since it will copy them without decompressing and recompressing them.

In general modifications are not possible without rewriting the archive, though it may be possible in some limited cases. Even then, rewriting the archive is usually a better choice since a failure can be handled without losing the whole archive. wxTempFileOutputStream can be helpful to do this.

For example to delete all entries matching the pattern "*.txt":

```
auto_ptr<wxFFileInputStream> in(new wxFFileInputStream(wxT("test.zip")));
wxTempFileOutputStream out(wxT("test.zip"));

wxZipInputStream inzip(*in);
wxZipOutputStream outzip(out);

auto_ptr<wxZipEntry> entry;

// transfer any meta-data for the archive as a whole (the zip comment
// in the case of zip)
outzip.CopyArchiveMetaData(inzip);

// call CopyEntry for each entry except those matching the pattern
while (entry.reset(inzip.GetNextEntry()), entry.get() != NULL)
    if (!entry->GetName().Matches(wxT("*.txt")))
        if (!outzip.CopyEntry(entry.release(), inzip))
            break;

// close the input stream by releasing the pointer to it, do this
// before closing the output stream so that the file can be replaced
in.reset();

// you can check for success as follows
bool success = inzip.Eof() && outzip.Close() && out.Commit();
```

10.30.4 Looking Up an Archive Entry by Name

Also see wxFileSystem for a higher level interface that is more convenient for accessing archive entries by name.

To open just one entry in an archive, the most efficient way is to simply search for it linearly by calling wxArchiveInputStream::GetNextEntry() until the required entry is found. This works both for archives on seekable and non-seekable streams.

The format of filenames in the archive is likely to be different from the local filename format. For example zips and tars use unix style names, with forward slashes as the path separator, and absolute paths are not allowed. So if on Windows the file "C:\MYDIR\MYFILE.TXT" is stored, then when reading the entry back wxArchiveEntry::GetName() will return "MYDIR\MYFILE.TXT". The conversion into the internal format and back has lost some information.

So to avoid ambiguity when searching for an entry matching a local name, it is better to convert the local name to the archive's internal format and search for that:

```
auto_ptr<wxZipEntry> entry;

// convert the local name we are looking for into the internal format
wxString name = wxZipEntry::GetInternalName(localname);

// open the zip
wxFFileInputStream in(wxT("test.zip"));
wxZipInputStream zip(in);

// call GetNextEntry() until the required internal name is found
do
{
    entry.reset(zip.GetNextEntry());
}
while (entry.get() != NULL && entry->GetInternalName() != name);

if (entry.get() != NULL)
{
    // read the entry's data...
}
```

To access several entries randomly, it is most efficient to transfer the entire catalogue of entries to a container such as a std::map or a wxHashMap then entries looked up by name can be opened using the wxArchiveInputStream::OpenEntry() method.

```
WX_DECLARE_STRING_HASH_MAP(wxZipEntry*, ZipCatalog);
ZipCatalog::iterator it;
wxZipEntry *entry;
ZipCatalog cat;

// open the zip
wxFFileInputStream in(wxT("test.zip"));
wxZipInputStream zip(in);

// load the zip catalog
while ((entry = zip.GetNextEntry()) != NULL)
{
    wxZipEntry*& current = cat[entry->GetInternalName()];
    // some archive formats can have multiple entries with the same name
    // (e.g. tar) though it is an error in the case of zip
    delete current;
    current = entry;
}

// open an entry by name
if ((it = cat.find(wxZipEntry::GetInternalName(localname))) != cat.end())
{
    zip.OpenEntry(*it->second);
    // ... now read entry's data
}
```

To open more than one entry simultaneously you need more than one underlying stream on the same archive:

```
// opening another entry without closing the first requires another
// input stream for the same file
wxFFileInputStream in2(wxT("test.zip"));
wxZipInputStream zip2(in2);
if ((it = cat.find(wxZipEntry::GetInternalName(local2))) != cat.end())
    zip2.OpenEntry(*it->second);
```

10.30.5 Generic Archive Programming

Also see wxFileSystem for a higher level interface that can handle archive files in a generic way.

The specific archive classes, such as the wxZip classes, inherit from the following abstract classes which can be used to write code that can handle any of the archive types:

- wxArchiveInputStream: Input stream
- wxArchiveOutputStream: Output stream
- wxArchiveEntry: Holds the meta-data for an entry (e.g. filename)

In order to able to write generic code it's necessary to be able to create instances of the classes without knowing which archive type is being used.

To allow this there is a class factory for each archive type, derived from wxArchiveClassFactory, that can create the other classes.

For example, given wxArchiveClassFactory* factory, streams and entries can be created like this:

```
// create streams without knowing their type
auto_ptr<wxArchiveInputStream> inarc(factory->NewStream(in));
auto_ptr<wxArchiveOutputStream> outarc(factory->NewStream(out));

// create an empty entry object
auto_ptr<wxArchiveEntry> entry(factory->NewEntry());
```

For the factory itself, the static member wxArchiveClassFactory::Find() can be used to find a class factory that can handle a given file extension or mime type. For example, given *filename:*

```
const wxArchiveClassFactory *factory;
factory = wxArchiveClassFactory::Find(filename,
      wxSTREAM_FILEEXT);

if (factory)
    stream = factory->NewStream(new wxFFileInputStream(filename));
```

Find() does not give away ownership of the returned pointer, so it does not need to be deleted.

There are similar class factories for the filter streams that handle the compression and decompression of a single stream, such as wxGzipInputStream. These can be found using wxFilterClassFactory::Find().

For example, to list the contents of archive *filename:*

```
auto_ptr<wxInputStream> in(new wxFFileInputStream(filename));

if (in->IsOk())
{
    // look for a filter handler, e.g. for '.gz'
    const wxFilterClassFactory *fcf;
    fcf = wxFilterClassFactory::Find(filename,
      wxSTREAM_FILEEXT);
    if (fcf)
    {
        in.reset(fcf->NewStream(in.release()));
        // pop the extension, so if it was '.tar.gz' it is now just '.tar'
        filename = fcf->PopExtension(filename);
    }

    // look for a archive handler, e.g. for '.zip' or '.tar'
    const wxArchiveClassFactory *acf;
    acf = wxArchiveClassFactory::Find(filename,
      wxSTREAM_FILEEXT);
    if (acf)
    {
        auto_ptr<wxArchiveInputStream> arc(acf->NewStream(in.release()));
        auto_ptr<wxArchiveEntry> entry;

        // list the contents of the archive
        while ((entry.reset(arc->GetNextEntry())), entry.get() != NULL)
            std::wcout << entry->GetName().c_str() << "\n";
    }
    else
    {
        wxLogError(wxT("can't handle '%s'"), filename.c_str());
    }
}
```

10.30.6 Archives on Non-Seekable Streams

In general, handling archives on non-seekable streams is done in the same way as for seekable streams, with a few caveats.

The main limitation is that accessing entries randomly using wxArchiveInputStream::OpenEntry() is not possible, the entries can only be accessed sequentially in the order they are stored within the archive.

For each archive type, there will also be other limitations which will depend on the order the entries' meta-data is stored within the archive. These are not too difficult to deal with, and are outlined below.

PutNextEntry and the Entry Size

When writing archives, some archive formats store the entry size before the entry's data (tar has this limitation, zip doesn't). In this case the entry's size must be passed to wxArchiveOutputStream::PutNextEntry() or an error occurs.

This is only an issue on non-seekable streams, since otherwise the archive output stream can seek back and fix up the header once the size of the entry is known.

For generic programming, one way to handle this is to supply the size whenever it is known, and rely on the error message from the output stream when the operation is not supported.

GetNextEntry and the Weak Reference Mechanism

Some archive formats do not store all an entry's meta-data before the entry's data (zip is an example). In this case, when reading from a non-seekable stream, wxArchiveInputStream::GetNextEntry() can only return a partially populated wxArchiveEntry object - not all the fields are set.

The input stream then keeps a weak reference to the entry object and updates it when more meta-data becomes

available. A weak reference being one that does not prevent you from deleting the wxArchiveEntry object - the input stream only attempts to update it if it is still around.

The documentation for each archive entry type gives the details of what meta-data becomes available and when. For generic programming, when the worst case must be assumed, you can rely on all the fields of wxArchiveEntry being fully populated when GetNextEntry() returns, with the following exceptions:

- wxArchiveEntry::GetSize(): Guaranteed to be available after the entry has been read to wxInputStream::Eof(), or wxArchiveInputStream::CloseEntry() has been called.
- wxArchiveEntry::IsReadOnly(): Guaranteed to be available after the end of the archive has been reached, i.e. after GetNextEntry() returns NULL and Eof() is true.

This mechanism allows wxArchiveOutputStream::CopyEntry() to always fully preserve entries' meta-data. No matter what order order the meta-data occurs within the archive, the input stream will always have read it before the output stream must write it.

wxArchiveNotifier

Notifier objects can be used to get a notification whenever an input stream updates a wxArchiveEntry object's data via the weak reference mechanism.

Consider the following code which renames an entry in an archive. This is the usual way to modify an entry's meta-data, simply set the required field before writing it with wxArchiveOutputStream::CopyEntry():

```
auto_ptr<wxArchiveInputStream> arc(factory->NewStream(in));
auto_ptr<wxArchiveOutputStream> outarc(factory->NewStream(out));
auto_ptr<wxArchiveEntry> entry;

outarc->CopyArchiveMetaData(*arc);

while (entry.reset(arc->GetNextEntry()), entry.get() != NULL)
{
    if (entry->GetName() == from)
        entry->SetName(to);
    if (!outarc->CopyEntry(entry.release(), *arc))
        break;
}

bool success = arc->Eof() && outarc->Close();
```

However, for non-seekable streams, this technique cannot be used for fields such as wxArchiveEntry::IsReadOnly(), which are not necessarily set when wxArchiveInputStream::GetNextEntry() returns.

In this case a wxArchiveNotifier can be used:

```
class MyNotifier : public wxArchiveNotifier
{
public:
    void OnEntryUpdated(wxArchiveEntry& entry) { entry.
      SetIsReadOnly(false); }
};
```

The meta-data changes are done in your notifier's wxArchiveNotifier::OnEntryUpdated() method, then wxArchive↩ Entry::SetNotifier() is called before CopyEntry():

```
auto_ptr<wxArchiveInputStream> arc(factory->NewStream(in));
auto_ptr<wxArchiveOutputStream> outarc(factory->NewStream(out));
auto_ptr<wxArchiveEntry> entry;
MyNotifier notifier;

outarc->CopyArchiveMetaData(*arc);

while (entry.reset(arc->GetNextEntry()), entry.get() != NULL)
{
    entry->SetNotifier(notifier);
    if (!outarc->CopyEntry(entry.release(), *arc))
        break;
}

bool success = arc->Eof() && outarc->Close();
```

SetNotifier() calls OnEntryUpdated() immediately, then the input stream calls it again whenever it sets more fields in the entry. Since OnEntryUpdated() will be called at least once, this technique always works even when it is not strictly necessary to use it. For example, changing the entry name can be done this way too and it works on seekable streams as well as non-seekable.

10.31 Interprocess Communication

wxWidgets has a number of different classes to help with interprocess communication and network programming.

This section only discusses one family of classes – the DDE-like protocol – but here's a list of other useful classes:

- wxSocketEvent, wxSocketBase, wxSocketClient, wxSocketServer - Classes for the low-level TCP/IP API.
- wxProtocol, wxURL, wxFTP, wxHTTP - Classes for programming popular Internet protocols.

wxWidgets' DDE-like protocol is a high-level protocol based on Windows DDE. There are two implementations of this DDE-like protocol: one using real DDE running on Windows only, and another using TCP/IP (sockets) that runs on most platforms. Since the API and virtually all of the behaviour is the same apart from the names of the classes, you should find it easy to switch between the two implementations.

Notice that by including <wx/ipc.h> you may define convenient synonyms for the IPC classes: wxServer for either wxDDEServer or wxTCPServer depending on whether DDE-based or socket-based implementation is used and the same thing for wxClient and wxConnection.

By default, the DDE implementation is used under Windows. DDE works within one computer only. If you want to use IPC between different workstations you should define wxUSE_DDE_FOR_IPC as 0 before including this header – this will force using TCP/IP implementation even under Windows.

The following description refers to wxWidgets, but remember that the equivalent wxTCP∗ and wxDDE∗ classes can be used in much the same way.

Three classes are central to the DDE-like API:

- wxClient - This represents the client application, and is used only within a client program.
- wxServer - This represents the server application, and is used only within a server program.
- wxConnection - This represents the connection from the client to the server. Both the client and the server use an instance of this class, one per connection. Most DDE transactions operate on this object.

Messages between applications are usually identified by three variables: connection object, topic name and item name. A data string is a fourth element of some messages. To create a connection (a conversation in Windows parlance), the client application uses wxClient::MakeConnection to send a message to the server object, with a string service name to identify the server and a topic name to identify the topic for the duration of the connection. Under Unix, the service name may be either an integer port identifier in which case an Internet domain socket will be used for the communications or a valid file name (which shouldn't exist and will be deleted afterwards) in which case a Unix domain socket is created.

SECURITY NOTE: Using Internet domain sockets is extremely insecure for IPC as there is absolutely no access control for them, use Unix domain sockets whenever possible!

The server then responds and either vetoes the connection or allows it. If allowed, both the server and client objects create wxConnection objects which persist until the connection is closed. The connection object is then used for sending and receiving subsequent messages between client and server - overriding virtual functions in your class derived from wxConnection allows you to handle the DDE messages.

To create a working server, the programmer must:

- Derive a class from wxConnection, providing handlers for various messages sent to the server side of a wxConnection (e.g. OnExecute, OnRequest, OnPoke). Only the handlers actually required by the application need to be overridden.

- Derive a class from wxServer, overriding OnAcceptConnection to accept or reject a connection on the basis of the topic argument. This member must create and return an instance of the derived connection class if the connection is accepted.
- Create an instance of your server object and call Create to activate it, giving it a service name.

To create a working client, the programmer must:

- Derive a class from wxConnection, providing handlers for various messages sent to the client side of a wxConnection (e.g. OnAdvise). Only the handlers actually required by the application need to be overridden.
- Derive a class from wxClient, overriding OnMakeConnection to create and return an instance of the derived connection class.
- Create an instance of your client object.
- When appropriate, create a new connection using wxClient::MakeConnection, with arguments host name (processed in Unix only, use 'localhost' for local computer), service name, and topic name for this connection. The client object will call OnMakeConnection to create a connection object of the derived class if the connection is successful.
- Use the wxConnection member functions to send messages to the server.

10.31.1 Data Transfer

These are the ways that data can be transferred from one application to another. These are methods of wxConnection.

- **Execute:** the client calls the server with a data string representing a command to be executed. This succeeds or fails, depending on the server's willingness to answer. If the client wants to find the result of the Execute command other than success or failure, it has to explicitly call Request.
- **Request:** the client asks the server for a particular data string associated with a given item string. If the server is unwilling to reply, the return value is NULL. Otherwise, the return value is a string (actually a pointer to the connection buffer, so it should not be deallocated by the application).
- **Poke:** The client sends a data string associated with an item string directly to the server. This succeeds or fails.
- **Advise:** The client asks to be advised of any change in data associated with a particular item. If the server agrees, the server will send an OnAdvise message to the client along with the item and data.

The default data type is wxCF_TEXT (ASCII text), and the default data size is the length of the null-terminated string. Windows-specific data types could also be used on the PC.

10.31.2 Examples

See the sample programs *server* and *client* in the IPC samples directory. Run the server, then the client. This demonstrates using the Execute, Request, and Poke commands from the client, together with an Advise loop: selecting an item in the server list box causes that item to be highlighted in the client list box.

10.31.3 More DDE Details

A wxClient object initiates the client part of a client-server DDE-like (Dynamic Data Exchange) conversation (available in both Windows and Unix).

To create a client which can communicate with a suitable server, you need to derive a class from wxConnection and another from wxClient. The custom wxConnection class will receive communications in a 'conversation' with a

server. and the custom wxServer is required so that a user-overridden wxClient::OnMakeConnection member can return a wxConnection of the required class, when a connection is made.

For example:

```
class MyConnection: public wxConnection
{
public:
    MyConnection(void)::wxConnection() { }
    ~MyConnection(void) { }

    bool OnAdvise(const wxString& topic, const wxString& item, char *data,
                  int size, wxIPCFormat format)
    {
        wxMessageBox(topic, data);
    }
};

class MyClient: public wxClient
{
public:
    MyClient(void) { }

    wxConnectionBase* OnMakeConnection(void)
    {
        return new MyConnection;
    }
};
```

Here, *MyConnection* will respond to OnAdvise messages sent by the server by displaying a message box.

When the client application starts, it must create an instance of the derived wxClient. In the following, command line arguments are used to pass the host name (the name of the machine the server is running on) and the server name (identifying the server process). Calling wxClient::MakeConnection implicitly creates an instance of *MyConnection* if the request for a connection is accepted, and the client then requests an *Advise* loop from the server (an Advise loop is where the server calls the client when data has changed).

```
wxString server = "4242";
wxString hostName;
wxGetHostName(hostName);

// Create a new client
MyClient *client = new MyClient;
connection = (MyConnection *)client->MakeConnection(hostName, server, "IPC TEST");

if (!connection)
{
    wxMessageBox("Failed to make connection to server", "Client Demo Error");
    return NULL;
}

connection->StartAdvise("Item");
```

10.32 Device Contexts

A wxDC is a *device* context onto which graphics and text can be drawn.

The device context is intended to represent a number of output devices in a generic way, with the same API being used throughout.

Some device contexts are created temporarily in order to draw on a window. This is true of wxScreenDC, wxClientDC, wxPaintDC, and wxWindowDC. The following describes the differences between these device contexts and when you should use them.

- **wxScreenDC**. Use this to paint on the screen, as opposed to an individual window.
- **wxClientDC**. Use this to paint on the client area of window (the part without borders and other decorations), but do not use it from within an wxPaintEvent.
- **wxPaintDC**. Use this to paint on the client area of a window, but *only* from within a wxPaintEvent.

- **wxWindowDC**. Use this to paint on the whole area of a window, including decorations. This may not be available on non-Windows platforms.

To use a client, paint or window device context, create an object on the stack with the window as argument, for example:

```
void MyWindow::OnMyCmd(wxCommandEvent& event)
{
    wxClientDC dc(window);
    DrawMyPicture(dc);
}
```

Try to write code so it is parameterised by wxDC - if you do this, the same piece of code may write to a number of different devices, by passing a different device context. This doesn't work for everything (for example not all device contexts support bitmap drawing) but will work most of the time.

See also

Device Contexts

10.33 Bitmaps and Icons

The wxBitmap class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour.

Platform-specific methods for creating a wxBitmap object from an existing file are catered for, and this is an occasion where conditional compilation will sometimes be required.

A bitmap created dynamically or loaded from a file can be selected into a memory device context (instance of wxMemoryDC). This enables the bitmap to be copied to a window or memory device context using wxDC::Blit(), or to be used as a drawing surface.

See wxMemoryDC for an example of drawing onto a bitmap.

All wxWidgets platforms support XPMs for small bitmaps and icons. You may include the XPM inline as below, since it's C code, or you can load it at run-time.

```
#include "sample.xpm"
```

Sometimes you wish to use a .ico resource on Windows, and XPMs on other platforms (for example to take advantage of Windows' support for multiple icon resolutions).

A macro, wxICON(), is available which creates an icon using an XPM on the appropriate platform, or an icon resource on Windows:

```
wxIcon icon(wxICON(sample));

// The above line is equivalent to this:

#if defined(__WXGTK__) || defined(__WXMOTIF__)
    wxIcon icon(sample_xpm);
#endif

#if defined(__WXMSW__)
    wxIcon icon("sample");
#endif
```

There is also a corresponding wxBITMAP() macro which allows to create the bitmaps in much the same way as wxICON() creates icons. It assumes that bitmaps live in resources under Windows and XPM files under all other platforms (for XPMs, the corresponding file must be included before this macro is used, of course, and the name of the bitmap should be the same as the resource name under Windows with _xpm suffix). For example:

```
// an easy and portable way to create a bitmap
wxBitmap bmp(wxBITMAP(bmpname));

// which is roughly equivalent to the following
```

```
#if defined(__WXMSW__)
    wxBitmap bmp("bmpname", wxBITMAP_TYPE_BMP_RESOURCE);
#else // Unix
    wxBitmap bmp(bmpname_xpm, wxBITMAP_TYPE_XPM);
#endif
```

You should always use wxICON() and wxBITMAP() macros because they work for any platform (unlike the code above which doesn't deal with wxMac, wxX11, ...) and are shorter and more clear than versions with many #ifdef blocks. Alternatively, you could use the same XPMs on all platforms and avoid dealing with Windows resource files.

If you'd like to embed bitmaps with alpha transparency in your program, neither XPM nor BMP formats are appropriate as they don't have support for alpha and another format, typically PNG, should be used. wxWidgets provides a similar helper for PNG bitmaps called wxBITMAP_PNG() that can be used to either load PNG files embedded in resources (meaning either Windows resource section of the executable file or OS X "Resource" subdirectory of the application bundle) or arrays containing PNG data included into the program code itself.

See also

Graphics Device Interface (GDI)

10.33.1 Supported Bitmap File Formats

The following lists the formats handled on different platforms. Note that missing or partially-implemented formats are automatically supplemented by using wxImage to load the data, and then converting it to wxBitmap form. Note that using wxImage is the preferred way to load images in wxWidgets, with the exception of resources (XPM-files or native Windows resources).

Writing an image format handler for wxImage is also far easier than writing one for wxBitmap, because wxImage has exactly one format on all platforms whereas wxBitmap can store pixel data very differently, depending on colour depths and platform.

wxBitmap

Under Windows, wxBitmap may load the following formats:

```
@li Windows bitmap resource (wxBITMAP_TYPE_BMP_RESOURCE)
@li Windows bitmap file (wxBITMAP_TYPE_BMP)
@li XPM data and file (wxBITMAP_TYPE_XPM)
@li All formats that are supported by the wxImage class.
```

Under wxGTK, wxBitmap may load the following formats:

```
@li XPM data and file (wxBITMAP_TYPE_XPM)
@li All formats that are supported by the wxImage class.
```

Under wxMotif and wxX11, wxBitmap may load the following formats:

```
@li XBM data and file (wxBITMAP_TYPE_XBM)
@li XPM data and file (wxBITMAP_TYPE_XPM)
@li All formats that are supported by the wxImage class.
```

wxIcon

Under Windows, wxIcon may load the following formats:

```
@li Windows icon resource (wxBITMAP_TYPE_ICO_RESOURCE)
@li Windows icon file (wxBITMAP_TYPE_ICO)
@li XPM data and file (wxBITMAP_TYPE_XPM)
```

Under wxGTK, wxIcon may load the following formats:

```
@li XPM data and file (wxBITMAP_TYPE_XPM)
@li All formats that are supported by the wxImage class.
```

Under wxMotif and wxX11, wxIcon may load the following formats:

```
@li XBM data and file (wxBITMAP_TYPE_XBM)
@li XPM data and file (wxBITMAP_TYPE_XPM)
@li All formats that are supported by the wxImage class.
```

wxCursor

Under Windows, wxCursor may load the following formats:

```
@li Windows cursor resource (wxBITMAP_TYPE_CUR_RESOURCE)
@li Windows cursor file (wxBITMAP_TYPE_CUR)
@li Windows icon file (wxBITMAP_TYPE_ICO)
@li Windows bitmap file (wxBITMAP_TYPE_BMP)
```

Under wxGTK, wxCursor may load the following formats (in addition to stock cursors):

```
@li None (stock cursors only).
```

Under wxMotif and wxX11, wxCursor may load the following formats:

```
@li XBM data and file (wxBITMAP_TYPE_XBM)
```

10.33.2 Bitmap Format Handlers

To provide extensibility, the functionality for loading and saving bitmap formats is not implemented in the wxBitmap class, but in a number of handler classes, derived from wxBitmapHandler. There is a static list of handlers which wxBitmap examines when a file load/save operation is requested.

Some handlers are provided as standard, but if you have special requirements, you may wish to initialise the wxBitmap class with some extra handlers which you write yourself or receive from a third party.

To add a handler object to wxBitmap, your application needs to include the header which implements it, and then call the static function wxBitmap::AddHandler().

Note

Bitmap handlers are not implemented on all platforms, and new ones rarely need to be implemented since wxImage can be used for loading most formats, as noted earlier.

10.34 wxFont Overview

A font is an object which determines the appearance of text, primarily when drawing text to a window or device context.

A font is determined by the following parameters (not all of them have to be specified, of course):

Point size	This is the standard way of referring to text size.
Family	Supported families are: **wxDEFAULT**, **wxDECORATIVE**, **wxROMAN**, **wxSCRIPT**, **wxSWISS**, **wxMODERN**. **wxMODERN** is a fixed pitch font; the others are either fixed or variable pitch.
Style	The value can be **wxNORMAL**, **wxSLANT** or **wxITALIC**.
Weight	The value can be **wxNORMAL**, **wxLIGHT** or **wxBOLD**.
Underlining	The value can be true or false.
Face name	An optional string specifying the actual typeface to be used. If NULL, a default typeface will chosen based on the family.
Encoding	The font encoding (see **wxFONTENCODING_XXX** constants and the Font Encodings for more details)

Specifying a family, rather than a specific typeface name, ensures a degree of portability across platforms because a suitable font will be chosen for the given font family, however it doesn't allow to choose a font precisely as the parameters above don't suffice, in general, to identify all the available fonts and this is where using the native font descriptions may be helpful - see below.

Under Windows, the face name can be one of the installed fonts on the user's system. Since the choice of fonts differs from system to system, either choose standard Windows fonts, or if allowing the user to specify a face name, store the family name with any file that might be transported to a different Windows machine or other platform.

See also

wxFont, wxFontDialog

Note

There is currently a difference between the appearance of fonts on the two platforms, if the mapping mode is anything other than wxMM_TEXT. Under X, font size is always specified in points. Under MS Windows, the unit for text is points but the text is scaled according to the current mapping mode. However, user scaling on a device context will also scale fonts under both environments.

10.34.1 Native Font Information

An alternative way of choosing fonts is to use the native font description. This is the only acceptable solution if the user is allowed to choose the font using the wxFontDialog because the selected font cannot be described using only the family name and so, if only family name is stored permanently, the user would almost surely see a different font in the program later.

Instead, you should store the value returned by wxFont::GetNativeFontInfoDesc and pass it to wxFont::SetNativeFontInfo later to recreate exactly the same font.

Note that the contents of this string depends on the platform and shouldn't be used for any other purpose (in particular, it is not meant to be shown to the user). Also please note that although the native font information is currently implemented for Windows and Unix (GTK+ and Motif) ports only, all the methods are available for all the ports and should be used to make your program work correctly when they are implemented later.

10.35 Font Encodings

wxWidgets has support for multiple font encodings.

By encoding we mean here the mapping between the character codes and the letters. Probably the most well-known encoding is (7 bit) ASCII one which is used almost universally now to represent the letters of the English alphabet and some other common characters. However, it is not enough to represent the letters of foreign alphabets and

here other encodings come into play. Please note that we will only discuss 8-bit fonts here and not Unicode (see Unicode Support in wxWidgets).

Font encoding support is ensured by several classes: wxFont itself, but also wxFontEnumerator and wxFontMapper. wxFont encoding support is reflected by a (new) constructor parameter *encoding* which takes one of the following values (elements of enumeration type `wxFontEncoding`):

wxFONTENCODING_SYSTEM	The default encoding of the underlying operating system (notice that this might be a "foreign" encoding for foreign versions of Windows 9x/NT).
wxFONTENCODING_DEFAULT	The applications default encoding as returned by wxFont::GetDefaultEncoding. On program startup, the applications default encoding is the same as wxFONTENCODING_SYSTEM, but may be changed to make all the fonts created later to use it (by default).
wxFONTENCODING_ISO8859_1..15	ISO8859 family encodings which are usually used by all non-Microsoft operating systems.
wxFONTENCODING_KOI8	Standard Cyrillic encoding for the Internet (but see also wxFONTENCODING_ISO8859_5 and wxFONTENCODING_CP1251).
wxFONTENCODING_CP1250	Microsoft analogue of ISO8859-2
wxFONTENCODING_CP1251	Microsoft analogue of ISO8859-5
wxFONTENCODING_CP1252	Microsoft analogue of ISO8859-1

As you may see, Microsoft's encoding partly mirror the standard ISO8859 ones, but there are (minor) differences even between ISO8859-1 (Latin1, ISO encoding for Western Europe) and CP1251 (WinLatin1, standard code page for English versions of Windows) and there are more of them for other encodings.

The situation is particularly complicated with Cyrillic encodings for which (more than) three incompatible encodings exist: KOI8 (the old standard, widely used on the Internet), ISO8859-5 (ISO standard for Cyrillic) and CP1251 (WinCyrillic).

This abundance of (incompatible) encodings should make it clear that using encodings is less easy than it might seem. The problems arise both from the fact that the standard encodings for the given language (say Russian, which is written in Cyrillic) are different on different platforms and because the fonts in the given encoding might just not be installed (this is especially a problem with Unix, or, in general, non-Win32 systems).

To clarify, the wxFontEnumerator class may be used to enumerate both all available encodings and to find the facename(s) in which the given encoding exists. If you can find the font in the correct encoding with wxFontEnumerator then your troubles are over, but, unfortunately, sometimes this is not enough. For example, there is no standard way (that I know of, please tell me if you do!) to find a font on a Windows system for KOI8 encoding (only for WinCyrillic one which is quite different), so wxFontEnumerator will never return one, even if the user has installed a KOI8 font on his system.

To solve this problem, a wxFontMapper class is provided.

This class stores the mapping between the encodings and the font face names which support them in wxConfigBase object. Of course, it would be fairly useless if it tried to determine these mappings by itself, so, instead, it (optionally) asks the user and remembers his answers so that the next time the program will automatically choose the correct font. All these topics are illustrated by the Font Sample; please refer to it and the documentation of the classes mentioned here for further explanations.

10.36 Printing Framework Overview

The printing framework relies on the application to provide classes whose member functions can respond to particular requests, such as 'print this page' or 'does this page exist in the document?'.

This method allows wxWidgets to take over the housekeeping duties of turning preview pages, calling the print dialog box, creating the printer device context, and so on: the application can concentrate on the rendering of the information onto a device context.

In most cases, the only class you will need to derive from is wxPrintout; all others will be used as-is.

A brief description of each class's role and how they work together follows.

For the special case of printing under Unix, where various different printing backends have to be offered, please have a look at Printing Under Unix (GTK+).

See also

Printing Framework

10.36.1 wxPrintout

A document's printing ability is represented in an application by a derived wxPrintout class. This class prints a page on request, and can be passed to the Print function of a wxPrinter object to actually print the document, or can be passed to a wxPrintPreview object to initiate previewing. The following code (from the printing sample) shows how easy it is to initiate printing, previewing and the print setup dialog, once the wxPrintout functionality has been defined. Notice the use of MyPrintout for both printing and previewing. All the preview user interface functionality is taken care of by wxWidgets. For more details on how MyPrintout is defined, please look at the printout sample code.

```
case WXPRINT_PRINT:
{
    wxPrinter printer;
    MyPrintout printout("My printout");
    printer.Print(this, &printout, true);
    break;
}
case WXPRINT_PREVIEW:
{
    // Pass two printout objects: for preview, and possible printing.
    wxPrintPreview *preview = new wxPrintPreview(new MyPrintout, new MyPrintout
      );
    wxPreviewFrame *frame = new wxPreviewFrame(preview, this,
                                               "Demo Print Preview",
                                               wxPoint(100, 100),
                                               wxSize(600, 650));
    frame->Centre(wxBOTH);
    frame->Initialize();
    frame->Show(true);
    break;
}
```

wxPrintout assembles the printed page and (using your subclass's overrides) writes requested pages to a wxDC that is passed to it. This wxDC could be a wxMemoryDC (for displaying the preview image on-screen), a wxPrinterDC (for printing under MSW and Mac), or a wxPostScriptDC (for printing under GTK or generating PostScript output).

The document/view framework creates a default wxPrintout object for every view, calling wxView::OnDraw() to achieve a prepackaged print/preview facility.

If your window classes have a Draw(wxDC *dc) routine to do screen rendering, your wxPrintout subclass will typically call those routines to create portions of the image on your printout. Your wxPrintout subclass can also make its own calls to its wxDC to draw headers, footers, page numbers, etc.

The scaling of the drawn image typically differs from the screen to the preview and printed images. This class provides a set of routines named FitThisSizeToXXX(), MapScreenSizeToXXX(), and GetLogicalXXXRect, which can be used to set the user scale and origin of the wxPrintout's DC so that your class can easily map your image to the printout without getting into the details of screen and printer PPI and scaling. See the printing sample for examples of how these routines are used.

10.36.2 wxPrinter

Class wxPrinter encapsulates the platform-dependent print function with a common interface. In most cases, you will not need to derive a class from wxPrinter; simply create a wxPrinter object in your Print function as in the example above.

10.36.3 wxPrintPreview

Class wxPrintPreview manages the print preview process. Among other things, it constructs the wxDCs that get passed to your wxPrintout subclass for printing and manages the display of multiple pages, a zoomable preview image, and so forth. In most cases you will use this class as-is, but you can create your own subclass, for example, to change the layout or contents of the preview window.

10.36.4 wxPrinterDC

Class wxPrinterDC is the wxDC that represents the actual printed page under MSW and Mac. During printing, an object of this class will be passed to your derived wxPrintout object to draw upon. The size of the wxPrinterDC will depend on the paper orientation and the resolution of the printer.

There are two important rectangles in printing: the *page rectangle* defines the printable area seen by the application, and under MSW and Mac, it is the printable area specified by the printer. (For PostScript printing, the page rectangle is the entire page.) The inherited function wxDC::GetSize() returns the page size in device pixels. The point (0,0) on the wxPrinterDC represents the top left corner of the page rectangle; that is, the page rect is given by wxRect(0, 0, w, h), where (w,h) are the values returned by GetSize.

The *paper rectangle*, on the other hand, represents the entire paper area including the non-printable border. Thus, the coordinates of the top left corner of the paper rectangle will have small negative values, while the width and height will be somewhat larger than that of the page rectangle. The wxPrinterDC-specific function wxPrinterDC::GetPaperRect() returns the paper rectangle of the given wxPrinterDC.

10.36.5 wxPostScriptDC

Class wxPostScriptDC is the wxDC that represents the actual printed page under GTK and other PostScript printing. During printing, an object of this class will be passed to your derived wxPrintout object to draw upon. The size of the wxPostScriptDC will depend upon the wxPrintData used to construct it.

Unlike a wxPrinterDC, there is no distinction between the page rectangle and the paper rectangle in a wxPostScriptDC; both rectangles are taken to represent the entire sheet of paper.

10.36.6 wxPrintDialog

Class wxPrintDialog puts up the standard print dialog, which allows you to select the page range for printing (as well as many other print settings, which may vary from platform to platform). You provide an object of type wxPrintDialogData to the wxPrintDialog at construction, which is used to populate the dialog.

10.36.7 wxPrintData

Class wxPrintData is a subset of wxPrintDialogData that is used (internally) to initialize a wxPrinterDC or wxPostScriptDC. (In fact, a wxPrintData is a data member of a wxPrintDialogData and a wxPageSetupDialogData). Essentially, wxPrintData contains those bits of information from the two dialogs necessary to configure the wxPrinterDC or wxPostScriptDC (e.g., size, orientation, etc.). You might wish to create a global instance of this object to provide call-to-call persistence to your application's print settings.

10.36.8 wxPrintDialogData

Class wxPrintDialogData contains the settings entered by the user in the print dialog. It contains such things as page range, number of copies, and so forth. In most cases, you won't need to access this information; the framework takes care of asking your wxPrintout derived object for the pages requested by the user.

10.36.9 wxPageSetupDialog

Class wxPageSetupDialog puts up the standard page setup dialog, which allows you to specify the orientation, paper size, and related settings. You provide it with a wxPageSetupDialogData object at initialization, which is used to populate the dialog; when the dialog is dismissed, this object contains the settings chosen by the user, including orientation and/or page margins.

Note that on Macintosh, the native page setup dialog does not contain entries that allow you to change the page margins. You can use the Mac-specific class wxMacPageMarginsDialog (which, like wxPageSetupDialog, takes a wxPageSetupDialogData object in its constructor) to provide this capability; see the printing sample for an example.

10.36.10 wxPageSetupDialogData

Class wxPageSetupDialogData contains settings affecting the page size (paper size), orientation, margins, and so forth. Note that not all platforms populate all fields; for example, the MSW page setup dialog lets you set the page margins while the Mac setup dialog does not.

You will typically create a global instance of each of a wxPrintData and wxPageSetupDialogData at program initiation, which will contain the default settings provided by the system. Each time the user calls up either the wxPrintDialog or the wxPageSetupDialog, you pass these data structures to initialize the dialog values and to be updated by the dialog. The framework then queries these data structures to get information like the printed page range (from the wxPrintDialogData) or the paper size and/or page orientation (from the wxPageSetupDialogData).

10.37 Printing Under Unix (GTK+)

Printing under Unix has always been a cause of problems as Unix does not provide a standard way to display text and graphics on screen and print it to a printer using the same application programming interface - instead, displaying on screen is done via the X11 library while printing has to be done with using PostScript commands.

This was particularly difficult to handle for the case of fonts with the result that only a selected number of application could offer WYSIWYG under Unix. Equally, wxWidgets offered its own printing implementation using PostScript which never really matched the screen display.

Since GTK+ 2.10, support for printing has been added to GTK+ itself and beginning with wxWidgets 2.9, GTK+ printing is used by default (i.e. unless `-without-gtkprint` was explicitly used when configuring the library). Support for GTK+ print is detected dynamically, i.e. during the run-time: if it is found, printing will be done through GTK+, otherwise the application will fall back to the old PostScript printing code. This allows the applications built with wxWidgets to still work on the very old systems using GTK+ earlier than 2.10.

10.38 Sizers Overview

Sizers, as represented by the wxSizer class and its descendants in the wxWidgets class hierarchy, have become the method of choice to define the layout of controls in dialogs in wxWidgets because of their ability to create visually appealing dialogs independent of the platform, taking into account the differences in size and style of the individual controls.

Unlike the original wxWidgets Dialog Editor, editors such as wxDesigner, DialogBlocks, XRCed and wxWorkshop create dialogs based exclusively on sizers, practically forcing the user to create platform independent layouts without compromises.

The next section describes and shows what can be done with sizers. The following sections briefly describe how to program with individual sizer classes.

For information about the wxWidgets resource system, which can describe sizer-based dialogs, see the XML Based Resource System (XRC).

See also

wxSizer, wxBoxSizer, wxStaticBoxSizer, wxGridSizer, wxFlexGridSizer, wxGridBagSizer

10.38.1 The Idea Behind Sizers

The layout algorithm used by sizers in wxWidgets is closely related to layout systems in other GUI toolkits, such as Java's AWT, the GTK toolkit or the Qt toolkit. It is based upon the idea of individual subwindows reporting their minimal required size and their ability to get stretched if the size of the parent window has changed. This will most often mean that the programmer does not set the start-up size of a dialog, the dialog will rather be assigned a sizer and this sizer will be queried about the recommended size. This sizer in turn will query its children (which can be normal windows, empty space or other sizers) so that a hierarchy of sizers can be constructed. Note that wxSizer does not derive from wxWindow and thus does not interfere with tab ordering and requires very few resources compared to a real window on screen.

What makes sizers so well fitted for use in wxWidgets is the fact that every control reports its own minimal size and the algorithm can handle differences in font sizes or different window (dialog item) sizes on different platforms without problems. For example, if the standard font as well as the overall design of Linux/GTK widgets requires more space than on Windows, the initial dialog size will automatically be bigger on Linux/GTK than on Windows.

There are currently five different kinds of sizers available in wxWidgets. Each represents either a certain way to lay out dialog items in a dialog or it fulfills a special task such as wrapping a static box around a dialog item (or another sizer). These sizers will be discussed one by one in the text below. For more detailed information on how to use sizers programmatically, please refer to the section Programming with wxBoxSizer.

10.38.2 Common Features

All sizers are containers, that is, they are used to lay out one dialog item (or several dialog items), which they contain. Such items are sometimes referred to as the children of the sizer. Independent of how the individual sizers lay out their children, all children have certain features in common:

A minimal size: This minimal size is usually identical to the initial size of the controls and may either be set explicitly in the wxSize field of the control constructor or may be calculated by wxWidgets, typically by setting the height and/or the width of the item to -1. Note that only some controls can calculate their size (such as a checkbox) whereas others (such as a listbox) don't have any natural width or height and thus require an explicit size. Some controls can calculate their height, but not their width (e.g. a single line text control):

A border: The border is just empty space and is used to separate dialog items in a dialog. This border can either be all around, or at any combination of sides such as only above and below the control. The thickness of this border must be set explicitly, typically 5 points. The following samples show dialogs with only one dialog item (a button) and a border of 0, 5, and 10 pixels around the button:

An alignment: Often, a dialog item is given more space than its minimal size plus its border. Depending on what flags are used for the respective dialog item, the dialog item can be made to fill out the available space entirely, i.e. it will grow to a size larger than the minimal size, or it will be moved to either the centre of the available space or to either side of the space. The following sample shows a listbox and three buttons in a horizontal box sizer; one button is centred, one is aligned at the top, one is aligned at the bottom:

A stretch factor: If a sizer contains more than one child and it is offered more space than its children and their borders need, the question arises how to distribute the surplus space among the children. For this purpose, a stretch factor may be assigned to each child, where the default value of 0 indicates that the child will not get more space than its requested minimum size. A value of more than zero is interpreted in relation to the sum of all stretch factors in the children of the respective sizer, i.e. if two children get a stretch factor of 1, they will get half the extra space each *independent of whether one control has a minimal sizer inferior to the other or not*. The following sample shows a dialog with three buttons, the first one has a stretch factor of 1 and thus gets stretched, whereas the other two buttons have a stretch factor of zero and keep their initial width:

Within wxDesigner, this stretch factor gets set from the *Option* menu.

10.38.3 Hiding Controls Using Sizers

You can hide controls contained in sizers the same way you would hide any control, using the wxWindow::Show method. However, wxSizer also offers a separate method which can tell the sizer not to consider that control in its size calculations. To hide a window using the sizer, call wxSizer::Show. You must then call Layout on the sizer to force an update.

This is useful when hiding parts of the interface, since you can avoid removing the controls from the sizer and having to add them back later.

Note

This is supported only by wxBoxSizer and wxFlexGridSizer.

wxBoxSizer

wxBoxSizer can lay out its children either vertically or horizontally, depending on what flag is being used in its constructor. When using a vertical sizer, each child can be centered, aligned to the right or aligned to the left. Correspondingly, when using a horizontal sizer, each child can be centered, aligned at the bottom or aligned at the top. The stretch factor described in the last paragraph is used for the main orientation, i.e. when using a horizontal box sizer, the stretch factor determines how much the child can be stretched horizontally. The following sample shows the same dialog as in the last sample, only the box sizer is a vertical box sizer now:

wxStaticBoxSizer

wxStaticBoxSixer is the same as a wxBoxSizer, but surrounded by a static box. Here is a sample:

wxGridSizer

wxGridSizer is a two-dimensional sizer. All children are given the same size, which is the minimal size required by the biggest child, in this case the text control in the left bottom border. Either the number of columns or the number or rows is fixed and the grid sizer will grow in the respectively other orientation if new children are added:

For programming information, see wxGridSizer.

wxFlexGridSizer

Another two-dimensional sizer derived from wxGridSizer. The width of each column and the height of each row are calculated individually according to the minimal requirements from the respectively biggest child. Additionally, columns and rows can be declared to be stretchable if the sizer is assigned a size different from the one it requested. The following sample shows the same dialog as the one above, but using a flex grid sizer:

10.38.4 Programming with wxBoxSizer

The basic idea behind a wxBoxSizer is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.

As an example, we will construct a dialog that will contain a text field at the top and two buttons at the bottom. This can be seen as a top-hierarchy column with the text at the top and buttons at the bottom and a low-hierarchy row with an OK button to the left and a Cancel button to the right. In many cases (particularly dialogs under Unix and normal frames) the main window will be resizable by the user and this change of size will have to get propagated to its children. In our case, we want the text area to grow with the dialog, whereas the button shall have a fixed size. In addition, there will be a thin border around all controls to make the dialog look nice and - to make matter worse - the buttons shall be centred as the width of the dialog changes.

It is the unique feature of a box sizer, that it can grow in both directions (height and width) but can distribute its growth in the main direction (horizontal for a row) *unevenly* among its children. In our example case, the vertical

sizer is supposed to propagate all its height changes to only the text area, not to the button area. This is determined by the *proportion* parameter when adding a window (or another sizer) to a sizer. It is interpreted as a weight factor, i.e. it can be zero, indicating that the window may not be resized at all, or above zero. If several windows have a value above zero, the value is interpreted relative to the sum of all weight factors of the sizer, so when adding two windows with a value of 1, they will both get resized equally much and each half as much as the sizer owning them. Then what do we do when a column sizer changes its width? This behaviour is controlled by *flags* (the second parameter of the Add() function): Zero or no flag indicates that the window will preserve it is original size, wxGR↩ OW flag (same as wxEXPAND) forces the window to grow with the sizer, and wxSHAPED flag tells the window to change it is size proportionally, preserving original aspect ratio. When wxGROW flag is not used, the item can be aligned within available space. wxALIGN_LEFT, wxALIGN_TOP, wxALIGN_RIGHT, wxALIGN_BOTTOM, wxALIG↩ N_CENTER_HORIZONTAL and wxALIGN_CENTER_VERTICAL do what they say. wxALIGN_CENTRE (same as wxALIGN_CENTER) is defined as (wxALIGN_CENTER_HORIZONTAL | wxALIGN_CENTER_VERTICAL). Default alignment is wxALIGN_LEFT | wxALIGN_TOP.

As mentioned above, any window belonging to a sizer may have a border, and it can be specified which of the four sides may have this border, using the wxTOP, wxLEFT, wxRIGHT and wxBOTTOM constants or wxALL for all directions (and you may also use wxNORTH, wxWEST etc instead). These flags can be used in combination with the alignment flags above as the second parameter of the Add() method using the binary or operator |. The sizer of the border also must be made known, and it is the third parameter in the Add() method. This means, that the entire behaviour of a sizer and its children can be controlled by the three parameters of the Add() method.

```
// We want to get a dialog that is stretchable because it
// has a text ctrl at the top and two buttons at the bottom.

MyDialog::MyDialog(wxFrame *parent, wxWindowID id, const
      wxString &title )
: wxDialog(parent, id, title, wxDefaultPosition,
      wxDefaultSize,
           wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
    wxBoxSizer *topsizer = new wxBoxSizer( wxVERTICAL );

    // create text ctrl with minimal size 100x60
    topsizer->Add(
        new wxTextCtrl( this, -1, "My text.", wxDefaultPosition,
      wxSize(100,60), wxTE_MULTILINE),
        1,            // make vertically stretchable
        wxEXPAND |    // make horizontally stretchable
        wxALL,        //   and make border all around
        10 );         // set border width to 10

    wxBoxSizer *button_sizer = new wxBoxSizer( wxHORIZONTAL );
    button_sizer->Add(
        new wxButton( this, wxID_OK, "OK" ),
        0,            // make horizontally unstretchable
        wxALL,        // make border all around (implicit top alignment)
        10 );         // set border width to 10
    button_sizer->Add(
        new wxButton( this, wxID_CANCEL, "Cancel" ),
        0,            // make horizontally unstretchable
        wxALL,        // make border all around (implicit top alignment)
        10 );         // set border width to 10

    topsizer->Add(
        button_sizer,
        0,                // make vertically unstretchable
        wxALIGN_CENTER ); // no border and centre horizontally

    SetSizerAndFit(topsizer); // use the sizer for layout and size window
                              // accordingly and prevent it from being resized
                              // to smaller size
}
```

Note that the new way of specifying flags to wxSizer is via wxSizerFlags. This class greatly eases the burden of passing flags to a wxSizer.

Here's how you'd do the previous example with wxSizerFlags:

```
// We want to get a dialog that is stretchable because it
// has a text ctrl at the top and two buttons at the bottom.

MyDialog::MyDialog(wxFrame *parent, wxWindowID id, const
      wxString &title )
: wxDialog(parent, id, title, wxDefaultPosition,
      wxDefaultSize,
```

```
            wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER)
{
    wxBoxSizer *topsizer = new wxBoxSizer( wxVERTICAL );

    // create text ctrl with minimal size 100x60 that is horizontally and
    // vertically stretchable with a border width of 10
    topsizer->Add(
        new wxTextCtrl( this, -1, "My text.", wxDefaultPosition,
      wxSize(100,60), wxTE_MULTILINE),
        wxSizerFlags(1).Align().Expand().Border(wxALL, 10));

    wxBoxSizer *button_sizer = new wxBoxSizer( wxHORIZONTAL );

    //create two buttons that are horizontally unstretchable,
    // with an all-around border with a width of 10 and implicit top alignment
    button_sizer->Add(
        new wxButton( this, wxID_OK, "OK" ),
        wxSizerFlags(0).Align().Border(wxALL, 10));

    button_sizer->Add(
        new wxButton( this, wxID_CANCEL, "Cancel" ),
        wxSizerFlags(0).Align().Border(wxALL, 10));

    //create a sizer with no border and centered horizontally
    topsizer->Add(
        button_sizer,
        wxSizerFlags(0).Center() );

    SetSizerAndFit(topsizer); // use the sizer for layout and set size and hints
}
```

10.38.5 Other Types of Sizers

wxGridSizer is a sizer which lays out its children in a two-dimensional table with all table fields having the same size, i.e. the width of each field is the width of the widest child, the height of each field is the height of the tallest child.

wxFlexGridSizer is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the wxGridSizer.

wxStaticBoxSizer is a sizer derived from wxBoxSizer but adds a static box around the sizer. Note that this static box has to be created separately.

wxGridBagSizer is a rather special kind of sizer which, unlike the other classes, allows to directly put the elements at the given position in the sizer. Please see its documentation for more details.

10.38.6 CreateButtonSizer

As a convenience, wxDialog::CreateButtonSizer(long flags) can be used to create a standard button sizer in which standard buttons are displayed. The following flags can be passed to this function:

```
wxYES_NO     // Add Yes/No subpanel
wxYES        // return wxID_YES
wxNO         // return wxID_NO
wxNO_DEFAULT // make the wxNO button the default,
             // otherwise wxYES or wxOK button will be default

wxOK     // return wxID_OK
wxCANCEL // return wxID_CANCEL
wxHELP   // return wxID_HELP

wxFORWARD   // return wxID_FORWARD
wxBACKWARD  // return wxID_BACKWARD
wxSETUP     // return wxID_SETUP
wxMORE      // return wxID_MORE
```

10.39 XML Based Resource System (XRC)

The XML-based resource system, known as XRC, allows user interface elements such as dialogs, menu bars and toolbars, to be stored in text files and loaded into the application at run-time.

XRC files can also be compiled into binary XRS files or C++ code (the former makes it possible to store all resources in a single file and the latter is useful when you want to embed the resources into the executable).

There are several advantages to using XRC resources:

- Recompiling and linking an application is not necessary if the resources change.
- If you use a dialog designer that generates C++ code, it can be hard to reintegrate this into existing C++ code. Separation of resources and code is a more elegant solution.
- You can choose between different alternative resource files at run time, if necessary.
- The XRC format uses sizers for flexibility, allowing dialogs to be resizable and highly portable.
- The XRC format is a wxWidgets standard, and can be generated or postprocessed by any program that understands it. As it is based on the XML standard, existing XML editors can be used for simple editing purposes.

XRC was written by Vaclav Slavik.

See also

wxXmlResource, wxXmlResourceHandler, XRC File Format

10.39.1 Getting Started with XRC

Creating an XRC file

You will need to write an XRC file. Though this *can* be done by hand in a text editor, for all but the smallest files it is advisable to use a specialised tool. Examples of these include:

Non-free:

- DialogBlocks http://www.anthemion.co.uk/dialogblocks/, a commercial dialog editor.

Free:

- XRCed http://xrced.sf.net/, a wxPython-based dialog editor that you can find in the wx↩ Python/tools subdirectory of the wxWidgets SVN archive.
- wxFormBuilder http://wxformbuilder.org/, a C++-based form designer that can output C++, XRC or python.
- wxCrafter (free version) http://www.codelite.org/wxcrafter/, a C++-based form designer that can output C++ or XRC.

There's a more complete list at http://www.wxwidgets.org/wiki/index.php/Tools

This small demonstration XRC file contains a simple dialog:

```
<?xml version="1.0" ?>
<resource version="2.3.0.1">
  <object class="wxDialog" name="SimpleDialog">
    <title>Simple dialog</title>
    <object class="wxBoxSizer">
      <orient>wxVERTICAL</orient>
      <object class="sizeritem">
        <object class="wxTextCtrl" name="text"/>
        <option>1</option>
        <flag>wxALL|wxEXPAND</flag>
        <border>10</border>
      </object>
      <object class="sizeritem">
        <object class="wxBoxSizer">
          <object class="sizeritem">
            <object class="wxButton" name="clickme_btn">
              <label>Click</label>
```

```
            </object>
            <flag>wxRIGHT</flag>
            <border>10</border>
          </object>
          <object class="sizeritem">
            <object class="wxButton" name="wxID_OK">
              <label>OK</label>
            </object>
            <flag>wxLEFT</flag>
            <border>10</border>
          </object>
          <orient>wxHORIZONTAL</orient>
        </object>
        <flag>wxALL|wxALIGN_CENTRE</flag>
        <border>10</border>
      </object>
    </object>
  </object>
</resource>
```

You can keep all your XRC elements together in one file, or split them between several.

Loading XRC files

Before you can use XRC in an app, it must first be loaded. This code fragment shows how to load a single XRC file "resource.xrc" from the current working directory, plus all the ∗.xrc files contained in the subdirectory "rc".

```
#include "wx/xrc/xmlres.h"

bool MyApp::OnInit()
{
    ...
    wxXmlResource::Get()->InitAllHandlers();

    wxXmlResource::Get()->Load("resource.xrc");
    wxXmlResource::Get()->LoadAllFiles("rc");
    ...
}
```

It's normal to load any XRC files at the beginning of an app. Though it is possible to unload a file later, it's seldom necessary.

Using an XRC item

The XRC file(s) are now loaded into the app's virtual filesystem. From there, you must do another sort of load when you want to use an individual object. Yes, it's confusingly named, but you first Load() the file, and later load each top-level object when its needed.

This is how you would use the above simple dialog in your code.

```
void MyClass::ShowDialog()
{
    wxDialog dlg;
    if (wxXmlResource::Get()->LoadDialog(&dlg, NULL, "SimpleDialog"))
        dlg.ShowModal();
}
```

See how simple the code is. All the instantiation is done invisibly by the XRC system.

Though you'll most often use wxXmlResource::LoadDialog, there are also equivalents that load a frame, a menu etc; and the generic wxXmlResource::LoadObject. See wxXmlResource for more details.

Accessing XRC child controls

The last section showed how to load top-level windows like dialogs, but what about child windows like the wxTextCtrl named "text" that the dialog contains? You can't 'load' an individual child control in the same way. Instead you use the XRCCTRL macro to get a pointer to the child. To expand the previous code:

```
void MyClass::ShowDialog()
{
    wxDialog dlg;
    if (!wxXmlResource::Get()->LoadDialog(&dlg, NULL, "SimpleDialog"))
        return;

    wxTextCtrl* pText = XRCCTRL(dlg, "text", wxTextCtrl);
```

```
    if (pText)
        pText->ChangeValue("This is a simple dialog");

    dlg.ShowModal();
}
```

XRCCTRL takes a reference to the parent container and uses wxWindow::FindWindow to search inside it for a wxWindow with the supplied name (here "text"). It returns a pointer to that control, cast to the type in the third parameter; so a similar effect could be obtained by writing:

```
pText = (wxTextCtrl*)(dlg.FindWindowByName("text"));
```

XRC and IDs

The ID of a control is often needed, e.g. for use in an event table or with wxEvtHandler::Bind. It can easily be found by passing the name of the control to the XRCID macro:

```
void MyClass::ShowDialog()
{
    wxDialog dlg;
    if (!wxXmlResource::Get()->LoadDialog(&dlg, NULL, "SimpleDialog"))
        return;

    XRCCTRL(dlg, "text", wxTextCtrl)->Bind(wxEVT_COMMAND_TEXT_UPDATED,
        wxTextEventHandler(MyClass::OnTextEntered), this, XRCID("text"));

    XRCCTRL(dlg, "clickme_btn", wxButton)->Bind(wxEVT_COMMAND_BUTTON_CLICKED,
        wxCommandEventHandler(MyClass::OnClickme), this, XRCID("clickme_btn"));

    dlg.ShowModal();
}
```

A few points to note:

- The value of the int returned by XRCID("foo") is guaranteed to be unique within an app.
- However that value isn't predictable, and you shouldn't rely on it being consistent between runs. It certainly won't be the same in different apps.
- Stock Items such as wxID_OK work correctly without requiring XRCID (because, internally, XRCID("wxID_↩ OK") is mapped to wxID_OK).
- Both XRCID and XRCCTRL use the 'name' of the control (as in wxWindow::GetName). This is different from the label that the user sees on e.g. a wxButton.

Subclassing in XRC

You will often want to use subclassed wx controls in your code. There are three ways to do this from XRC:

- Very rarely you might need to create your own wxXmlResourceHandler
- Occasionally wxXmlResource::AttachUnknownControl may be best. See Unknown Objects
- Usually though, the simple 'subclass' keyword will suffice.

Suppose you wanted the wxTextCtrl named "text" to be created as your derived class MyTextCtrl. The only change needed in the XRC file would be in this line:

```
<object class="wxTextCtrl" name="text" subclass="MyTextCtrl"/>
```

The only change in your code would be to use MyTextCtrl in XRCCTRL. However for the subclass to be created successfully, it's important to ensure that it uses wxWidget's RTTI mechanism: see Subclassing for the details.

10.39.2 The XRC sample

A major resource for learning how to use XRC is the XRC Sample. This demonstrates all of the standard uses of XRC, and some of the less common ones. It is strongly suggested that you run it, and look at the well-commented source code to see how it works.

10.39.3 Binary Resource Files

To compile binary resource files, use the command-line `wxrc` utility. It takes one or more file parameters (the input XRC files) and the following switches and options:

- -h (–help): Show a help message.
- -v (–verbose): Show verbose logging information.
- -c (–cpp-code): Write C++ source rather than a XRS file.
- -e (–extra-cpp-code): If used together with -c, generates C++ header file containing class definitions for the windows defined by the XRC file (see special subsection).
- -u (–uncompressed): Do not compress XML files (C++ only).
- -g (–gettext): Output underscore-wrapped strings that poEdit or gettext can scan. Outputs to stdout, or a file if -o is used.
- -n (–function) <name>: Specify C++ function name (use with -c).
- -o (–output) <filename>: Specify the output file, such as resource.xrs or resource.cpp.
- -l (–list-of-handlers) <filename>: Output a list of necessary handlers to this file.

For example:

```
$ wxrc resource.xrc
$ wxrc resource.xrc -o resource.xrs
$ wxrc resource.xrc -v -c -o resource.cpp
```

Note

XRS file is essentially a renamed ZIP archive which means that you can manipulate it with standard ZIP tools. Note that if you are using XRS files, you have to initialize the wxFileSystem archive handler first! It is a simple thing to do:

```
#include <wx/filesys.h>
#include <wx/fs_arc.h>
...
wxFileSystem::AddHandler(new wxArchiveFSHandler);
```

10.39.4 Using Embedded Resources

It is sometimes useful to embed resources in the executable itself instead of loading an external file (e.g. when your app is small and consists only of one exe file). XRC provides means to convert resources into regular C++ file that can be compiled and included in the executable.

Use the `-c` switch to `wxrc` utility to produce C++ file with embedded resources. This file will contain a function called `InitXmlResource` (unless you override this with a command line switch). Use it to load the resource:

```
extern void InitXmlResource(); // defined in generated file
...
wxXmlResource::Get()->InitAllHandlers();
InitXmlResource();
...
```

10.39.5 C++ header file generation

Using the `-e` switch together with `-c`, a C++ header file is written containing class definitions for the GUI windows defined in the XRC file. This code generation can make it easier to use XRC and automate program development. The classes can be used as basis for development, freeing the programmer from dealing with most of the XRC specifics (e.g. `XRCCTRL`).

For each top level window defined in the XRC file a C++ class definition is generated, containing as class members the named widgets of the window. A default constructor for each class is also generated. Inside the constructor all XRC loading is done and all class members representing widgets are initialized.

A simple example will help understand how the scheme works. Suppose you have a XRC file defining a top level window `TestWnd_Base`, which subclasses wxFrame (any other class like `wxDialog` will do also), and has subwidgets wxTextCtrl A and wxButton B.

The XRC file and corresponding class definition in the header file will be something like:

```
<?xml version="1.0"?>
<resource version="2.3.0.1">
    <object class="wxFrame" name="TestWnd_Base">
        <size>-1,-1</size>
        <title>Test</title>
        <object class="wxBoxSizer">
            <orient>wxHORIZONTAL</orient>
            <object class="sizeritem">
                <object class="wxTextCtrl" name="A">
                    <label>Test label</label>
                </object>
            </object>
            <object class="sizeritem">
                <object class="wxButton" name="B">
                    <label>Test button</label>
                </object>
            </object>
        </object>
    </object>
</resource>
```

```
class TestWnd_Base : public wxFrame
{
protected:
    wxTextCtrl* A;
    wxButton* B;

private:
    void InitWidgetsFromXRC()
    {
        wxXmlResource::Get()->LoadObject(this, NULL, "TestWnd", "wxFrame");
        A = XRCCTRL(*this, "A", wxTextCtrl);
        B = XRCCTRL(*this, "B", wxButton);
    }
public:
    TestWnd::TestWnd()
    {
        InitWidgetsFromXRC();
    }
};
```

The generated window class can be used as basis for the full window class. The class members which represent widgets may be accessed by name instead of using `XRCCTRL` every time you wish to reference them (note that they are `protected` class members), though you must still use `XRCID` to refer to widget IDs in the event table.

Example:

```
#include "resource.h"

class TestWnd : public TestWnd_Base
{
public:
    TestWnd()
    {
        // A, B already initialised at this point
        A->SetValue("Updated in TestWnd::TestWnd");
        B->SetValue("Nice :)");
    }
    void OnBPressed(wxEvent& event)
    {
```

```
        Close();
    }
    DECLARE_EVENT_TABLE();
};

BEGIN_EVENT_TABLE(TestWnd,TestWnd_Base)
    EVT_BUTTON(XRCID("B"), TestWnd::OnBPressed)
END_EVENT_TABLE()
```

It is also possible to access the wxSizerItem of a sizer that is part of a resource. This can be done using `XRCSIZERITEM` as shown.

The resource file can have something like this for a sizer item.

```
<object class="spacer" name="area">
  <size>400, 300</size>
</object>
```

The code can then access the sizer item by using `XRCSIZERITEM` and `XRCID` together.

```
wxSizerItem* item = XRCSIZERITEM(*this, "area");
```

10.39.6 Adding New Resource Handlers

Adding a new resource handler is pretty easy.

Typically, to add an handler for the `MyControl` class, you'll want to create the `xh_mycontrol.h` and `xh_mycontrol.cpp` files.

The header needs to contains the `MyControlXmlHandler` class definition:

```
class MyControlXmlHandler : public wxXmlResourceHandler
{
public:
    // Constructor.
    MyControlXmlHandler();

    // Creates the control and returns a pointer to it.
    virtual wxObject *DoCreateResource();

    // Returns true if we know how to create a control for the given node.
    virtual bool CanHandle(wxXmlNode *node);

    // Register with wxWidgets' dynamic class subsystem.
    DECLARE_DYNAMIC_CLASS(MyControlXmlHandler)
};
```

The implementation of your custom XML handler will typically look as:

```
// Register with wxWidgets' dynamic class subsystem.
IMPLEMENT_DYNAMIC_CLASS(MyControlXmlHandler, wxXmlResourceHandler)

MyControlXmlHandler::MyControlXmlHandler()
{
    // this call adds support for all wxWidgets class styles
    // (e.g. wxBORDER_SIMPLE, wxBORDER_SUNKEN, wxWS_EX_* etc etc)
    AddWindowStyles();

    // if MyControl class supports e.g. MYCONTROL_DEFAULT_STYLE
    // you should use:
    //     XRC_ADD_STYLE(MYCONTROL_DEFAULT_STYLE);
}

wxObject *MyControlXmlHandler::DoCreateResource()
{
    // the following macro will init a pointer named "control"
    // with a new instance of the MyControl class, but will NOT
    // Create() it!
    XRC_MAKE_INSTANCE(control, MyControl)

    // this is the point where you'll typically need to do the most
    // important changes: here the control is created and initialized.
    // You'll want to use the wxXmlResourceHandler's getters to
    // do most of your work.
    // If e.g. the MyControl::Create function looks like:
```

```
    //
    // bool MyControl::Create(wxWindow *parent, int id,
    //                        const wxBitmap &first, const wxPoint &posFirst,
    //                        const wxBitmap &second, const wxPoint &posSecond,
    //                        const wxString &theTitle, const wxFont &titleFont,
    //                        const wxPoint &pos, const wxSize &size,
    //                        long style = MYCONTROL_DEFAULT_STYLE,
    //                        const wxString &name = wxT("MyControl"));
    //
    // Then the XRC for your component should look like:
    //
    //    <object class="MyControl" name="some_name">
    //      <first-bitmap>first.xpm</first-bitmap>
    //      <second-bitmap>text.xpm</second-bitmap>
    //      <first-pos>3,3</first-pos>
    //      <second-pos>4,4</second-pos>
    //      <the-title>a title</the-title>
    //      <title-font>
    //        <!-- Standard XRC tags for a font: <size>, <style>, <weight>, etc -->
    //      </title-font>
    //      <!-- XRC also accepts other usual tags for wxWindow-derived classes:
    //           like e.g. <name>, <style>, <size>, <position>, etc -->
    //    </object>
    //
    // And the code to read your custom tags from the XRC file is just:
    control->Create(m_parentAsWindow, GetID(),
                    GetBitmap(wxT("first-bitmap")),
                    GetPosition(wxT("first-pos")),
                    GetBitmap(wxT("second-bitmap")),
                    GetPosition(wxT("second-pos")),
                    GetText(wxT("the-title")),
                    GetFont(wxT("title-font")),
                    GetPosition(), GetSize(), GetStyle(), GetName());

    SetupWindow(control);

    return control;
}

bool MyControlXmlHandler::CanHandle(wxXmlNode *node)
{
    // this function tells XRC system that this handler can parse
    // the <object class="MyControl"> tags
    return IsOfClass(node, wxT("MyControl"));
}
```

You may want to check the wxXmlResourceHandler documentation to see how many built-in getters it contains. It's very easy to retrieve also complex structures out of XRC files using them.

10.40 XRC File Format

This document describes the format of XRC resource files, as used by wxXmlResource.

Formal description in the form of a RELAX NG schema is located in the `misc/schema` subdirectory of the wx↩ Widgets sources.

XRC file is a XML file with all of its elements in the `http://www.wxwidgets.org/wxxrc` namespace. For backward compatibility, `http://www.wxwindows.org/wxxrc` namespace is accepted as well (and treated as identical to `http://www.wxwidgets.org/wxxrc`), but it shouldn't be used in new XRC files.

XRC file contains definitions for one or more *objects* – typically windows. The objects may themselves contain child objects.

Objects defined at the top level, under the root element, can be accessed using wxXmlResource::LoadDialog() and other LoadXXX methods. They must have `name` attribute that is used as LoadXXX's argument (see Object Element for details).

Child objects are not directly accessible via wxXmlResource, they can only be accessed using XRCCTRL().

10.40.1 Resource Root Element

The root element is always `<resource>`. It has one optional attribute, `version`. If set, it specifies version of the file. In absence of `version` attribute, the default is `"0.0.0.0"`.

The version consists of four integers separated by periods. The first three components are major, minor and release number of the wxWidgets release when the change was introduced, the last one is revision number and is 0 for the first incompatible change in given wxWidgets release, 1 for the second and so on. The version changes only if there was an incompatible change introduced; merely adding new kind of objects does not constitute incompatible change.

At the time of writing, the latest version is "2.5.3.0".

Note that even though version attribute is optional, it should always be specified to take advantage of the latest capabilities:

```
<?xml version="1.0"?>
<resource xmlns="http://www.wxwidgets.org/wxxrc" version="2.5.3.0">
    ...
</resource>
```

<resource> may have arbitrary number of object elements as its children; they are referred to as *toplevel* objects in the rest of this document. Unlike objects defined deeper in the hierarchy, toplevel objects *must* have their name attribute set and it must be set to a value unique among root's children.

10.40.2 Defining Objects

Object Element

The <object> element represents a single object (typically a GUI element) and it usually maps directly to a wxWidgets class instance. It has one mandatory attribute, class, and optional name and subclass attributes.

The class attribute must always be present, it tells XRC what wxWidgets object should be created and by which wxXmlResourceHandler.

name is the identifier used to identify the object. This name serves three purposes:

1. It is used by wxXmlResource's various LoadXXX() methods to find the resource by name passed as argument.
2. wxWindow's name (see wxWindow::GetName()) is set to it.
3. Numeric ID of a window or menu item is derived from the name. If the value represents an integer (in decimal notation), it is used for the numeric ID unmodified. If it is one of the wxID_XXX literals defined by wxWidgets (see Stock Items), its respective value is used. Otherwise, the name is transformed into dynamically generated ID. See wxXmlResource::GetXRCID() for more information.

Name attributes must be unique at the top level (where the name is used to load resources) and should be unique among all controls within the same toplevel window (wxDialog, wxFrame).

The subclass attribute optional name of class whose constructor will be called instead of the constructor for "class". See Subclassing for more details.

<object> element may – and almost always do – have children elements. These come in two varieties:

1. Object's properties. A *property* is a value describing part of object's behaviour, for example the "label" property on wxButton defines its label. In the most common form, property is a single element with text content ("\<label\>Cancel\</label\>"), but they may use nested subelements too (e.g. font property). A property can only be listed once in an object's definition.
2. Child objects. Window childs, sizers, sizer items or notebook pages are all examples of child objects. They are represented using nested <object> elements and are can be repeated more than once. The specifics of which object classes are allowed as children are class-specific and are documented below in Supported Controls.

Example:

```
<object class="wxDialog" name="example_dialog">
    <!-- properties: -->
    <title>Non-Derived Dialog Example</title>
    <centered>1</centered>
    <!-- child objects: -->
    <object class="wxBoxSizer">
        <orient>wxVERTICAL</orient>
        <cols>1</cols>
        <rows>0</rows>
        ...
    </object>
</object>
```

Object References

Anywhere an <object> element can be used, <object_ref> may be used instead. <object_ref> is a *reference* to another named (i.e. with the name attribute) <object> element. It has one mandatory attribute, ref, with value containing the name of a named <object> element. When an <object_ref> is encountered, a copy of the referenced <object> element is made in place of <object_ref> occurrence and processed as usual.

For example, the following code:

```
<object class="wxDialog" name="my_dlg">
    ...
</object>
<object_ref name="my_dlg_alias" ref="my_dlg"/>
```

is equivalent to

```
<object class="wxDialog" name="my_dlg">
    ...
</object>
<object class="wxDialog" name="my_dlg_alias">
    ... <!-- same as in my_dlg -->
</object>
```

Additionally, it is possible to override some parts of the referenced object in the <object_ref> pointing to it. This is useful for putting repetitive parts of XRC definitions into a template that can be reused and customized in several places. The two parts are merged as follows:

1. The referred object is used as the initial content.
2. All attributes set on <object_ref> are added to it.
3. All child elements of <object_ref> are scanned. If an element with the same name (and, if specified, the name attribute too) is found in the referred object, they are recursively merged.
4. Child elements in <object_ref> that do not have a match in the referred object are appended to the list of children of the resulting element by default. Optionally, they may have insert_at attribute with two possible values, "begin" or "end". When set to "begin", the element is prepended to the list of children instead of appended.

For example, "my_dlg" in this snippet:

```
<object class="wxDialog" name="template">
    <title>Dummy dialog</title>
    <size>400,400</size>
</object>
<object_ref ref="template" name="my_dlg">
    <title>My dialog</title>
    <centered>1</centered>
</object_ref>
```

is identical to:

```
<object class="wxDialog" name="my_dlg">
    <title>My dialog</title>
    <size>400,400</size>
    <centered>1</centered>
</object>
```

10.40.3 Data Types

There are several property data types that are frequently reused by different properties. Rather than describing their format in the documentation of every property, we list commonly used types in this section and document their format.

Boolean

Boolean values are expressed using either "1" literal (true) or "0" (false).

Floating-point value

Floating point values use POSIX (C locale) formatting – decimal separator is "." regardless of the locale.

Colour

Colour specification can be either any string colour representation accepted by wxColour::Set() or any wxSYS_COLOUR_XXX symbolic name accepted by wxSystemSettings::GetColour(). In particular, the following forms are supported:

- named colours from wxColourDatabase
- HTML-like "#rrggbb" syntax (but not "#rgb")
- CSS-style "rgb(r,g,b)" and "rgba(r,g,b,a)"
- wxSYS_COLOUR_XXX symbolic names

Some examples:

```
<fg>red</fg>
<fg>#ff0000</fg>
<fg>rgb(255,0,0)</fg>
<fg>wxSYS_COLOUR_HIGHLIGHT</fg>
```

Size

Sizes and positions have the form of string with two comma-separated integer components, with optional "d" suffix. Semi-formally:

size := x "," y ["d"]

where x and y are integers. Either of the components (or both) may be "-1" to signify default value. As a shortcut, empty string is equivalent to "-1,-1" (= wxDefaultSize or wxDefaultPosition).

When the "d" suffix is used, integer values are interpreted as dialog units in the parent window.

Examples:

```
42,-1
100,100
100,50d
```

Position

Same as Size.

Dimension

Similarly to sizes, dimensions are expressed as integers with optional "d" suffix. When "d" suffix is used, the integer preceding it is interpreted as dialog units in the parent window.

Text

String properties use several escape sequences that are translated according to the following table:

"_"	"&" (used for accelerators in wxWidgets)
"__"	"_"
"\n"	line break
"\r"	carriage return
"\t"	tab
"\\"	"\"

By default, the text is translated using wxLocale::GetTranslation() before it is used. This can be disabled either globally by not passing wxXRC_USE_LOCALE to wxXmlResource constructor, or by setting the `translate` attribute on the property node to "0":

```
<!-- this is not translated: -->
<label translate="0">_Unix</label>
<!-- but this is: -->
<help>Use Unix-style newlines</help>
```

Note

Even though the "_" character is used instead of "&" for accelerators, it is still possible to use "&". The latter has to be encoded as "&", though, so using "_" is more convenient.

See also

Versions Before 2.5.3.0, Versions Before 2.3.0.1

Non-Translatable Text

Like Text, but the text is never translated and `translate` attribute cannot be used.

String

An unformatted string. Unlike with Text, no escaping or translations are done.

URL

Any URL accepted by wxFileSystem (typically relative to XRC file's location, but can be absolute too). Unlike with Text, no escaping or translations are done.

Bitmap

Bitmap properties contain specification of a single bitmap or icon. In the most basic form, their text value is simply a relative filename (or another wxFileSystem URL) of the bitmap to use. For example:

```
<object class="tool" name="wxID_NEW">
    <tooltip>New</tooltip>
    <bitmap>new.png</bitmap>
</object>
```

The value is interpreted as path relative to the location of XRC file where the reference occurs.

Alternatively, it is possible to specify the bitmap using wxArtProvider IDs. In this case, the property element has no textual value (filename) and instead has the `stock_id` XML attribute that contains stock art ID as accepted by wxArtProvider::GetBitmap(). This can be either custom value (if the app uses app-specific art provider) or one of the predefined wxART_XXX constants.

Optionally, `stock_client` attribute may be specified too and contain one of the predefined wxArtClient values. If it is not specified, the default client ID most appropriate in the context where the bitmap is referenced will be used. In most cases, specifying `stock_client` is not needed.

Examples of stock bitmaps usage:

```
<bitmap stock_id="fixed-width"/>        <!-- custom app-specific art -->
<bitmap stock_id="wxART_FILE_OPEN"/>    <!-- standard art -->
```

If both specifications are provided, then `stock_id` is used if it is recognized by wxArtProvider and the provided bitmap file is used as a fallback.

Style

Style properties (such as window's style or sizer flags) use syntax similar to C++: the style value is OR-combination of individual flags. Symbolic names identical to those used in C++ code are used for the flags. Flags are separated with "|" (whitespace is allowed but not required around it).

The flags that are allowed for a given property are context-dependent.

Examples:

```
<style>wxCAPTION|wxSYSTEM_MENU | wxRESIZE_BORDER</style>
<exstyle>wxDIALOG_EX_CONTEXTHELP</exstyle>
```

Font

XRC uses similar, but more flexible, abstract description of fonts to that used by wxFont class. A font can be described either in terms of its elementary properties, or it can be derived from one of system fonts or the parent window font.

The font property element is a "composite" element: unlike majority of properties, it doesn't have text value but contains several child elements instead. These children are handled in the same way as object properties and can be one of the following "sub-properties":

property	type	description
size	unsigned integer	Pixel size of the font (default: wxNORMAL_FONT's size or `sysfont's` size if the `sysfont` property is used or the current size of the font of the enclosing control if the `inherit` property is used.
style	enum	One of "normal", "italic" or "slant" (default: normal).
weight	enum	One of "normal", "bold" or "light" (default: normal).
family	enum	One of "default", "roman", "script", "decorative", "swiss", "modern" or "teletype" (default: default).

underlined	Boolean	Whether the font should be underlined (default: 0).
face		Comma-separated list of face names; the first one available is used (default: unspecified).
encoding		Charset of the font, unused in Unicode build), as string (default: unspecified).
sysfont		Symbolic name of system standard font(one of wxSYS_*_FONT constants).
inherit	Boolean	If true, the font of the enclosing control is used. If this property and the `sysfont` property are specified the `sysfont` property takes precedence.
relativesize	float	Float, font size relative to chosen system font's or inherited font's size; can only be used when 'sysfont' or 'inherit' is used and when 'size' is not used.

All of them are optional, if they are missing, appropriate wxFont default is used. If the `sysfont` or `inherit` property is used, then the defaults are taken from it instead.

Examples:

```
<font>
    <!-- fixed font: Arial if available, fall back to Helvetica -->
    <face>arial,helvetica</face>
    <size>12</size>
</font>

<font>
    <!-- enlarged, enboldened standard font: -->
    <sysfont>wxSYS_DEFAULT_GUI_FONT</sysfont>
    <weight>bold</weight>
    <relativesize>1.5</relativesize>
</font>
```

Note

You cannot use `inherit` for a font that gets used before the enclosing control is created, e.g. if the control gets the font passed as parameter for its constructor, or if the control is not derived from wxWindow.

Image List

Defines a wxImageList.

The imagelist property element is a "composite" element: unlike majority of properties, it doesn't have text value but contains several child elements instead. These children are handled similarly to object properties and can be one of the following "sub-properties":

property	type	description
mask	Boolean	If masks should be created for all images (default: 1).
size	Size	The size of the images in the list (default: the size of the first bitmap).
bitmap	Bitmap	Adds a new image. Unlike normal object properties, `bitmap` may be used more than once to add multiple images to the list. At least one `bitmap` value is required.

Example:

```
<imagelist>
    <size>32,32</size>
    <bitmap stock_id="wxART_QUESTION"/>
    <bitmap stock_id="wxART_INFORMATION"/>
</imagelist>
```

10.40.4 Controls and Windows

This section describes support wxWindow-derived classes in XRC format.

Standard Properties

The following properties are always (unless stated otherwise in control-specific docs) available for *windows* objects. They are omitted from properties lists below.

property	type	description
pos	Position	Initial position of the window (default: wxDefaultPosition).
size	Size	Initial size of the window (default: wxDefaultSize).
style	Style	Window style for this control. The allowed values depend on what window is being created, consult respective class' constructor documentation for details (default: window-dependent default, usually wxFOO_DEFAULT_STYLE if defined for class wxFoo, 0 if not).
exstyle	Style	Extra style for the window, if any. See wxWindow::SetExtraStyle() (default: not set).
fg	Colour	Foreground colour of the window (default: window's default).
ownfg	Colour	Non-inheritable foreground colour of the window, see wxWindow::SetOwnForegroundColour() (default: none).
bg	Colour	Background colour of the window (default: window's default).
ownbg	Colour	Non-inheritable background colour of the window, see wxWindow::SetOwnBackgroundColour() (default: none).
enabled	Boolean	If set to 0, the control is disabled (default: 1).
focused	Boolean	If set to 1, the control has focus initially (default: 0).
hidden	Boolean	If set to 1, the control is created hidden (default: 0).
tooltip	Text	Tooltip to use for the control (default: not set).

variant	String	Window variant (see wxWindow::SetWindowVariant()), one of "normal", "small", "mini" or "large" (default: "normal") (new since wxWidgets 3.0.2).
font	Font	Font to use for the control (default: window's default).
ownfont	Font	Non-inheritable font to use for the control, see wxWindow::SetOwnFont() (default: none).
help	Text	Context-sensitive help for the control, used by wxHelpProvider (default: not set).

All of these properties are optional.

Supported Controls

This section lists all controls supported by default. For each control, its control-specific properties are listed. If the control can have child objects, it is documented there too; unless said otherwise, XRC elements for these controls cannot have children.

wxAnimationCtrl

property	type	description
animation	URL	Animation file to load into the control (default: none).
inactive-bitmap	Bitmap	Bitmap to use when not playing the animation (default: the default).

wxAuiNotebook

A wxAuiNotebook can have one or more child objects of the `notebookpage` pseudo-class. `notebookpage` objects have the following properties:

property	type	description
label	Text	Page label (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

Each `notebookpage` must have exactly one non-toplevel window as its child.

Example:

```
<object class="wxAuiNotebook">
    <style>wxBK_BOTTOM</style>
    <object class="notebookpage">
        <label>Page 1</label>
        <bitmap>bitmap.png</bitmap>
        <object class="wxPanel" name="page_1">
            ...
        </object>
    </object>
</object>
```

Notice that wxAuiNotebook support in XRC is available in wxWidgets 2.9.5 and later only and you need to explicitly register its handler using

```
#include <wx/xrc/xh_auinotbk.h>

AddHandler(new wxAuiNotebookXmlHandler);
```

to use it.

wxAuiToolBar

Building an XRC for wxAuiToolBar is quite similar to wxToolBar. The only significant differences are:

- the use of the class name wxAuiToolBar
- the styles supported are the ones described in the wxAuiToolBar class definition
- the 'space' pseudo-class has two optional, mutually exclusive, integer properties: 'proportion' and 'width'. If 'width' is specified, a space is added using wxAuiToolBar::AddSpacer(); if 'proportion', the value is used in wxAuiToolBar::AddStretchSpacer(). If neither are provided, the default is a stretch-spacer with a proportion of 1.
- there is an additional pseudo-class, 'label', that has a string property. See wxAuiToolBar::AddLabel().

Refer to the section wxToolBar for more details.

Note

The XML Handler should be explicitly registered:

```
#include <wx/xrc/xh_auitoolb.h>

AddHandler(new wxAuiToolBarXmlHandler);
```

Since

3.1.0

wxBannerWindow

property	type	description
direction	wxLEFT|wxRIGHT|wxTOP|wx↩ BOTTOM	The side along which the banner will be positioned (default: wxLEFT).
bitmap	Bitmap	Bitmap to use as the banner background (default: none).
title	Text	Banner title, should be single line (default: none).
message	Text	Possibly multi-line banner message (default: none).
gradient-start	Colour	Starting colour of the gradient used as banner background. (Optional. Can't be used if a valid bitmap is specified. If used, both gradient values must be set.)
gradient-end	Colour	End colour of the gradient used as banner background. (Optional. Can't be used if a valid bitmap is specified. If used, both gradient values must be set.)

wxBitmapButton

property	type	description
default	Boolean	Should this button be the default button in dialog (default: 0)?
bitmap	Bitmap	Bitmap to show on the button (default: none).
selected	Bitmap	Bitmap to show when the button is selected (default: none, same as `bitmap`).
focus	Bitmap	Bitmap to show when the button has focus (default: none, same as `bitmap`).
disabled	Bitmap	Bitmap to show when the button is disabled (default: none, same as `bitmap`).
hover	Bitmap	Bitmap to show when mouse cursor hovers above the bitmap (default: none, same as `bitmap`).

wxBitmapComboBox

property	type	description
selection	integer	Index of the initially selected item or -1 for no selection (default: -1).
value	String	Initial value in the control (doesn't have to be one of @ content values; default: empty).

If both `value` and `selection` are specified and `selection` is not -1, then `selection` takes precedence.

A wxBitmapComboBox can have one or more child objects of the `ownerdrawnitem` pseudo-class. `ownerdrawnitem` objects have the following properties:

property	type	description
text	Text	Item's label (default: empty).
bitmap	Bitmap	Item's bitmap (default: no bitmap).

Example:

```
<object class="wxBitmapComboBox">
    <selection>1</selection>
    <object class="ownerdrawnitem">
        <text>Foo</text>
        <bitmap>foo.png</bitmap>
    </object>
    <object class="ownerdrawnitem">
        <text>Bar</text>
        <bitmap>bar.png</bitmap>
    </object>
</object>
```

wxBitmapToggleButton

property	type	description
bitmap	Bitmap	Label to display on the button (default: none).
checked	Boolean	Should the button be checked/pressed initially (default: 0)?

wxButton

property	type	description
label	Text	Label to display on the button (may be omitted if only the bitmap or stock ID is used).
bitmap	Bitmap	Bitmap to display in the button (optional).
bitmapposition	wxLEFT|wxRIGHT|wxTOP|wxBOTTOM	Position of the bitmap in the button, see wxButton::SetBitmapPosition() (default: wxLEFT).
default	Boolean	Should this button be the default button in dialog (default: 0)?

wxCalendarCtrl

No additional properties.

wxCheckBox

property	type	description
label	Text	Label to use for the checkbox (default: empty).
checked	Boolean	Should the checkbox be checked initially (default: 0)?

wxCheckListBox

property	type	description
content	items	Content of the control; this property has any number of <item> XML elements as its children, with the items text as their text values (default: empty).

The <item> elements have listbox items' labels as their text values. They can also have optional checked XML attribute – if set to "1", the value is initially checked.

Example:

```
<object class="wxCheckListBox">
    <content>
        <item checked="1">Download library</item>
        <item checked="1">Compile samples</item>
        <item checked="1">Skim docs</item>
        <item checked="1">Finish project</item>
        <item>Wash car</item>
    </content>
</object>
```

wxChoice

property	type	description
selection	integer	Index of the initially selected item or -1 for no selection (default: -1).
content	items	Content of the control; this property has any number of <item> XML elements as its children, with the items text as their text values (default: empty).

Example:

```
<object class="wxChoice" name="controls_choice">
    <content>
        <item>See</item>
        <item>Hear</item>
        <item>Feel</item>
```

```
        <item>Smell</item>
        <item>Taste</item>
        <item>The Sixth Sense!</item>
    </content>
</object>
```

wxChoicebook

property	type	description
imagelist	Image List	Image list to use for the images (default: none, built implicitly).

Additionally, a choicebook can have one or more child objects of the `choicebookpage` pseudo-class (similarly to wxNotebook and its `notebookpage`).

`choicebookpage` objects have the following properties:

property	type	description
label	Text	Sheet page's title (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none, mutually exclusive with `image`).
image	integer	The zero-based index of the image associated with the item into the image list (default: none, mutually exclusive with `bitmap`, only if imagelist was set).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

Each `choicebookpage` has exactly one non-toplevel window as its child.

wxCommandLinkButton

The wxCommandLinkButton contains a main title-like `label` and an optional `note` for longer description. The main `label` and the `note` can be concatenated into a single string using a new line character between them (notice that the `note` part can have more new lines in it).

property	type	description
label	Text	First line of text on the button, typically the label of an action that will be made when the button is pressed (default: empty).
note	Text	Second line of text describing the action performed when the button is pressed (default: none).

wxCollapsiblePane

property	type	description
label	Text	Label to use for the collapsible section (default: empty).
collapsed	Boolean	Should the pane be collapsed initially (default: 0)?

wxCollapsiblePane may contain single optional child object of the `panewindow` pseudo-class type. `panewindow` itself must contain exactly one child that is a sizer or a non-toplevel window object.

wxColourPickerCtrl

property	type	description
value	Colour	Initial value of the control (default: wxBLACK).

wxComboBox

property	type	description
selection	integer	Index of the initially selected item or -1 for no selection (default: not used).
content	items	Content of the control; this property has any number of `<item>` XML elements as its children, with the items text as their text values (default: empty).
value	String	Initial value in the control (doesn't have to be one of @ content values; default: empty).

If both `value` and `selection` are specified and `selection` is not -1, then `selection` takes precedence.

Example:

```
<object class="wxComboBox" name="controls_combobox">
    <style>wxCB_DROPDOWN</style>
    <value>nedit</value>
    <content>
        <item>vim</item>
        <item>emacs</item>
        <item>notepad.exe</item>
        <item>bbedit</item>
    </content>
</object>
```

wxComboCtrl

property	type	description
value	String	Initial value in the control (default: empty).

wxDatePickerCtrl

No additional properties.

wxDialog

property	type	description
title	Text	Dialog's title (default: empty).
icon	Bitmap	Dialog's icon (default: not used).
centered	Boolean	Whether the dialog should be centered on the screen (default: 0).

wxDialog may have optional children: either exactly one sizer child or any number of non-toplevel window objects. If sizer child is used, it sets size hints too.

wxDirPickerCtrl

property	type	description
value	String	Initial value of the control (default: empty).
message	Text	Message shown to the user in wxDirDialog shown by the control (default: empty).

wxEditableListBox

property	type	description
label	Text	Label shown above the list (default: empty).
content	items	Content of the control; this property has any number of <item> XML elements as its children, with the items text as their text values (default: empty).

Example:

```
<object class="wxEditableListBox" name="controls_listbox">
    <size>250,160</size>
    <label>List of things</label>
    <content>
        <item>Milk</item>
        <item>Pizza</item>
        <item>Bread</item>
        <item>Orange juice</item>
        <item>Paper towels</item>
    </content>
</object>
```

wxFileCtrl

property	type	description
defaultdirectory	String	Sets the current directory displayed in the control (default: empty).
defaultfilename	String	Selects a certain file (default: empty).
wildcard	String	Sets the wildcard, which can contain multiple file types, for example: "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.gif" (default: all files).

wxFilePickerCtrl

property	type	description
value	String	Initial value of the control (default: empty).
message	Text	Message shown to the user in wxDirDialog shown by the control (default: empty).
wildcard	String	Sets the wildcard, which can contain multiple file types, for example: "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.gif" (default: all files).

wxFontPickerCtrl

property	type	description
value	Font	Initial value of the control (default: wxNORMAL_FONT).

wxFrame

property	type	description
title	Text	Frame's title (default: empty).
icon	Bitmap	Frame's icon (default: not used).
centered	Boolean	Whether the frame should be centered on the screen (default: 0).

wxFrame may have optional children: either exactly one sizer child or any number of non-toplevel window objects. If sizer child is used, it sets size hints too.

wxGauge

property	type	description
range	integer	Maximum value of the gauge (default: 100).
value	integer	Initial value of the control (default: 0).
shadow	Dimension	Rendered shadow size (default: none; ignored by most platforms).
bezel	Dimension	Rendered bezel size (default: none; ignored by most platforms).

wxGenericDirCtrl

property	type	description
defaultfolder	String	Initial folder (default: empty).
filter	Text	Filter string, using the same syntax as used by wxFileDialog, e.g. "All files (∗.∗)|∗.∗|JPEG files (∗.jpg)|∗.jpg" (default: empty).
defaultfilter	integer	Zero-based index of default filter (default: 0).

wxGrid

No additional properties.

wxHtmlWindow

property	type	description
url	URL	Page to display in the window (default: none).
htmlcode	Text	HTML markup to display in the window (default: none).
borders	Dimension	Border around HTML content (default: 0).

At most one of `url` and `htmlcode` properties may be specified, they are mutually exclusive. If neither is set, the window is initialized to show empty page.

wxHyperlinkCtrl

property	type	description
label	Text	Label to display on the control (default: empty).
url	URL	URL to open when the link is clicked (default: empty).

wxListBox

property	type	description
selection	integer	Index of the initially selected item or -1 for no selection (default: -1).
content	items	Content of the control; this property has any number of `<item>` XML elements as its children, with the items text as their text values (default: empty).

Example:

```
<object class="wxListBox" name="controls_listbox">
    <size>250,160</size>
    <style>wxLB_SINGLE</style>
    <content>
        <item>Milk</item>
        <item>Pizza</item>
        <item>Bread</item>
        <item>Orange juice</item>
        <item>Paper towels</item>
    </content>
</object>
```

wxListbook

property	type	description
imagelist	Image List	Image list to use for the images (default: none, built implicitly).

Additionally, a listbook can have one or more child objects of the `listbookpage` pseudo-class (similarly to wxNotebook and its `notebookpage`).

`listbookpage` objects have the following properties:

property	type	description
label	Text	Sheet page's title (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none, mutually exclusive with `image`).
image	integer	The zero-based index of the image associated with the item into the image list (default: none, mutually exclusive with `bitmap`, only if imagelist was set).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

Each `listbookpage` has exactly one non-toplevel window as its child.

wxListCtrl

property	type	description
imagelist	Image List	The normal (wxIMAGE_LIST_NORMAL) image list (default: none, built implicitly).
imagelist-small	Image List	The small (wxIMAGE_LIST_SMALL) image list (default: none, built implicitly).

A list control can have optional child objects of the listitem class. Report mode list controls (i.e. created with `wxLC_REPORT` style) can in addition have optional listcol child objects.

listcol

The `listcol` class can only be used for wxListCtrl children. It can have the following properties (all of them optional):

property	type	description
align	wxListColumnFormat	The alignment for the item. Can be one of `wxLIST_FORMAT_LEFT`, `wxLIST_FORMAT_RIGHT` or `wxLIST_FORMAT_CENTRE`.
text	Text	The title of the column.
width	integer	The column width.
image	integer	The zero-based index of the image associated with the item in the 'small' image list.

The columns are appended to the control in order of their appearance and may be referenced by 0-based index in the `col` attributes of subsequent `listitem` objects.

listitem

The `listitem` is a child object for the class wxListCtrl. It can have the following properties (all of them optional):

property	type	description
align	wxListColumnFormat	The alignment for the item. Can be one of `wxLIST_FORMAT_LEFT`, `wxLIST_FORMAT_RIGHT` or `wxLIST_FORMAT_CENTRE`.
bg	Colour	The background color for the item.
bitmap	Bitmap	Add a bitmap to the (normal) Image List associated with the wxListCtrl parent and associate it with this item. If the imagelist is not defined it will be created implicitly (default: none, mutually exclusive with `image`).
bitmap-small	Bitmap	Add a bitmap in the 'small' Image List associated with the wxListCtrl parent and associate it with this item. If the 'small' imagelist is not defined it will be created implicitly (default: none, mutually exclusive with `image-small`).
col	integer	The zero-based column index.
image	integer	The zero-based index of the image associated with the item in the (normal) image list (default: none, mutually exclusive with `bitmap`, only if imagelist was set).
image-small	integer	The zero-based index of the image associated with the item in the 'small' image list (default: none, mutually exclusive with `bitmap-small`, only if imagelist-small was set).

data	integer	The client data for the item.
font	Font	The font for the item.
state	Style	The item state. Can be any combination of the following values: • `wxLIST_STATE_FOCUSED`: The item has the focus. • `wxLIST_STATE_SELECTED`: The item is selected.
text	Text	The text label for the item.
textcolour	Colour	The text colour for the item.

Notice that the item position can't be specified here, the items are appended to the list control in order of their appearance.

wxMDIParentFrame

wxMDIParentFrame supports the same properties that wxFrame does.

wxMDIParentFrame may have optional children. When used, the child objects must be of wxMDIChildFrame type.

wxMDIChildFrame

wxMDIChildFrame supports the same properties that wxFrame and wxMDIParentFrame do.

wxMDIChildFrame can only be used as as immediate child of wxMDIParentFrame.

wxMDIChildFrame may have optional children: either exactly one sizer child or any number of non-toplevel window objects. If sizer child is used, it sets size hints too.

wxMenu

property	type	description
label	Text	Menu's label (default: empty, but required for menus other than popup menus).
style	Style	Window style for the menu.
help	Text	Help shown in statusbar when the menu is selected (only for submenus of another wxMenu, default: none).
enabled	Boolean	Is the submenu item enabled (only for submenus of another wxMenu, default: 1)?

Note that unlike most controls, wxMenu does *not* have Standard Properties, with the exception of `style`.

A menu object can have one or more child objects of the wxMenuItem or wxMenu classes or `break` or `separator` pseudo-classes.

The `separator` pseudo-class is used to insert separators into the menu and has neither properties nor children. Likewise, `break` inserts a break (see wxMenu::Break()).

wxMenuItem objects support the following properties:

property	type	description
label	Text	Item's label (may be omitted if stock ID is used).
accel	Non-Translatable Text	Item's accelerator (default: none).
radio	Boolean	Item's kind is wxITEM_RADIO (default: 0)?
checkable	Boolean	Item's kind is wxITEM_CHECK (default: 0)?
bitmap	Bitmap	Bitmap to show with the item (default: none).
bitmap2	Bitmap	Bitmap for the checked state (wxMSW, if checkable; default: none).
help	Text	Help shown in statusbar when the item is selected (default: none).
enabled	Boolean	Is the item enabled (default: 1)?
checked	Boolean	Is the item checked initially (default: 0)?

Example:

```
<object class="wxMenu" name="menu_edit">
  <style>wxMENU_TEAROFF</style>
  <label>_Edit</label>
  <object class="wxMenuItem" name="wxID_FIND">
    <label>_Find...</label>
    <accel>Ctrl-F</accel>
  </object>
  <object class="separator"/>
  <object class="wxMenuItem" name="menu_fuzzy">
    <label>Translation is _fuzzy</label>
    <checkable>1</checkable>
  </object>
  <object class="wxMenu" name="submenu">
    <label>A submenu</label>
    <object class="wxMenuItem" name="foo">...</object>
    ...
  </object>
  <object class="separator" platform="unix"/>
  <object class="wxMenuItem" name="wxID_PREFERENCES" platform="unix">
    <label>_Preferences</label>
  </object>
</object>
```

wxMenuBar

property	type	description
style	Style	Window style for the menu bar.

Note that unlike most controls, wxMenuBar does *not* have Standard Properties, with the exception of `style`.

A menubar can have one or more child objects of the wxMenu class.

wxNotebook

property	type	description
imagelist	Image List	Image list to use for the images (default: none, built implicitly).

A notebook can have one or more child objects of the `notebookpage` pseudo-class.

`notebookpage` objects have the following properties:

property	type	description
label	Text	Page's title (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none, mutually exclusive with `image`).
image	integer	The zero-based index of the image associated with the item into the image list (default: none, mutually exclusive with `bitmap`, only if imagelist was set).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

Each `notebookpage` has exactly one non-toplevel window as its child.

Example:

```
<object class="wxNotebook">
    <style>wxBK_BOTTOM</style>
    <object class="notebookpage">
        <label>Page 1</label>
        <object class="wxPanel" name="page_1">
            ...
        </object>
    </object>
    <object class="notebookpage">
        <label>Page 2</label>
        <object class="wxPanel" name="page_2">
            ...
        </object>
    </object>
</object>
```

wxOwnerDrawnComboBox

wxOwnerDrawnComboBox has the same properties as wxComboBox, plus the following additional properties:

property	type	description
buttonsize	Size	Size of the dropdown button (default: default).

wxPanel

No additional properties.

wxPanel may have optional children: either exactly one sizer child or any number of non-toplevel window objects.

wxPropertySheetDialog

property	type	description
title	Text	Dialog's title (default: empty).
icon	Bitmap	Dialog's icon (default: not used).
centered	Boolean	Whether the dialog should be centered on the screen (default: 0).
buttons	Style	Buttons to show, combination of flags accepted by wxPropertySheetDialog::CreateButtons() (default: 0).

A sheet dialog can have one or more child objects of the `propertysheetpage` pseudo-class (similarly to wxNotebook and its `notebookpage`). `propertysheetpage` objects have the following properties:

property	type	description
label	Text	Sheet page's title (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

Each `propertysheetpage` has exactly one non-toplevel window as its child.

wxRadioButton

property	type	description
label	Text	Label shown on the radio button (default: empty).
value	Boolean	Initial value of the control (default: 0).

wxRadioBox

property	type	description
label	Text	Label for the whole box (default: empty).
dimension	integer	Specifies the maximum number of rows (if style contains `wxRA_SPECIFY_ROWS`) or columns (if style contains `wxRA_SPECIFY_COLS`) for a two-dimensional radiobox (default: 1).
selection	integer	Index of the initially selected item or -1 for no selection (default: -1).
content	items	Content of the control; this property has any number of `<item>` XML elements as its children, with the items text as their text values (see below; default: empty).

The `<item>` elements have radio buttons' labels as their text values. They can also have some optional XML *attributes* (not properties!):

attribute	type	description
tooltip	String	Tooltip to show over this ratio button (default: none).
helptext	String	Contextual help for this radio button (default: none).
enabled	Boolean	Is the button enabled (default: 1)?
hidden	Boolean	Is the button hidden initially (default: 0)?

Example:

```
<object class="wxRadioBox" name="controls_radiobox">
    <style>wxRA_SPECIFY_COLS</style>
    <label>Radio stations</label>
    <dimension>1</dimension>
    <selection>0</selection>
    <content>
        <item tooltip="Powerful radio station" helptext="This station is for amateurs of hard rock and
       heavy metal">Power 108</item>
        <item tooltip="Disabled radio station" enabled="0">Power 0</item>
        <item tooltip="">WMMS 100.7</item>
        <item tooltip="E=mc^2">Energy 98.3</item>
        <item helptext="Favourite chukcha's radio">CHUM FM</item>
        <item>92FM</item>
```

```
        <item hidden="1">Very quit station</item>
    </content>
</object>
```

wxRibbonBar

property	type	description
art-provider	String	One of `default`, `aui` or `msw` (default: `default`).

A wxRibbonBar may have wxRibbonPage child objects. The `page` pseudo-class may be used instead of `wxRibbonPage` when used as wxRibbonBar children.

Example:

```
<object class="wxRibbonBar" name="ribbonbar">
    <object class="page" name="FilePage">
        <label>First</label>
        <object class="panel">
            <label>File</label>
            <object class="wxRibbonButtonBar">
                <object class="button" name="Open">
                    <bitmap>open.xpm</bitmap>
                    <label>Open</label>
                </object>
            </object>
        </object>
    </object>
    <object class="page" name="ViewPage">
        <label>View</label>
        <object class="panel">
            <label>Zoom</label>
            <object class="wxRibbonGallery">
                <object class="item">
                    <bitmap>zoomin.xpm</bitmap>
                </object>
                <object class="item">
                    <bitmap>zoomout.xpm</bitmap>
                </object>
            </object>
        </object>
    </object>
</object>
```

Notice that wxRibbonBar support in XRC is available in wxWidgets 2.9.5 and later only and you need to explicitly register its handler using

```
#include <wx/xrc/xh_ribbon.h>

AddHandler(new wxRibbonXmlHandler);
```

to use it.

wxRibbonButtonBar

No additional properties.

wxRibbonButtonBar can have child objects of the `button` pseudo-class. `button` objects have the following properties:

property	type	description
hybrid	Boolean	If true, the `wxRIBBON_BUTTON_HYBRID` kind is used (default: false).
disabled	Boolean	Whether the button should be disabled (default: false).
label	Text	Item's label (default: empty).
bitmap	Bitmap	Item's bitmap (default: none).
small-bitmap	Bitmap	Small bitmap (default: none).
disabled-bitmap	Bitmap	Disabled bitmap (default: none).
small-disabled-bitmap	Bitmap	Small disabled bitmap (default: none).
help	Text	Item's help text (default: none).

wxRibbonControl

No additional properties.

Objects of this type *must* be subclassed with the `subclass` attribute.

wxRibbonGallery

No additional properties.

wxRibbonGallery can have child objects of the `item` pseudo-class. `item` objects have the following properties:

property	type	description
bitmap	Bitmap	Item's bitmap (default: none).

wxRibbonPage

property	type	description
label	Text	Label (default: none).
icon	Bitmap	Icon (default: none).

A wxRibbonPage may have children of any type derived from wxRibbonControl. Most commonly, wxRibbonPanel is used. As a special case, the `panel` pseudo-class may be used instead of `wxRibbonPanel` when used as wxRibbonPage children.

wxRibbonPanel

property	type	description
label	Text	Label (default: none).
icon	Bitmap	Icon (default: none).

A wxRibbonPanel may have children of any type derived from wxRibbonControl or a single wxSizer child with non-ribbon windows in it.

wxRichTextCtrl

property	type	description
value	Text	Initial value of the control (default: empty).
maxlength	integer	Maximum length of the text entered (default: unlimited).

Notice that wxRichTextCtrl support in XRC is available in wxWidgets 2.9.5 and later only and you need to explicitly register its handler using

```
#include <wx/xrc/xh_richtext.h>

AddHandler(new wxRichTextCtrl);
```

to use it.

wxScrollBar

property	type	description
value	integer	Initial position of the scrollbar (default: 0).
range	integer	Maximum value of the gauge (default: 10).
thumbsize	integer	Size of the thumb (default: 1).
pagesize	integer	Page size (default: 1).

wxScrolledWindow

property	type	description
scrollrate	Size	Scroll rate in *x* and *y* directions (default: not set; required if the window has a sizer child).

wxScrolledWindow may have optional children: either exactly one sizer child or any number of non-toplevel window objects. If sizer child is used, wxSizer::FitInside() is used (instead of wxSizer::Fit() as usual) and so the children don't determine scrolled window's minimal size, they only affect *virtual* size. Usually, both `scrollrate` and either `size` or `minsize` on containing sizer item should be used in this case.

wxSimpleHtmlListBox

wxSimpleHtmlListBox has same properties as wxListBox. The only difference is that the text contained in `<item>` elements is HTML markup. Note that the markup has to be escaped:

```
<object class="wxSimpleHtmlListBox">
    <content>
        <item>&amp;lt;b&amp;gt;Bold&amp;lt;/b&amp;gt; Milk</item>
    </content>
</object>
```

(X)HTML markup elements cannot be included directly:

```
<object class="wxSimpleHtmlListBox">
    <content>
        <!-- This is incorrect, doesn't work! -->
        <item><b>Bold</b> Milk</item>
    </content>
</object>
```

wxSimplebook

wxSimplebook is similar to wxNotebook but simpler: as it doesn't show any page headers, it doesn't use neither image list nor individual page bitmaps and while it still accepts page labels, they are optional as they are not shown to the user neither.

So `simplebookpage` child elements, that must occur inside this object, only have the following properties:

`choicebookpage` objects have the following properties:

property	type	description
label	Text	Page's label (default: empty).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

As with all the other book page elements, each `simplebookpage` must have exactly one non-toplevel window as its child.

Since

3.0.2

wxSlider

property	type	description
value	integer	Initial value of the control (default: 0).
min	integer	Minimum allowed value (default: 0).
max	integer	Maximum allowed value (default: 100).
pagesize	integer	Page size; number of steps the slider moves when the user moves pages up or down (default: unset).
linesize	integer	Line size; number of steps the slider moves when the user moves it up or down a line (default: unset).
tickfreq	integer	Tick marks frequency (Windows only; default: unset).
tick	integer	Tick position (Windows only; default: unset).
thumb	integer	Thumb length (Windows only; default: unset).
selmin	integer	Selection start position (Windows only; default: unset).
selmax	integer	Selection end position (Windows only; default: unset).

wxSpinButton

property	type	description
value	integer	Initial value of the control (default: 0).
min	integer	Minimum allowed value (default: 0).
max	integer	Maximum allowed value (default: 100).

wxSpinCtrl

wxSpinCtrl supports the same properties as wxSpinButton and, since wxWidgets 2.9.5, another one:

base	integer	Numeric base, currently can be only 10 or 16 (default: 10).

wxSplitterWindow

property	type	description
orientation	String	Orientation of the splitter, either "vertical" or "horizontal" (default: horizontal).
sashpos	Dimension	Initial position of the sash (default: 0).
minsize	Dimension	Minimum child size (default: not set).
gravity	Floating-point value	Sash gravity, see wxSplitterWindow::SetSashGravity() (default: not set).

wxSplitterWindow must have one or two children that are non-toplevel window objects. If there's only one child, it is used as wxSplitterWindow's only visible child. If there are two children, the first one is used for left/top child and the second one for right/bottom child window.

wxSearchCtrl

property	type	description
value	Text	Initial value of the control (default: empty).

wxStatusBar

property	type	description
fields	integer	Number of status bar fields (default: 1).
widths	String	Comma-separated list of *fields* integers. Each value specifies width of one field; the values are interpreted using the same convention used by wxStatusBar::SetStatusWidths() (default: not set).
styles	String	Comma-separated list of *fields* style values. Each value specifies style of one field and can be one of `wxSB_NORMAL`, `wxSB_FLAT`, `wxSB_RAISED` or `wxSB_SUNKEN` (default: not set).

wxStaticBitmap

property	type	description
bitmap	Bitmap	Bitmap to display (required).

wxStaticBox

property	type	description
label	Text	Static box's label (default: empty).

wxStaticLine

No additional properties.

wxStaticText

property	type	description
label	Text	Label to display (default: empty).
wrap	Dimension	Wrap the text so that each line is at most the given number of pixels, see wxStaticText::Wrap() (default: no wrap).

wxTextCtrl

property	type	description
value	Text	Initial value of the control (default: empty).
maxlength	integer	Maximum length of the text which can be entered by user (default: unlimited).
hint	Text	Hint shown in empty control (new since wxWidgets 3.0.1).

wxTimePickerCtrl

No additional properties.

wxToggleButton

property	type	description
label	Text	Label to display on the button (default: empty).
checked	Boolean	Should the button be checked/pressed initially (default: 0)?

wxToolBar

property	type	description
bitmapsize	Size	Size of toolbar bitmaps (default: not set).
margins	Size	Margins (default: platform default).
packing	integer	Packing, see wxToolBar::SetToolPacking() (default: not set).
separation	integer	Default separator size, see wxToolBar::SetToolSeparation() (default: not set).
dontattachtoframe	Boolean	If set to 0 and the toolbar object is child of a wxFrame, wxFrame::SetToolBar() is called; otherwise, you have to add it to a frame manually. The toolbar is attached by default, you have to set this property to 1 to disable this behaviour (default: 0).

A toolbar can have one or more child objects of any wxControl-derived class or one of three pseudo-classes: `separator`, `space` or `tool`.

The `separator` pseudo-class is used to insert separators into the toolbar and has neither properties nor children. Similarly, the `space` pseudo-class is used for stretchable spaces (see wxToolBar::AddStretchableSpace(), new since wxWidgets 2.9.1).

The `tool` pseudo-class objects specify toolbar buttons and have the following properties:

property	type	description
bitmap	Bitmap	Tool's bitmap (default: empty).
bitmap2	Bitmap	Bitmap for disabled tool (default: derived from `bitmap`).
label	Text	Label to display on the tool (default: no label).
radio	Boolean	Item's kind is wxITEM_RADIO (default: 0)?
toggle	Boolean	Item's kind is wxITEM_CHECK (default: 0)?
dropdown	see below	Item's kind is wxITEM_DROPDOWN (default: 0)? (only available since wxWidgets 2.9.0)
tooltip	Text	Tooltip to use for the tool (default: none).
longhelp	Text	Help text shown in statusbar when the mouse is on the tool (default: none).
disabled	Boolean	Is the tool initially disabled (default: 0)?
checked	Boolean	Is the tool initially checked (default: 0)? (only available since wxWidgets 2.9.3)

The presence of a `dropdown` property indicates that the tool is of type wxITEM_DROPDOWN. It must be either empty or contain exactly one wxMenu child object defining the drop-down button associated menu.

Notice that `radio`, `toggle` and `dropdown` are mutually exclusive.

Children that are not `tool`, `space` or `separator` must be instances of classes derived from wxControl and are added to the toolbar using wxToolBar::AddControl().

Example:

```
<object class="wxToolBar">
    <style>wxTB_FLAT|wxTB_NODIVIDER</style>
    <object class="tool" name="foo">
        <bitmap>foo.png</bitmap>
        <label>Foo</label>
    </object>
    <object class="tool" name="bar">
        <bitmap>bar.png</bitmap>
        <label>Bar</label>
    </object>
    <object class="separator"/>
    <object class="tool" name="view_auto">
        <bitmap>view.png</bitmap>
        <label>View</label>
        <dropdown>
            <object class="wxMenu">
                <object class="wxMenuItem" name="view_as_text">
                    <label>View as text</label>
                </object>
                <object class="wxMenuItem" name="view_as_hex">
                    <label>View as binary</label>
                </object>
            </object>
        </dropdown>
    </object>
    <object class="space"/>
    <object class="wxComboBox">
        <content>
            <item>Just</item>
            <item>a combobox</item>
            <item>in the toolbar</item>
        </content>
    </object>
</object>
```

wxToolbook

property	type	description
imagelist	Image List	Image list to use for the images (default: none, built implicitly).

A toolbook can have one or more child objects of the `toolbookpage` pseudo-class (similarly to wxNotebook and its `notebookpage`).

`toolbookpage` objects have the following properties:

property	type	description
label	Text	Sheet page's title (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none, mutually exclusive with `image`).
image	integer	The zero-based index of the image associated with the item into the image list (default: none, mutually exclusive with `bitmap`, only if imagelist was set).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

Each `toolbookpage` has exactly one non-toplevel window as its child.

wxTreeCtrl

property	type	description
imagelist	Image List	Image list to use for the images (default: none).

wxTreebook

property	type	description
imagelist	Image List	Image list to use for the images (default: none, built implicitly).

A treebook can have one or more child objects of the `treebookpage` pseudo-class (similarly to wxNotebook and its `notebookpage`).

`treebookpage` objects have the following properties:

property	type	description
depth	integer	Page's depth in the labels tree (default: 0; see below).
label	Text	Sheet page's title (default: empty).
bitmap	Bitmap	Bitmap shown alongside the label (default: none, mutually exclusive with `image`).
image	integer	The zero-based index of the image associated with the item into the image list (default: none, mutually exclusive with `bitmap`, only if imagelist was set).
selected	Boolean	Is the page selected initially (only one page can be selected; default: 0)?

expanded	Boolean	If set to 1, the page is initially expanded. By default all pages are initially collapsed.

Each `treebookpage` has exactly one non-toplevel window as its child.

The tree of labels is not described using nested `treebookpage` objects, but using the *depth* property. Toplevel pages have depth 0, their child pages have depth 1 and so on. A `treebookpage's` label is inserted as child of the latest preceding page with depth equal to *depth-1*. For example, this XRC markup:

```
<object class="wxTreebook">
  ...
  <object class="treebookpage">
    <depth>0</depth>
    <label>Page 1</label>
    <object class="wxPanel">...</object>
  </object>
  <object class="treebookpage">
    <depth>1</depth>
    <label>Subpage 1A</label>
    <object class="wxPanel">...</object>
  </object>
  <object class="treebookpage">
    <depth>2</depth>
    <label>Subsubpage 1</label>
    <object class="wxPanel">...</object>
  </object>
  <object class="treebookpage">
    <depth>1</depth>
    <label>Subpage 1B</label>
    <object class="wxPanel">...</object>
  </object>
  <object class="treebookpage">
    <depth>2</depth>
    <label>Subsubpage 2</label>
    <object class="wxPanel">...</object>
  </object>
  <object class="treebookpage">
    <depth>0</depth>
    <label>Page 2</label>
    <object class="wxPanel">...</object>
  </object>
</object>
```

corresponds to the following tree of labels:

- Page 1
  - Subpage 1A
    - Subsubpage 1
  - Subpage 1B
    - Subsubpage 2
- Page 2

wxWizard

property	type	description
bitmap	Bitmap	Bitmap to display on the left side of the wizard (default: none).

A wizard object can have one or more child objects of the wxWizardPage or wxWizardPageSimple classes. They both support the following properties (in addition to Standard Properties):

property	type	description
title	Text	Wizard window's title (default: none).
bitmap	Bitmap	Page-specific bitmap (default: none).

wxWizardPage and wxWizardPageSimple nodes may have optional children: either exactly one sizer child or any number of non-toplevel window objects.

wxWizardPageSimple pages are automatically chained together; wxWizardPage pages transitions must be handled programmatically.

10.40.5 Sizers

Sizers are handled slightly differently in XRC resources than they are in wxWindow hierarchy. wxWindow's sizers hierarchy is parallel to the wxWindow children hierarchy: child windows are children of their parent window and the sizer (or sizers) form separate hierarchy attached to the window with wxWindow::SetSizer().

In XRC, the two hierarchies are merged together: sizers are children of other sizers or windows and they can contain child window objects.

If a sizer is child of a window object in the resource, it must be the only child and it will be attached to the parent with wxWindow::SetSizer(). Additionally, if the window doesn't have its size explicitly set, wxSizer::Fit() is used to resize the window. If the parent window is toplevel window, wxSizer::SetSizeHints() is called to set its hints.

A sizer object can have one or more child objects of one of two pseudo-classes: `sizeritem` or `spacer` (see wxStdDialogButtonSizer for an exception). The former specifies an element (another sizer or a window) to include in the sizer, the latter adds empty space to the sizer.

`sizeritem` objects have exactly one child object: either another sizer object, or a window object. `spacer` objects don't have any children, but they have one property:

property	type	description
size	Size	Size of the empty space (default: `wxDefaultSize`).

Both `sizeritem` and `spacer` objects can have any of the following properties:

property	type	description
option	integer	The "option" value for sizers. Used by wxBoxSizer to set proportion of the item in the growable direction (default: 0).
flag	Style	wxSizerItem flags (default: 0).
border	Dimension	Size of the border around the item (directions are specified in flags) (default: 0).
minsize	Size	Minimal size of this item (default: no min size).
ratio	Size	Item ratio, see wxSizer::SetRatio() (default: no ratio).
cellpos	Position	(wxGridBagSizer only) Position, see wxGBSizerItem::SetPos() (required).
cellspan	Size	(wxGridBagSizer only) Span, see wxGBSizerItem::SetSpan() (required).

Example of sizers XRC code:

```
<object class="wxDialog" name="derived_dialog">
    <title>Derived Dialog Example</title>
    <centered>1</centered>
    <!-- this sizer is set to be this dialog's sizer: -->
    <object class="wxFlexGridSizer">
        <cols>1</cols>
        <rows>0</rows>
        <vgap>0</vgap>
        <hgap>0</hgap>
        <growablecols>0:1</growablecols>
        <growablerows>0:1</growablerows>
        <object class="sizeritem">
            <flag>wxALIGN_CENTRE|wxALL</flag>
            <border>5</border>
            <object class="wxButton" name="my_button">
                <label>My Button</label>
```

```
            </object>
        </object>
        <object class="sizeritem">
            <flag>wxALIGN_CENTRE|wxALL</flag>
            <border>5</border>
            <object class="wxBoxSizer">
                <orient>wxHORIZONTAL</orient>
                <object class="sizeritem">
                    <flag>wxALIGN_CENTRE|wxALL</flag>
                    <border>5</border>
                    <object class="wxCheckBox" name="my_checkbox">
                        <label>Enable this text control:</label>
                    </object>
                </object>
                <object class="sizeritem">
                    <flag>wxALIGN_CENTRE|wxALL</flag>
                    <border>5</border>
                    <object class="wxTextCtrl" name="my_textctrl">
                        <size>80,-1</size>
                        <value></value>
                    </object>
                </object>
            </object>
        </object>
        ...
    </object>
</object>
```

The sizer classes that can be used are listed below, together with their class-specific properties. All classes support the following properties:

property	type	description
minsize	Size	Minimal size that this sizer will have, see wxSizer::SetMinSize() (default: no min size).

wxBoxSizer

property	type	description
orient	Style	Sizer orientation, "wxHORIZONTAL" or "wxVERTICAL" (default: wxHORIZONTAL).

wxStaticBoxSizer

property	type	description
orient	Style	Sizer orientation, "wxHORIZONTAL" or "wxVERTICAL" (default: wxHORIZONTAL).
label	Text	Label to be used for the static box around the sizer (default: empty).

wxGridSizer

property	type	description
rows	unsigned integer	Number of rows in the grid (default: 0 - determine automatically).
cols	unsigned integer	Number of columns in the grid (default: 0 - determine automatically).
vgap	Dimension	Vertical gap between children (default: 0).
hgap	Dimension	Horizontal gap between children (default: 0).

wxFlexGridSizer

property	type	description
rows	unsigned integer	Number of rows in the grid (default: 0 - determine automatically).
cols	unsigned integer	Number of columns in the grid (default: 0 - determine automatically).
vgap	Dimension	Vertical gap between children (default: 0).
hgap	Dimension	Horizontal gap between children (default: 0).
flexibledirection	Style	Flexible direction, `wxVERTICAL`, `wxHORIZONTAL` or `wxBOTH` (default). This property is only available since wxWidgets 2.9.5.
nonflexiblegrowmode	Style	Grow mode in the non-flexible direction, `wxFLEX_GROWMODE_NONE`, `wxFLEX_GROWMODE_SPECIFIED` (default) or `wxFLEX_GROWMODE_ALL`. This property is only available since wxWidgets 2.9.5.
growablerows	comma-separated integers list	Comma-separated list of indexes of rows that are growable (none by default). Since wxWidgets 2.9.5 optional proportion can be appended to each number after a colon (:).
growablecols	comma-separated integers list	Comma-separated list of indexes of columns that are growable (none by default). Since wxWidgets 2.9.5 optional proportion can be appended to each number after a colon (:).

wxGridBagSizer

property	type	description
vgap	Dimension	Vertical gap between children (default: 0).
hgap	Dimension	Horizontal gap between children (default: 0).
flexibledirection	Style	Flexible direction, `wxVERTICAL`, `wxHORIZONTAL`, `wxBOTH` (default: `wxBOTH`).
nonflexiblegrowmode	Style	Grow mode in the non-flexible direction, `wxFLEX_GROWMODE_NONE`, `wxFLEX_GROWMODE_SPECIFIED`, `wxFLEX_GROWMODE_ALL` (default: `wxFLEX_GROWMODE_SPECIFIED`).
growablerows	comma-separated integers list	Comma-separated list of indexes of rows that are growable, optionally the proportion can be appended after each number separated by a : (default: none).
growablecols	comma-separated integers list	Comma-separated list of indexes of columns that are growable, optionally the proportion can be appended after each number separated by a : (default: none).

wxWrapSizer

property	type	description
orient	Style	Sizer orientation, "wxHORIZONTAL" or "wxVERTICAL" (default: wxHORIZONTAL).
flag	Style	wxWrapSizer flags (default: 0).

wxStdDialogButtonSizer

Unlike other sizers, wxStdDialogButtonSizer has neither `sizeritem` nor `spacer` children. Instead, it has one or more children of the `button` pseudo-class. `button` objects have no properties and they must always have exactly one child of the `wxButton` class or a class derived from wxButton.

Example:

```
<object class="wxStdDialogButtonSizer">
    <object class="button">
        <object class="wxButton" name="wxID_OK">
            <label>OK</label>
        </object>
    </object>
    <object class="button">
        <object class="wxButton" name="wxID_CANCEL">
            <label>Cancel</label>
        </object>
    </object>
</object>
```

10.40.6 Other Objects

In addition to describing UI elements, XRC files can contain non-windows objects such as bitmaps or icons. This is a concession to Windows developers used to storing them in Win32 resources.

Note that unlike Win32 resources, bitmaps included in XRC files are *not* embedded in the XRC file itself. XRC file only contains a reference to another file with bitmap data.

wxBitmap

Bitmaps are stored in `<object>` element with class set to `wxBitmap`. Such bitmaps can then be loaded using wxXmlResource::LoadBitmap(). The content of the element is exactly same as in the case of bitmap properties, except that toplevel `<object>` is used.

For example, instead of:

```
<bitmap>mybmp.png</bitmap>
<bitmap stock_id="wxART_NEW"/>
```

toplevel wxBitmap resources would look like:

```
<object class="wxBitmap" name="my_bitmap">mybmp.png</object>
<object class="wxBitmap" name="my_new_bitmap" stock_id="wxART_NEW"/>
```

wxIcon

wxIcon resources are identical to wxBitmap ones, except that the class is `wxIcon`.

10.40.7 Platform Specific Content

It is possible to conditionally process parts of XRC files on some platforms only and ignore them on other platforms. *Any* element in XRC file, be it toplevel or arbitrarily nested one, can have the `platform` attribute. When used, `platform` contains |-separated list of platforms that this element should be processed on. It is filtered out and ignored on any other platforms.

Possible elemental values are:

`win`	Windows
`mac`	Mac OS X (or Mac Classic in wxWidgets version supporting it)
`unix`	Any Unix platform *except* OS X

Examples:

```
<label platform="win">Windows</label>
<label platform="unix">Unix</label>
<label platform="mac">Mac OS X</label>
<help platform="mac|unix">Not a Windows machine</help>
```

10.40.8 ID Ranges

Usually you won't care what value the XRCID macro returns for the ID of an object. Sometimes though it is convenient to have a range of IDs that are guaranteed to be consecutive. An example of this would be connecting a group of similar controls to the same event handler.

The following XRC fragment 'declares' an ID range called *foo* and another called *bar*; each with some items.

```
    <object class="wxButton" name="foo[start]">
    <object class="wxButton" name="foo[end]">
    <object class="wxButton" name="foo[2]">
    ...
```

```
    <object class="wxButton" name="bar[0]">
    <object class="wxButton" name="bar[2]">
    <object class="wxButton" name="bar[1]">
    ...
<ids-range name="foo" />
<ids-range name="bar" size="30" start="10000" />
```

For the range foo, no *size* or *start* parameters were given, so the size will be calculated from the number of range items, and IDs allocated by wxWindow::NewControlId (so they'll be negative). Range bar asked for a size of 30, so this will be its minimum size: should it have more items, the range will automatically expand to fit them. It specified a start ID of 10000, so XRCID("bar[0]") will be 10000, XRCID("bar[1]") 10001 etc. Note that if you choose to supply a start value it must be positive, and it's your responsibility to avoid clashes.

For every ID range, the first item can be referenced either as *rangename*[0] or *rangename*[start]. Similarly *rangename*[end] is the last item. Using [start] and [end] is more descriptive in e.g. a Bind() event range or a *for* loop, and they don't have to be altered whenever the number of items changes.

Whether a range has positive or negative IDs, [start] is always a smaller number than [end]; so code like this works as expected:

```
for (int n=XRCID("foo[start]"); n <= XRCID("foo[end]"); ++n)
    ...
```

ID ranges can be seen in action in the *objref* dialog section of the XRC Sample.

Note

- All the items in an ID range must be contained in the same XRC file.
- You can't use an ID range in a situation where static initialisation occurs; in particular, they won't work as expected in an event table. This is because the event table's IDs are set to their integer values before the XRC file is loaded, and aren't subsequently altered when the XRCID value changes.

Since

2.9.2

10.40.9 Extending the XRC Format

The XRC format is designed to be extensible and allows specifying and loading custom controls. The three available mechanisms are described in the rest of this section in the order of increasing complexity.

Subclassing

The simplest way to add custom controls is to set the `subclass` attribute of `<object>` element:

```
<object name="my_value" class="wxTextCtrl" subclass="MyTextCtrl">
  <style>wxTE_MULTILINE</style>
  ...etc., setup wxTextCtrl as usual...
</object>
```

In that case, wxXmlResource will create an instance of the specified subclass (`MyTextCtrl` in the example above) instead of the class (`wxTextCtrl` above) when loading the resource. However, the rest of the object's loading (calling its Create() method, setting its properties, loading any children etc.) will proceed in *exactly* the same way as it would without `subclass` attribute. In other words, this approach is only sufficient when the custom class is just a small modification (e.g. overridden methods or customized events handling) of an already supported classes.

The subclass must satisfy a number of requirements:

1. It must be derived from the class specified in `class` attribute.

2. It must be visible in wxWidget's pseudo-RTTI mechanism, i.e. there must be a DECLARE_DYNAMIC_CLASS() entry for it.

3. It must support two-phase creation. In particular, this means that it has to have default constructor.

4. It cannot provide custom Create() method and must be constructible using base `class'` Create() method (this is because XRC will call Create() of `class`, not `subclass`). In other words, *creation* of the control must not be customized.

Unknown Objects

A more flexible solution is to put a *placeholder* in the XRC file and replace it with custom control after the resource is loaded. This is done by using the `unknown` pseudo-class:

```
<object class="unknown" name="my_placeholder"/>
```

The placeholder is inserted as dummy wxPanel that will hold custom control in it. At runtime, after the resource is loaded and a window created from it (using e.g. wxXmlResource::LoadDialog()), use code must call wxXmlResource::AttachUnknownControl() to insert the desired control into placeholder container.

This method makes it possible to insert controls that are not known to XRC at all, but it's also impossible to configure the control in XRC description in any way. The only properties that can be specified are the standard window properties.

Note

`unknown` class cannot be combined with `subclass` attribute, they are mutually exclusive.

Adding Custom Classes

Finally, XRC allows adding completely new classes in addition to the ones listed in this document. A class for which wxXmlResourceHandler is implemented can be used as first-class object in XRC simply by passing class name as the value of `class` attribute:

```
<object name="my_ctrl" class="MyWidget">
  <my_prop>foo</my_prop>
  ...etc., whatever MyWidget handler accepts...
</object>
```

The only requirements on the class are that

1. the class must derive from wxObject

2. it must support wxWidget's pseudo-RTTI mechanism

Child elements of `<object>` are handled by the custom handler and there are no limitations on them imposed by XRC format.

This is the only mechanism that works for toplevel objects – custom controls are accessible using the type-unsafe wxXmlResource::LoadObject() method.

10.40.10 Packed XRC Files

In addition to plain XRC files, wxXmlResource supports (if wxFileSystem support is compiled in) compressed XRC resources. Compressed resources have either .zip or .xrs extension and are simply ZIP files that contain arbitrary number of XRC files and their dependencies (bitmaps, icons etc.).

10.40.11 Older Format Versions

This section describes differences in older revisions of XRC format (i.e. files with older values of `version` attribute of `<resource>`).

Versions Before 2.5.3.0

Version 2.5.3.0 introduced C-like handling of "\\" in text. In older versions, "\n", "\t" and "\r" escape sequences were replaced with respective characters in the same matter it's done in C, but "\\" was left intact instead of being replaced with single "\", as one would expect. Starting with 2.5.3.0, all of them are handled in C-like manner.

Versions Before 2.3.0.1

Prior to version 2.3.0.1, "$" was used for accelerators instead of "_" or "&". For example,

```
<label>$File</label>
```

was used in place of current version's

```
<label>_File</label>
```

(or "&File").

10.41 Scrolled Windows

Scrollbars come in various guises in wxWidgets.

All windows have the potential to show a vertical scrollbar and/or a horizontal scrollbar: it is a basic capability of a window. However, in practice, not all windows do make use of scrollbars, such as a single-line wxTextCtrl.

Because any class derived from wxWindow may have scrollbars, there are functions to manipulate the scrollbars and event handlers to intercept scroll events. But just because a window generates a scroll event, doesn't mean that the window necessarily handles it and physically scrolls the window. The base class wxWindow in fact doesn't have any default functionality to handle scroll events. If you created a wxWindow object with scrollbars, and then clicked on the scrollbars, nothing at all would happen. This is deliberate, because the *interpretation* of scroll events varies from one window class to another.

wxScrolledWindow (formerly wxCanvas) is an example of a window that adds functionality to make scrolling really work. It assumes that scrolling happens in consistent units, not different-sized jumps, and that page size is represented by the visible portion of the window. It is suited to drawing applications, but perhaps not so suitable for a sophisticated editor in which the amount scrolled may vary according to the size of text on a given line. For this, you would derive from wxWindow and implement scrolling yourself. wxGrid is an example of a class that implements its own scrolling, largely because columns and rows can vary in size.

See also

wxScrollBar

10.41.1 The Scrollbar Model

The function wxWindow::SetScrollbar gives a clue about the way a scrollbar is modeled. This function takes the following arguments:

orientation	Which scrollbar: wxVERTICAL or wxHORIZONTAL.
position	The position of the scrollbar in scroll units.
visible	The size of the visible portion of the scrollbar, in scroll units.
range	The maximum position of the scrollbar.
refresh	Whether the scrollbar should be repainted.

orientation determines whether we're talking about the built-in horizontal or vertical scrollbar.

position is simply the position of the 'thumb' (the bit you drag to scroll around). It is given in scroll units, and so is relative to the total range of the scrollbar.

visible gives the number of scroll units that represents the portion of the window currently visible. Normally, a scrollbar is capable of indicating this visually by showing a different length of thumb.

range is the maximum value of the scrollbar, where zero is the start position. You choose the units that suit you, so if you wanted to display text that has 100 lines, you would set this to 100. Note that this doesn't have to correspond to the number of pixels scrolled - it is up to you how you actually show the contents of the window.

refresh just indicates whether the scrollbar should be repainted immediately or not.

10.41.2 An Example

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time. You would use:

```
SetScrollbar(wxVERTICAL, 0, 16, 50);
```

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34. You can determine how many lines are currently visible by dividing the current view size by the character height in pixels.

When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and SetScrollbar call into a function named AdjustScrollbars, which can be called initially and also from your wxSizeEvent handler function.

10.42 wxDialog Overview

Classes: wxDialog, wxDialogLayoutAdapter

A dialog box is similar to a panel, in that it is a window which can be used for placing controls, with the following exceptions:

- A surrounding frame is implicitly created.
- Extra functionality is automatically given to the dialog box, such as tabbing between items (currently Windows only).
- If the dialog box is *modal*, the calling program is blocked until the dialog box is dismissed.

For a set of dialog convenience functions, including file selection, see Dialogs.

See also wxTopLevelWindow and wxWindow for inherited member functions. Validation of data in controls is covered in wxValidator Overview.

10.42.1 Automatic Scrolled Dialogs

As an ever greater variety of mobile hardware comes to market, it becomes more imperative for wxWidgets applications to adapt to these platforms without putting too much burden on the programmer. One area where wxWidgets can help is in adapting dialogs for the lower resolution screens that inevitably accompany a smaller form factor.

wxDialog therefore supplies a global wxDialogLayoutAdapter class that implements automatic scrolling adaptation for most sizer-based custom dialogs.

Many applications should therefore be able to adapt to small displays with little or no work, as far as dialogs are concerned. By default this adaptation is off. To switch scrolling adaptation on globally in your application, call the static function wxDialog::EnableLayoutAdaptation passing true. You can also adjust adaptation on a per-dialog basis by calling wxDialog::SetLayoutAdaptationMode with one of `wxDIALOG_ADAPTATION_MODE_DEFAULT` (use the global setting), `wxDIALOG_ADAPTATION_MODE_ENABLED` or `wxDIALOG_ADAPTATION_MODE_DISABLED`.

The last two modes override the global adaptation setting. With adaptation enabled, if the display size is too small for the dialog, wxWidgets (or rather the standard adapter class wxStandardDialogLayoutAdapter) will make part of the dialog scrolling, leaving standard buttons in a non-scrolling part at the bottom of the dialog. This is done as follows, in wxDialogLayoutAdapter::DoLayoutAdaptation called from within wxDialog::Show or wxDialog::ShowModal:

- If wxDialog::GetContentWindow returns a window derived from wxBookCtrlBase, the pages are made scrollable and no other adaptation is done.
- wxWidgets looks for a wxStdDialogButtonSizer and uses it for the non-scrolling part.
- If that search failed, wxWidgets looks for a horizontal wxBoxSizer with one or more standard buttons, with identifiers such as `wxID_OK` and `wxID_CANCEL`.
- If that search failed too, wxWidgets finds 'loose' standard buttons (in any kind of sizer) and adds them to a wxStdDialogButtonSizer. If no standard buttons were found, the whole dialog content will scroll.
- All the children apart from standard buttons are reparented onto a new wxScrolledWindow object, using the old top-level sizer for the scrolled window and creating a new top-level sizer to lay out the scrolled window and standard button sizer.

Customising Scrolling Adaptation

In addition to switching adaptation on and off globally and per dialog, you can choose how aggressively wxWidgets will search for standard buttons by setting wxDialog::SetLayoutAdaptationLevel. By default, all the steps described above will be performed but by setting the level to 1, for example, you can choose to only look for wxStdDialogButtonSizer.

You can use wxDialog::AddMainButtonId to add identifiers for buttons that should also be treated as standard buttons for the non-scrolling area.

You can derive your own class from wxDialogLayoutAdapter or wxStandardDialogLayoutAdapter and call wxDialog::SetLayoutAdapter, deleting the old object that this function returns. Override the functions CanDoLayoutAdaptation and DoLayoutAdaptation to test for adaptation applicability and perform the adaptation.

You can also override wxDialog::CanDoLayoutAdaptation and wxDialog::DoLayoutAdaptation in a class derived from wxDialog.

Where Scrolling Adaptation May Fail

Because adaptation rearranges your sizer and window hierarchy, it is not fool-proof, and may fail in the following situations:

- The dialog doesn't use sizers.
- The dialog implementation makes assumptions about the window hierarchy, for example getting the parent of a control and casting to the dialog class.
- The dialog does custom painting and/or event handling not handled by the scrolled window. If this problem can be solved globally, you can derive a new adapter class from wxStandardDialogLayoutAdapter and override its CreateScrolledWindow function to return an instance of your own class.

- The dialog has unusual layout, for example a vertical sizer containing a mixture of standard buttons and other controls.
- The dialog makes assumptions about the sizer hierarchy, for example to show or hide children of the top-level sizer. However, the original sizer hierarchy will still hold until Show or ShowModal is called.

You can help make sure that your dialogs will continue to function after adaptation by:

- avoiding the above situations and assumptions;
- using wxStdDialogButtonSizer;
- only making assumptions about hierarchy immediately after the dialog is created;
- using an intermediate sizer under the main sizer, a false top-level sizer that can be relied on to exist for the purposes of manipulating child sizers and windows;
- overriding wxDialog::GetContentWindow to return a book control if your dialog implements pages: wxWidgets will then only make the pages scrollable.

wxPropertySheetDialog and wxWizard

Adaptation for wxPropertySheetDialog is always done by simply making the pages scrollable, since wxDialog::GetContentWindow returns the dialog's book control and this is handled by the standard layout adapter.

wxWizard uses its own CanDoLayoutAdaptation and DoLayoutAdaptation functions rather than the global adapter: again, only the wizard pages are made scrollable.

10.43 wxValidator Overview

The aim of the validator concept is to make dialogs very much easier to write.

A validator is an object that can be plugged into a control (such as a wxTextCtrl), and mediates between C++ data and the control, transferring the data in either direction and validating it. It also is able to intercept events generated by the control, providing filtering behaviour without the need to derive a new control class.

You can use a stock validator, such as wxTextValidator (which does text control data transfer, validation and filtering) and wxGenericValidator (which does data transfer for a range of controls); or you can write your own.

Here is an example of wxTextValidator usage.

```
wxTextCtrl *txt1 = new wxTextCtrl(
    this, -1, wxT(""), wxDefaultPosition, wxDefaultSize, 0,
    wxTextValidator(wxFILTER_ALPHA, &g_data.m_string));
```

In this example, the text validator object provides the following functionality:

- It transfers the value of g_data.m_string (a wxString variable) to the wxTextCtrl when the dialog is initialised.
- It transfers the wxTextCtrl data back to this variable when the dialog is dismissed.
- It filters input characters so that only alphabetic characters are allowed.

The validation and filtering of input is accomplished in two ways. When a character is input, wxTextValidator checks the character against the allowed filter flag (`wxFILTER_ALPHA` in this case). If the character is inappropriate, it is vetoed (does not appear) and a warning beep sounds (unless wxValidator::SetBellOnError(false) has been called). The second type of validation is performed when the dialog is about to be dismissed, so if the default string contained invalid characters already, a dialog box is shown giving the error, and the dialog is not dismissed.

Note that any wxWindow may have a validator; using the `wxWS_EX_VALIDATE_RECURSIVELY` style (see wxWindow extended styles) you can also implement recursive validation.

See also

wxValidator, wxTextValidator, wxGenericValidator, wxIntegerValidator, wxFloatingPointValidator

10.43.1 Anatomy of a Validator

A programmer creating a new validator class should provide the following functionality.

A validator constructor is responsible for allowing the programmer to specify the kind of validation required, and perhaps a pointer to a C++ variable that is used for storing the data for the control. If such a variable address is not supplied by the user, then the validator should store the data internally.

The wxValidator::Validate member function should return true if the data in the control (not the C++ variable) is valid. It should also show an appropriate message if data was not valid.

The wxValidator::TransferToWindow member function should transfer the data from the validator or associated C++ variable to the control.

The wxValidator::TransferFromWindow member function should transfer the data from the control to the validator or associated C++ variable.

There should be a copy constructor, and a wxValidator::Clone function which returns a copy of the validator object. This is important because validators are passed by reference to window constructors, and must therefore be cloned internally.

You can optionally define event handlers for the validator, to implement filtering. These handlers will capture events before the control itself does (see How Events are Processed). For an example implementation, see the `valtext.h` and `valtext.cpp` files in the wxWidgets library.

10.43.2 How Validators Interact with Dialogs

For validators to work correctly, validator functions must be called at the right times during dialog initialisation and dismissal.

When a wxDialog::Show is called (for a modeless dialog) or wxDialog::ShowModal is called (for a modal dialog), the function wxWindow::InitDialog is automatically called. This in turn sends an initialisation event to the dialog. The default handler for the `wxEVT_INIT_DIALOG` event is defined in the wxWindow class to simply call the function wxWindow::TransferDataToWindow. This function finds all the validators in the window's children and calls the wx↩Validator::TransferToWindow function for each. Thus, data is transferred from C++ variables to the dialog just as the dialog is being shown.

Note

If you are using a window or panel instead of a dialog, you will need to call wxWindow::InitDialog explicitly before showing the window.

When the user clicks on a button, for example the OK button, the application should first call wxWindow::Validate, which returns false if any of the child window validators failed to validate the window data. The button handler should return immediately if validation failed. Secondly, the application should call wxWindow::TransferDataFromWindow and return if this failed. It is then safe to end the dialog by calling wxDialog::EndModal (if modal) or wxDialog::Show (if modeless).

In fact, wxDialog contains a default command event handler for the `wxID_OK` button. It goes like this:

```
void wxDialog::OnOK(wxCommandEvent& event)
{
    if ( Validate() && TransferDataFromWindow() )
    {
        if ( IsModal() )
            EndModal(wxID_OK);
        else
        {
            SetReturnCode(wxID_OK);
            this->Show(false);
        }
    }
}
```

So if using validators and a normal OK button, you may not even need to write any code for handling dialog dismissal.

If you load your dialog from a resource file, you will need to iterate through the controls setting validators, since validators can't be specified in a dialog resource.

10.44 wxDataObject Overview

This overview discusses data transfer through clipboard or drag and drop.

In wxWidgets, these two ways to transfer data (either between different applications or inside one and the same) are very similar which allows to implement both of them using almost the same code - or, in other words, if you implement drag and drop support for your application, you get clipboard support for free and vice versa.

At the heart of both clipboard and drag and drop operations lies the wxDataObject class. The objects of this class (or, to be precise, classes derived from it) represent the data which is being carried by the mouse during drag and drop operation or copied to or pasted from the clipboard. wxDataObject is a "smart" piece of data because it knows which formats it supports (see GetFormatCount and GetAllFormats) and knows how to render itself in any of them (see GetDataHere). It can also receive its value from the outside in a format it supports if it implements the SetData method. Please see the documentation of this class for more details.

Both clipboard and drag and drop operations have two sides: the source and target, the data provider and the data receiver. These which may be in the same application and even the same window when, for example, you drag some text from one position to another in a word processor. Let us describe what each of them should do.

See also

Drag and Drop Overview, Clipboard and Drag & Drop, Drag & Drop Sample

10.44.1 The Data Provider (Source)

The data provider is responsible for creating a wxDataObject containing the data to be transferred. Then it should either pass it to the clipboard using wxClipboard::SetData function or to wxDropSource and call wxDropSource::DoDragDrop function.

The only (but important) difference is that the object for the clipboard transfer must always be created on the heap (i.e. using `new`) and it will be freed by the clipboard when it is no longer needed (indeed, it is not known in advance when, if ever, the data will be pasted from the clipboard). On the other hand, the object for drag and drop operation must only exist while wxDropSource::DoDragDrop executes and may be safely deleted afterwards and so can be created either on heap or on stack (i.e. as a local variable).

Another small difference is that in the case of clipboard operation, the application usually knows in advance whether it copies or cuts (i.e. copies and deletes) data - in fact, this usually depends on which menu item the user chose. But for drag and drop it can only know it after wxDropSource::DoDragDrop returns (from its return value).

10.44.2 The Data Receiver (Target)

To receive (paste in usual terminology) data from the clipboard, you should create a wxDataObject derived class which supports the data formats you need and pass it as argument to wxClipboard::GetData. If it returns false, no data in (any of) the supported format(s) is available. If it returns true, the data has been successfully transferred to wxDataObject.

For drag and drop case, the wxDropTarget::OnData virtual function will be called when a data object is dropped, from which the data itself may be requested by calling wxDropTarget::GetData method which fills the data object.

10.45 Drag and Drop Overview

It may be noted that data transfer to and from the clipboard is quite similar to data transfer with drag and drop and the code to implement these two types is almost the same.

In particular, both data transfer mechanisms store data in some kind of wxDataObject and identify its format(s) using the wxDataFormat class.

Note that `wxUSE_DRAG_AND_DROP` must be defined in `setup.h` in order to use drag and drop in wxWidgets.

See also

wxDataObject Overview, Clipboard and Drag & Drop, Drag & Drop Sample

10.45.1 Drop Source Requirements

To be a *"drop source"*, i.e. to provide the data which may be dragged by the user elsewhere, you should implement the following steps:

- **Preparation:** First of all, a data object must be created and initialized with the data you wish to drag. For example:

```
wxTextDataObject my_data("This text will be dragged.");
```

- **Drag start**: To start the dragging process (typically in response to a mouse click) you must call wxDropSource::DoDragDrop like this:

```
wxDropSource dragSource( this );
dragSource.SetData( my_data );
wxDragResult result = dragSource.DoDragDrop( true );
```

- **Dragging:** The call to DoDragDrop() blocks the program until the user releases the mouse button (unless you override the wxDropSource::GiveFeedback function to do something special). When the mouse moves in a window of a program which understands the same drag-and-drop protocol (any program under Windows or any program supporting the XDnD protocol under X Windows), the corresponding wxDropTarget methods are called - see below.

- **Processing the result**: DoDragDrop() returns an *effect* code which is one of the values of `wxDragResult` enum (explained in wxDropTarget page):

```
switch (result)
{
    case wxDragCopy:
        // copy the data
        break;
    case wxDragMove:
        // move the data
        break;
    default:
        // do nothing
        break;
}
```

10.45.2 Drop Target Requirements

To be a *"drop target"*, i.e. to receive the data dropped by the user you should follow the instructions below:

- **Initialization:** For a window to be a drop target, it needs to have an associated wxDropTarget object. Normally, you will call wxWindow::SetDropTarget during window creation associating your drop target with it. You must derive a class from wxDropTarget and override its pure virtual methods. Alternatively, you may derive from wxTextDropTarget or wxFileDropTarget and override their OnDropText() or OnDropFiles() method.

- **Drop:** When the user releases the mouse over a window, wxWidgets asks the associated wxDropTarget object if it accepts the data. For this, a wxDataObject must be associated with the drop target and this data object will be responsible for the format negotiation between the drag source and the drop target. If all goes well, then wxDropTarget::OnData will get called and the wxDataObject belonging to the drop target can get filled with data.

- **The end**: After processing the data, DoDragDrop() returns either wxDragCopy or wxDragMove depending on the state of the keys Ctrl, Shift and Alt at the moment of the drop. There is currently no way for the drop target to change this return code.

10.46 wxHTML Overview

The wxHTML library provides classes for parsing and displaying HTML.

It is not intended to be a high-end HTML browser. If you are looking for something like that try http://www.mozilla.org/.

wxHTML can be used as a generic rich text viewer - for example to display a nice About Box (like those of GNOME apps) or to display the result of database searching. There is a wxFileSystem class which allows you to use your own virtual file systems.

wxHtmlWindow supports tag handlers. This means that you can easily extend wxHtml library with new, unsupported tags. Not only that, you can even use your own application-specific tags!

See src/html/m_*.cpp files for details.

There is a generic wxHtmlParser class, independent of wxHtmlWindow.

10.46.1 wxHTML Quick Start

Displaying HTML

First of all, you must include wx/wxhtml.h.

Class wxHtmlWindow (derived from wxScrolledWindow) is used to display HTML documents.

It has two important methods: wxHtmlWindow::LoadPage and wxHtmlWindow::SetPage. LoadPage loads and displays HTML file while SetPage displays directly the passed **string**. See the example:

```
mywin->LoadPage("test.htm");
mywin->SetPage("htmlbody"
               "h1Error/h1"
               "Some error occurred :-H)"
               "/body/hmtl");
```

Setting up wxHtmlWindow

Because wxHtmlWindow is derived from wxScrolledWindow and not from wxFrame, it doesn't have visible frame. But the user usually wants to see the title of HTML page displayed somewhere and the frame's titlebar is the ideal place for it.

wxHtmlWindow provides 2 methods in order to handle this: wxHtmlWindow::SetRelatedFrame and wxHtmlWindow::SetRelatedStatusBar. See the example:

```
html = new wxHtmlWindow(this);
html->SetRelatedFrame(this, "HTML : %%s");
html->SetRelatedStatusBar(0);
```

The first command associates the HTML object with its parent frame (this points to wxFrame object there) and sets the format of the title. Page title "Hello, world!" will be displayed as "HTML : Hello, world!" in this example.

The second command sets which frame's status bar should be used to display browser's messages (such as "Loading..." or "Done" or hypertext links).

Customizing wxHtmlWindow

You can customize wxHtmlWindow by setting font size, font face and borders (space between border of window and displayed HTML). Related functions:

- wxHtmlWindow::SetFonts
- wxHtmlWindow::SetBorders

- wxHtmlWindow::ReadCustomization
- wxHtmlWindow::WriteCustomization

The last two functions are used to store user customization info wxConfig stuff (for example in the registry under Windows, or in a dotfile under Unix).

10.46.2 HTML Printing

The wxHTML library provides printing facilities with several levels of complexity. The easiest way to print an HTML document is to use the wxHtmlEasyPrinting class.

It lets you print HTML documents with only one command and you don't have to worry about deriving from the wxPrintout class at all. It is only a simple wrapper around the wxHtmlPrintout, normal wxWidgets printout class.

And finally there is the low level class wxHtmlDCRenderer which you can use to render HTML into a rectangular area on any DC. It supports rendering into multiple rectangles with the same width. The most common use of this is placing one rectangle on each page or printing into two columns.

10.46.3 Help Files Format

wxHTML library can be used to show an help manual to the user; in fact, it supports natively (through wxHtmlHelpController) a reduced version of MS HTML Workshop format.

A **book** consists of three files: the header file, the contents file and the index file.

You can make a regular zip archive of these files, plus the HTML and any image files, for wxHTML (or helpview) to read; and the "`.zip`" file can optionally be renamed to "`.htb`".

Header file (.hhp)

The header file must contain these lines (and may contain additional lines which are ignored):

```
Contents file=filename.hhc
Index file=filename.hhk
Title=title of your book
Default topic=default page to be displayed.htm
```

All filenames (including the Default topic) are relative to the location of the "`.hhp`" file.

Note

For localization, in addition the "`.hhp`" file may contain the line

```
Charset=rfc_charset
```

which specifies what charset (e.g. "iso8859_1") was used in contents and index files. Please note that this line is incompatible with MS HTML Help Workshop and it would either silently remove it or complain with some error. See also Writing Non-English Applications.

Contents file (.hhc)

Contents file has HTML syntax and it can be parsed by regular HTML parser. It contains exactly one list (<ul>.... statement):

```
<ul>
    <li><object type="text/sitemap">
            <param name="Name" value="@topic name@">
            <param name="ID" value=@numeric_id@>
            <param name="Local" value="@filename.htm@">
        </object>
```

```
    <li><object type="text/sitemap">
            <param name="Name" value="@topic name@">
            <param name="ID" value=@numeric_id@>
            <param name="Local" value="@filename.htm@">
        </object>
    ...
</ul>
```

You can modify value attributes of param tags. The *topic name* is name of chapter/topic as is displayed in contents, *filename.htm* is the HTML page name (relative to the "`.hhp`" file) and *numeric_id* is optional, it is used only when you use wxHtmlHelpController::Display(int).

Items in the list may be nested - one <li> statement may contain a <ul> sub-statement:

```
<ul>
    <li><object type="text/sitemap">
            <param name="Name" value="Top node">
            <param name="Local" value="top.htm">
        </object>
    <ul>
        <li><object type="text/sitemap">
            <param name="Name" value="subnode in topnode">
            <param name="Local" value="subnode1.htm">
            </object>
        ...
    </ul>
    <li><object type="text/sitemap">
            <param name="Name" value="Another Top">
            <param name="Local" value="top2.htm">
        </object>
    ...
</ul>
```

Index Files (.hhk)

Index files have same format as contents files except that ID params are ignored and sublists are **not** allowed.

10.46.4 Input Filters

The wxHTML library provides a mechanism for reading and displaying files of many different file formats.

wxHtmlWindow::LoadPage can load not only HTML files but any known file. To make a file type known to wxHtml↩ Window you must create a wxHtmlFilter filter and register it using wxHtmlWindow::AddFilter.

10.46.5 Cells and Containers

This article describes mechanism used by wxHtmlWinParser and wxHtmlWindow to parse and display HTML documents.

Cells

You can divide any text (or HTML) into small fragments. Let's call these fragments **cells**. Cell is for example one word, horizontal line, image or any other part of document. Each cell has width and height (except special "magic" cells with zero dimensions - e.g. colour changers or font changers). See wxHtmlCell.

Containers

Container is kind of cell that may contain sub-cells. Its size depends on number and sizes of its sub-cells (and also depends on width of window). See wxHtmlContainerCell, wxHtmlCell::Layout. This image shows the cells and containers:

Using Containers in Tag Handler

wxHtmlWinParser provides a user-friendly way of managing containers. It is based on the idea of opening and closing containers.

Use wxHtmlWinParser::OpenContainer to open new a container *within* an already opened container. This new container is a *sub-container* of the old one. (If you want to create a new container with the same depth level you can call `CloseContainer()`; OpenContainer();.)

Use wxHtmlWinParser::CloseContainer to close the container. This doesn't create a new container with same depth level but it returns "control" to the parent container. See explanation:

There clearly must be same number of calls to OpenContainer as to CloseContainer.

This code creates a new paragraph (container at same depth level) with "Hello, world!":

```
m_WParser->CloseContainer();
c = m_WParser->OpenContainer();

m_WParser->AddText("Hello, ");
m_WParser->AddText("world!");

m_WParser->CloseContainer();
m_WParser->OpenContainer();
```

and here is image of the situation:

You can see that there was an opened container before the code was executed. We closed it, created our own container, then closed our container and opened new container.

The result was that we had *same* depth level after executing. This is general rule that should be followed by tag handlers: leave depth level of containers unmodified (in other words, number of OpenContainer and CloseContainer calls should be same within wxHtmlTagHandler::HandleTag's body).

Notice that it would be usually better to use wxHtmlContainerCell::InsertCell instead of adding text to the parser directly.

10.46.6 Tag Handlers

The wxHTML library provides architecture of pluggable *tag* handlers. Tag handler is class that understands particular HTML tag (or tags) and is able to interpret it.

wxHtmlWinParser has a static table of **modules**. Each module contains one or more tag handlers. Each time a new wxHtmlWinParser object is constructed all modules are scanned and handlers are added to wxHtmlParser's list of available handlers (note: wxHtmlParser's list is non-static).

How it works

Common tag handler's wxHtmlTagHandler::HandleTag method works in four steps:

- Save state of parent parser into local variables
- Change parser state according to tag's params
- Parse text between the tag and paired ending tag (if present)
- Restore original parser state

See wxHtmlWinParser for methods for modifying parser's state. In general you can do things like opening/closing containers, changing colors, fonts etc.

Providing own tag handlers

You should create a new .cpp file and place the following lines into it:

```
#include <mod_templ.h>
#include <forcelink.h>
FORCE_LINK_ME(yourmodulefilenamewithoutcpp)
```

Then you must define handlers and one module.

Tag handlers

The handler is derived from wxHtmlWinTagHandler (or directly from wxHtmlTagHandler).

You can use set of macros to define the handler (see src/html/m_∗.cpp files for details). Handler definition must start with **TAG_HANDLER_BEGIN** macro and end with **TAG_HANDLER_END** macro.

I strongly recommend to have a look at *include/wxhtml/mod_templ.h* file. Otherwise you won't understand the structure of macros.

See macros reference:

- **TAG_HANDLER_BEGIN**(*name*, *tags*): Starts handler definition. *name* is handler identifier (in fact part of class name), *tags* is string containing list of tags supported by this handler (in uppercase). This macro derives new class from wxHtmlWinTagHandler and implements it is wxHtmlTagHandler::GetSupportedTags method. Example: TAG_HANDLER_BEGIN(FONTS, "B,I,U,T")

- **TAG_HANDLER_VARS:** This macro starts block of variables definitions. (Variables are identical to class attributes.) Example:

```
TAG_HANDLER_BEGIN(VARS_ONLY, "CRAZYTAG")
    TAG_HANDLER_VARS
        int my_int_var;
        wxString something_else;
TAG_HANDLER_END(VARS_ONLY)
```

This macro is used only in rare cases.

- **TAG_HANDLER_CONSTR**(*name*): This macro supplies object constructor. *name* is same name as the one from TAG_HANDLER_BEGIN macro. Body of constructor follow after this macro (you must use { and }). Example:

```
TAG_HANDLER_BEGIN(VARS2, "CRAZYTAG")
    TAG_HANDLER_VARS
        int my_int_var;
    TAG_HANDLER_CONSTR(vars2)
        { // !!!!!!
        my_int_var = 666;
        } // !!!!!!
TAG_HANDLER_END(VARS2)
```

Never used in wxHTML :-)

- **TAG_HANDLER_PROC**(*varib*): This is very important macro. It defines wxHtmlTagHandler::HandleTag method. *varib* is name of parameter passed to the method, usually *tag*. Body of method follows after this macro. Note than you must use { and } ! Example:

```
TAG_HANDLER_BEGIN(TITLE, "TITLE")
    TAG_HANDLER_PROC(tag)
        {
        printf("TITLE found...\n");
        }
TAG_HANDLER_END(TITLE)
```

- **TAG_HANDLER_END**(*name*): Ends definition of tag handler *name*.

Tags Modules

You can use set of 3 macros TAGS_MODULE_BEGIN, TAGS_MODULE_ADD and TAGS_MODULE_END to inherit new module from wxHtmlTagsModule and to create instance of it.

See macros reference:

- **TAGS_MODULE_BEGIN**(*modname*): Begins module definition. *modname* is part of class name and must be unique.
- **TAGS_MODULE_ADD**(*name*): Adds the handler to this module. *name* is the identifier from TAG_HANDLER_BEGIN.
- **TAGS_MODULE_END**(*modname*): Ends the definition of module. Example:

```
TAGS_MODULE_BEGIN(Examples)
    TAGS_MODULE_ADD(VARS_ONLY)
    TAGS_MODULE_ADD(VARS2)
    TAGS_MODULE_ADD(TITLE)
TAGS_MODULE_END(Examples)
```

10.46.7 Supported HTML Tags

wxHTML is not full implementation of HTML standard. Instead, it supports most common tags so that it is possible to display *simple* HTML documents with it. (For example it works fine with pages created in Netscape Composer or generated by tex2rtf).

Following tables list all tags known to wxHTML, together with supported parameters.

A tag has general form of `tagname` param_1 param_2 ... param_n where param_i is either `paramname="paramvalue"` or `paramname=paramvalue` - these two are equivalent. Unless stated otherwise, wxHTML is case-insensitive.

Common Parameter Values

We will use these substitutions in tags descriptions:

```
[alignment]     CENTER
                LEFT
                RIGHT
                JUSTIFY

[v_alignment]   TOP
                BOTTOM
                CENTER

[color]         HTML 4.0-compliant colour specification

[fontsize]      -2
                -1
                +0
                +1
                +2
                +3
                +4
                1
                2
                3
                4
                5
                6
                7

[pixels]        integer value that represents dimension in pixels

[percent]       i%
                where i is integer

[url]           an URL

[string]        text string

[coords]        c(1),c(2),c(3),...,c(n)
                where c(i) is integer
```

List of Supported Tags

```
A               NAME=[string]
                HREF=[url]
                TARGET=[target window spec]
ADDRESS
AREA            SHAPE=POLY
                SHAPE=CIRCLE
                SHAPE=RECT
                COORDS=[coords]
                HREF=[url]
B
BIG
BLOCKQUOTE
BODY            TEXT=[color]
                LINK=[color]
                BGCOLOR=[color]
BR              ALIGN=[alignment]
CENTER
CITE
CODE
DD
DIV             ALIGN=[alignment]
DL
DT
EM
FONT            COLOR=[color]
                SIZE=[fontsize]
                FACE=[comma-separated list of facenames]
HR              ALIGN=[alignment]
                SIZE=[pixels]
                WIDTH=[percent|pixels]
                NOSHADE
H1
H2
H3
H4
H5
H6
I
IMG             SRC=[url]
                WIDTH=[percent|pixels]
                HEIGHT=[pixels]
                ALIGN=TEXTTOP
                ALIGN=CENTER
                ALIGN=ABSCENTER
                ALIGN=BOTTOM
                USEMAP=[url]
KBD
LI
MAP             NAME=[string]
META            HTTP-EQUIV="Content-Type"
                CONTENT=[string]
OL
P               ALIGN=[alignment]
PRE
SAMP
SMALL
SPAN
STRIKE
STRONG
SUB
SUP
TABLE           ALIGN=[alignment]
                WIDTH=[percent|pixels]
                BORDER=[pixels]
                VALIGN=[v_alignment]
                BGCOLOR=[color]
                CELLSPACING=[pixels]
                CELLPADDING=[pixels]
TD              ALIGN=[alignment]
                VALIGN=[v_alignment]
                BGCOLOR=[color]
                WIDTH=[percent|pixels]
                COLSPAN=[pixels]
                ROWSPAN=[pixels]
                NOWRAP
TH              ALIGN=[alignment]
                VALIGN=[v_alignment]
                BGCOLOR=[color]
                WIDTH=[percent|pixels]
                COLSPAN=[pixels]
                ROWSPAN=[pixels]
TITLE
TR              ALIGN=[alignment]
                VALIGN=[v_alignment]
                BGCOLOR=[color]
```

```
TT
U
UL
```

Supported Styles

wxHTML doesn't really have CSS support but it does support a few simple styles: you can use "text-align", "width", "vertical-align" and "background" with all elements and for SPAN elements a few other styles are additionally recognized:

- color
- font-family
- font-size (only in point units)
- font-style (only "oblique", "italic" and "normal" values are supported)
- font-weight (only "bold" and "normal" values are supported)
- text-decoration (only "underline" value is supported)

10.47 wxRichTextCtrl Overview

wxRichTextCtrl provides a generic implementation of a rich text editor that can handle different character styles, paragraph formatting, and images.

It's aimed at editing 'natural' language text - if you need an editor that supports code editing, wxStyledTextCtrl is a better choice.

Despite its name, it cannot currently read or write RTF (rich text format) files. Instead, it uses its own XML format, and can also read and write plain text. In future we expect to provide RTF or OpenDocument file capabilities. Custom file formats can be supported by creating additional file handlers and registering them with the control.

wxRichTextCtrl is largely compatible with the wxTextCtrl API, but extends it where necessary. The control can be used where the native rich text capabilities of wxTextCtrl are not adequate (this is particularly true on Windows) and where more direct access to the content representation is required. It is difficult and inefficient to read the style information in a wxTextCtrl, whereas this information is readily available in wxRichTextCtrl. Since it's written in pure wxWidgets, any customizations you make to wxRichTextCtrl will be reflected on all platforms.

wxRichTextCtrl supports basic printing via the easy-to-use wxRichTextPrinting class. Creating applications with simple word processing features is simplified with the inclusion of wxRichTextFormattingDialog, a tabbed dialog allowing interactive tailoring of paragraph and character styling. Also provided is the multi-purpose dialog wxRich↩TextStyleOrganiserDialog that can be used for managing style definitions, browsing styles and applying them, or selecting list styles with a renumber option.

There are a few disadvantages to using wxRichTextCtrl. It is not native, so does not behave exactly as a native wxTextCtrl, although common editing conventions are followed. Users may miss the built-in spelling correction on Mac OS X, or any special character input that may be provided by the native control. It would also be a poor choice if intended users rely on screen readers that would be not work well with non-native text input implementation. You might mitigate this by providing the choice between wxTextCtrl and wxRichTextCtrl, with fewer features in the former case.

A good way to understand wxRichTextCtrl's capabilities is to compile and run the sample, samples/richtext, and browse the code.

10.47.1 Related Classes

Major classes: wxRichTextCtrl, wxRichTextBuffer, wxRichTextEvent

Helper classes: wxTextAttr, wxRichTextRange

File handler classes: wxRichTextFileHandler, wxRichTextHTMLHandler, wxRichTextXMLHandler

Style classes: wxRichTextCharacterStyleDefinition, wxRichTextParagraphStyleDefinition, wxRichTextListStyleDefinition, wxRichTextStyleSheet

Additional controls: wxRichTextStyleComboCtrl, wxRichTextStyleListBox, wxRichTextStyleListCtrl

Printing classes: wxRichTextPrinting, wxRichTextPrintout, wxRichTextHeaderFooterData

Dialog classes: wxRichTextStyleOrganiserDialog, wxRichTextFormattingDialog, wxSymbolPickerDialog

10.47.2 Code Example

The following code is an example taken from the sample, and adds text and styles to a rich text control programmatically.

```
wxRichTextCtrl* richTextCtrl = new wxRichTextCtrl(
    splitter, wxID_ANY, wxEmptyString, wxDefaultPosition,
    wxSize(200, 200), wxVSCROLL | wxHSCROLL |
      wxBORDER_NONE | wxWANTS_CHARS);

wxFont textFont = wxFont(12, wxROMAN, wxNORMAL, wxNORMAL);
wxFont boldFont = wxFont(12, wxROMAN, wxNORMAL, wxBOLD);
wxFont italicFont = wxFont(12, wxROMAN, wxITALIC, wxNORMAL);

wxFont font(12, wxROMAN, wxNORMAL, wxNORMAL);

m_richTextCtrl->SetFont(font);

wxRichTextCtrl& r = richTextCtrl;

r.BeginSuppressUndo();

r.BeginParagraphSpacing(0, 20);

r.BeginAlignment(wxTEXT_ALIGNMENT_CENTRE);
r.BeginBold();

r.BeginFontSize(14);
r.WriteText(wxT("Welcome to wxRichTextCtrl, a wxWidgets control for editing and presenting
       styled text and images"));
r.EndFontSize();
r.Newline();

r.BeginItalic();
r.WriteText(wxT("by Julian Smart"));
r.EndItalic();

r.EndBold();

r.Newline();
r.WriteImage(wxBitmap(zebra_xpm));

r.EndAlignment();

r.Newline();
r.Newline();

r.WriteText(wxT("What can you do with this thing? "));
r.WriteImage(wxBitmap(smiley_xpm));
r.WriteText(wxT(" Well, you can change text "));

r.BeginTextColour(wxColour(255, 0, 0));
r.WriteText(wxT("colour, like this red bit."));
r.EndTextColour();

r.BeginTextColour(wxColour(0, 0, 255));
r.WriteText(wxT(" And this blue bit."));
r.EndTextColour();

r.WriteText(wxT(" Naturally you can make things "));
r.BeginBold();
r.WriteText(wxT("bold "));
r.EndBold();
r.BeginItalic();
r.WriteText(wxT("or italic "));
r.EndItalic();
r.BeginUnderline();
r.WriteText(wxT("or underlined."));
r.EndUnderline();
```

```
r.BeginFontSize(14);
r.WriteText(wxT(" Different font sizes on the same line is allowed, too."));
r.EndFontSize();

r.WriteText(wxT(" Next we'll show an indented paragraph."));

r.BeginLeftIndent(60);
r.Newline();

r.WriteText(wxT("Indented paragraph."));
r.EndLeftIndent();

r.Newline();

r.WriteText(wxT("Next, we'll show a first-line indent, achieved using BeginLeftIndent(100,
       -40)."));

r.BeginLeftIndent(100, -40);
r.Newline();

r.WriteText(wxT("It was in January, the most down-trodden month of an Edinburgh winter."));
r.EndLeftIndent();

r.Newline();

r.WriteText(wxT("Numbered bullets are possible, again using subindents:"));

r.BeginNumberedBullet(1, 100, 60);
r.Newline();

r.WriteText(wxT("This is my first item. Note that wxRichTextCtrl doesn't automatically do
       numbering, but this will be added later."));
r.EndNumberedBullet();

r.BeginNumberedBullet(2, 100, 60);
r.Newline();

r.WriteText(wxT("This is my second item."));
r.EndNumberedBullet();

r.Newline();

r.WriteText(wxT("The following paragraph is right-indented:"));

r.BeginRightIndent(200);
r.Newline();

r.WriteText(wxT("It was in January, the most down-trodden month of an Edinburgh winter. An
       attractive woman came into the cafe, which is nothing remarkable."));
r.EndRightIndent();

r.Newline();

wxArrayInt tabs;
tabs.Add(400);
tabs.Add(600);
tabs.Add(800);
tabs.Add(1000);
wxTextAttr attr;
attr.SetFlags(wxTEXT_ATTR_TABS);
attr.SetTabs(tabs);
r.SetDefaultStyle(attr);

r.WriteText(wxT("This line contains tabs:\tFirst tab\tSecond tab\tThird tab"));

r.Newline();
r.WriteText(wxT("Other notable features of wxRichTextCtrl include:"));

r.BeginSymbolBullet(wxT('*'), 100, 60);
r.Newline();
r.WriteText(wxT("Compatibility with wxTextCtrl API"));
r.EndSymbolBullet();

r.WriteText(wxT("Note: this sample content was generated programmatically from within the
       MyFrame constructor in the demo. The images were loaded from inline XPMs. Enjoy wxRichTextCtrl!"));

r.EndSuppressUndo();
```

10.47.3 Starting to Use wxRichTextCtrl

You need to include <wx/richtext/richtextctrl.h> in your source, and link with the appropriate wx↩ Widgets library with `richtext` suffix. Put the rich text library first in your link line to avoid unresolved symbols.

Then you can create a wxRichTextCtrl, with the wxWANT_CHARS style if you want tabs to be processed by the control rather than being used for navigation between controls.

10.47.4 Text Styles

Styling attributes are represented by wxTextAttr, or for more control over attributes such as margins and size, the derived class wxRichTextAttr.

When setting a style, the flags of the attribute object determine which attributes are applied. When querying a style, the passed flags are ignored except (optionally) to determine whether attributes should be retrieved from character content or from the paragraph object.

wxRichTextCtrl takes a layered approach to styles, so that different parts of the content may be responsible for contributing different attributes to the final style you see on the screen.

There are four main notions of style within a control:

- **Basic style**: The fundamental style of a control, onto which any other styles are layered. It provides default attributes, and changing the basic style may immediately change the look of the content depending on what other styles the content uses. Calling wxRichTextCtrl::SetFont changes the font for the basic style. The basic style is set with wxRichTextCtrl::SetBasicStyle.
- **Paragraph style**: Each paragraph has attributes that are set independently from other paragraphs and independently from the content within the paragraph. Normally, these attributes are paragraph-related, such as alignment and indentation, but it is possible to set character attributes too. The paragraph style can be set independently of its content by passing wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY to wxRichTextCtrl::SetStyleEx.
- **Character style**: Characters within each paragraph can have attributes. A single character, or a run of characters, can have a particular set of attributes. The character style can be with wxRichTextCtrl::SetStyle or wxRichTextCtrl::SetStyleEx.
- **Default style**: This is the 'current' style that determines the style of content that is subsequently typed, pasted or programmatically inserted. The default style is set with wxRichTextCtrl::SetDefaultStyle.

What you see on the screen is the dynamically *combined* style, found by merging the first three of the above style types (the fourth is only a guide for future content insertion and therefore does not affect the currently displayed content).

To make all this more concrete, here are examples of where you might set these different styles:

- You might set the *basic style* to have a Times Roman font in 12 point, left-aligned, with two millimetres of spacing after each paragraph.
- You might set the *paragraph style* (for one particular paragraph) to be centred.
- You might set the *character style* of one particular word to bold.
- You might set the *default style* to be underlined, for subsequent inserted text.

Naturally you can do any of these things either using your own UI, or programmatically.

The basic wxTextCtrl doesn't make the same distinctions as wxRichTextCtrl regarding attribute storage. So we need finer control when setting and retrieving attributes. wxRichTextCtrl::SetStyleEx takes a *flags* parameter:

- wxRICHTEXT_SETSTYLE_OPTIMIZE specifies that the style should be changed only if the combined attributes are different from the attributes for the current object. This is important when applying styling that has been edited by the user, because he has just edited the *combined* (visible) style, and wxRichTextCtrl wants to leave unchanged attributes associated with their original objects instead of applying them to both paragraph and content objects.

- wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY specifies that only paragraph objects within the given range should take on the attributes.
- wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY specifies that only content objects (text or images) within the given range should take on the attributes.
- wxRICHTEXT_SETSTYLE_WITH_UNDO specifies that the operation should be undoable.

It's great to be able to change arbitrary attributes in a wxRichTextCtrl, but it can be unwieldy for the user or programmer to set attributes separately. Word processors have collections of styles that you can tailor or use as-is, and this means that you can set a heading with one click instead of marking text in bold, specifying a large font size, and applying a certain paragraph spacing and alignment for every such heading. Similarly, wxWidgets provides a class called wxRichTextStyleSheet which manages style definitions (wxRichTextParagraphStyleDefinition, wxRichTextListStyleDefinition and wxRichTextCharacterStyleDefinition). Once you have added definitions to a style sheet and associated it with a wxRichTextCtrl, you can apply a named definition to a range of text. The classes wxRichTextStyleComboCtrl and wxRichTextStyleListBox can be used to present the user with a list of styles in a sheet, and apply them to the selected text.

You can reapply a style sheet to the contents of the control, by calling wxRichTextCtrl::ApplyStyleSheet. This is useful if the style definitions have changed, and you want the content to reflect this. It relies on the fact that when you apply a named style, the style definition name is recorded in the content. So ApplyStyleSheet works by finding the paragraph attributes with style names and re-applying the definition's attributes to the paragraph. Currently, this works with paragraph and list style definitions only.

10.47.5 Included Dialogs

wxRichTextCtrl comes with standard dialogs to make it easier to implement text editing functionality.

wxRichTextFormattingDialog can be used for character or paragraph formatting, or a combination of both. It's a wxPropertySheetDialog with the following available tabs: Font, Indents & Spacing, Tabs, Bullets, Style, Borders, Margins, Background, Size, and List Style. You can select which pages will be shown by supplying flags to the dialog constructor. In a character formatting dialog, typically only the Font page will be shown. In a paragraph formatting dialog, you'll show the Indents & Spacing, Tabs and Bullets pages. The Style tab is useful when editing a style definition.

You can customize this dialog by providing your own wxRichTextFormattingDialogFactory object, which tells the formatting dialog how many pages are supported, what their identifiers are, and how to creates the pages.

wxRichTextStyleOrganiserDialog is a multi-purpose dialog that can be used for managing style definitions, browsing styles and applying them, or selecting list styles with a renumber option. See the sample for usage - it is used for the "Manage Styles" and "Bullets and Numbering" menu commands.

wxSymbolPickerDialog lets the user insert a symbol from a specified font. It has no wxRichTextCtrl dependencies besides being included in the rich text library.

10.47.6 How wxRichTextCtrl is Implemented

Data representation is handled by wxRichTextBuffer, and a wxRichTextCtrl always has one such buffer.

The content is represented by a hierarchy of objects, all derived from wxRichTextObject. An object might be an image, a fragment of text, a paragraph, or a further composite object. Objects store a wxRichTextAttr containing style information; a paragraph object can contain both paragraph and character information, but content objects such as text can only store character information. The final style displayed in the control or in a printout is a combination of base style, paragraph style and content (character) style.

The top of the hierarchy is the buffer, a kind of wxRichTextParagraphLayoutBox, containing further wxRichTextParagraph objects, each of which can include text, images and potentially other types of object.

Each object maintains a range (start and end position) measured from the start of the main parent object.

When Layout is called on an object, it is given a size which the object must limit itself to, or one or more flexible directions (vertical or horizontal). So, for example, a centred paragraph is given the page width to play with (minus any

margins), but can extend indefinitely in the vertical direction. The implementation of Layout caches the calculated size and position.

When the buffer is modified, a range is invalidated (marked as requiring layout), so that only the minimum amount of layout is performed.

A paragraph of pure text with the same style contains just one further object, a wxRichTextPlainText object. When styling is applied to part of this object, the object is decomposed into separate objects, one object for each different character style. So each object within a paragraph always has just one wxTextAttr object to denote its character style. Of course, this can lead to fragmentation after a lot of edit operations, potentially leading to several objects with the same style where just one would do. So a Defragment function is called when updating the control's display, to ensure that the minimum number of objects is used.

10.47.7 Nested Objects

wxRichTextCtrl supports nested objects such as text boxes and tables. To achieve compatibility with the existing API, there is the concept of *object focus*. When the user clicks on a nested text box, the object focus is set to that container object so all keyboard input and API functions apply to that container. The application can change the focus using wxRichTextCtrl::SetObjectFocus. Call this function with a `null` parameter to set the focus back to the top-level object.

An event will be sent to the control when the focus changes.

When the user clicks on the control, wxRichTextCtrl determines which container to set as the current object focus by calling the found container's overrided wxRichTextObject::AcceptsFocus function. For example, although a table is a container, it must not itself be the object focus because there is no text editing at the table level. Instead, a cell within the table must accept the focus.

Since with nested objects it is not possible to represent a section with merely a start position and an end position, the class wxRichTextSelection is provided which stores multiple ranges (for non-contiguous selections such as table cells) and a pointer to the container object in question. You can pass wxRichTextSelection to wxRichTextCtrl::SetSelection or get an instance of it from wxRichTextCtrl::GetSelection.

When selecting multiple objects, such as cell tables, the wxRichTextCtrl dragging handler code calls the function wxRichTextObject::HandlesChildSelections to determine whether the children can be individual selections. Currently only table cells can be multiply-selected in this way.

10.47.8 Context Menus and Property Dialogs

There are three ways you can make use of context menus: you can let wxRichTextCtrl handle everything and provide a basic menu; you can set your own context menu using wxRichTextCtrl::SetContextMenu but let wxRichTextCtrl handle showing it and adding property items; or you can override the default context menu behaviour by adding a context menu event handler to your class in the normal way.

If you right-click over a text box in cell in a table, you may want to edit the properties of one of these objects - but which properties will you be editing?

Well, the default behaviour allows up to three property-editing menu items simultaneously - for the object clicked on, the container of that object, and the container's parent (depending on whether any of these objects return true from their wxRichTextObject::CanEditProperties functions). If you supply a context menu, add a property command item using the wxID_RICHTEXT_PROPERTIES1 identifier, so that wxRichTextCtrl can find the position to add command items. The object should tell the control what label to use by returning a string from wxRichTextObject::GetPropertiesMenuLabel.

Since there may be several property-editing commands showing, it is recommended that you don't include the word Properties - just the name of the object, such as Text Box or Table.

10.47.9 Development Roadmap

Bugs

This is an incomplete list of bugs.

- Moving the caret up at the beginning of a line sometimes incorrectly positions the caret.
- As the selection is expanded, the text jumps slightly due to kerning differences between drawing a single text string versus drawing several fragments separately. This could be improved by using wxDC::GetPartialTextExtents to calculate exactly where the separate fragments should be drawn. Note that this problem also applies to separation of text fragments due to difference in their attributes.

Features

This is a list of some of the features that have yet to be implemented. Help with them will be appreciated.

- support for composite objects in some functions where it's not yet implemented, for example ApplyStyleSheet
- Table API enhancements and dialogs; improved table layout especially row spans and fitting
- Conversion from HTML, and a rewrite of the HTML output handler that includes CSS, tables, text boxes, and floating images, in addition to a simplified-HTML mode for wxHTML compatibility
- Open Office input and output
- RTF input and output
- A ruler control
- Standard editing toolbars
- Bitmap bullets
- Justified text, in print/preview at least
- scaling: either everything scaled, or rendering using a custom reference point size and an optional dimension scale

There are also things that could be done to take advantage of the underlying text capabilities of the platform; higher-level text formatting APIs are available on some platforms, such as Mac OS X, and some of translation from high level to low level wxDC API is unnecessary. However this would require additions to the wxWidgets API.

10.48 wxAUI Overview

wxAUI stands for Advanced User Interface.

It aims to give the user a cutting edge interface with floatable windows, and a user-customizable layout. The original wxAUI sources have kindly been made available under the wxWindows licence by Kirix Corp. and they have since then been integrated into wxWidgets and further improved.

See also

Window Docking (wxAUI)

10.48.1 Frame Management

Frame management provides the means to open, move and hide common controls that are needed to interact with the document, and allow these configurations to be saved into different perspectives and loaded at a later time.

10.48.2 Toolbars

Toolbars are a specialized subset of the frame management system and should behave similarly to other docked components. However, they also require additional functionality, such as "spring-loaded" rebar support, "chevron" buttons and end-user customizability.

10.48.3 Modeless Controls

Modeless controls expose a tool palette or set of options that float above the application content while allowing it to be accessed. Usually accessed by the toolbar, these controls disappear when an option is selected, but may also be "torn off" the toolbar into a floating frame of their own.

10.48.4 Look and Feel

Look and feel encompasses the way controls are drawn, both when shown statically as well as when they are being moved. This aspect of user interface design incorporates "special effects" such as transparent window dragging as well as frame animation.

wxAUI adheres to the following principles: Use native floating frames to obtain a native look and feel for all platforms. Use existing wxWidgets code where possible, such as sizer implementation for frame management. Use classes included in wxCore and wxBase only.

10.49 wxPropertyGrid Overview

wxPropertyGrid is a specialized grid for editing properties - in other words name = value pairs.

List of ready-to-use property classes include strings, numbers, flag sets, fonts, colours and many others. It is possible, for example, to categorize properties, set up a complete tree-hierarchy, add more than two columns, and set arbitrary per-property attributes.

As this version of wxPropertyGrid has some backward-incompatible changes from version 1.4, everybody who need to maintain custom property classes should carefully read final section in Changes from wxPropertyGrid 1.4.

See also

wxPropertyGrid, wxPropertyGridEvent, wxPropertyGridManager, wxPropertyGridPage, wxPGProperty

10.49.1 Creating and Populating wxPropertyGrid

As seen here, wxPropertyGrid is constructed in the same way as other wxWidgets controls:

```
// Necessary header file
#include <wx/propgrid/propgrid.h>

...

    // Assumes code is in frame/dialog constructor

    // Construct wxPropertyGrid control
    wxPropertyGrid* pg = new wxPropertyGrid(
        this, // parent
        PGID, // id
        wxDefaultPosition, // position
        wxDefaultSize, // size
        // Here are just some of the supported window styles
        wxPG_AUTO_SORT | // Automatic sorting after items added
        wxPG_SPLITTER_AUTO_CENTER | // Automatically center splitter until user
       manually adjusts it
        // Default style
        wxPG_DEFAULT_STYLE );

    // Window style flags are at premium, so some less often needed ones are
    // available as extra window styles (wxPG_EX_xxx) which must be set using
```

```
    // SetExtraStyle member function. wxPG_EX_HELP_AS_TOOLTIPS, for instance,
    // allows displaying help strings as tool tips.
    pg->SetExtraStyle( wxPG_EX_HELP_AS_TOOLTIPS );
```

(for complete list of new window styles, see wxPropertyGrid Window Styles)

wxPropertyGrid is usually populated with lines like this:

```
pg->Append( new wxStringProperty("Label", "Name", "Initial Value") );
```

Naturally, wxStringProperty is a property class. Only the first function argument (label) is mandatory. Second one, name, defaults to label and, third, the initial value, to default value. If constant wxPG_LABEL is used as the name argument, then the label is automatically used as a name as well (this is more efficient than manually defining both as the same). Use of empty name is discouraged and will sometimes result in run-time error. Note that all property class constructors have quite similar constructor argument list.

To demonstrate other common property classes, here's another code snippet:

```
// Add int property
pg->Append( new wxIntProperty("IntProperty", wxPG_LABEL, 12345678) );

// Add float property (value type is actually double)
pg->Append( new wxFloatProperty("FloatProperty", wxPG_LABEL, 12345.678) );

// Add a bool property
pg->Append( new wxBoolProperty("BoolProperty", wxPG_LABEL, false) );

// A string property that can be edited in a separate editor dialog.
pg->Append( new wxLongStringProperty("LongStringProperty",
                                     wxPG_LABEL,
                                     "This is much longer string than the "
                                     "first one. Edit it by clicking the button."));

// String editor with dir selector button.
pg->Append( new wxDirProperty("DirProperty", wxPG_LABEL, ::wxGetUserHome()) );

// wxArrayStringProperty embeds a wxArrayString.
pg->Append( new wxArrayStringProperty("Label of ArrayStringProperty",
                                      "NameOfArrayStringProp"));

// A file selector property.
pg->Append( new wxFileProperty("FileProperty", wxPG_LABEL, wxEmptyString) );

// Extra: set wild card for file property (format same as in wxFileDialog).
pg->SetPropertyAttribute( "FileProperty",
                          wxPG_FILE_WILDCARD,
                          "All files (*.*)|*.*" );
```

Operations on properties are usually done by directly calling wxPGProperty's or wxPropertyGridInterface's member functions. wxPropertyGridInterface is an abstract base class for property containers such as wxPropertyGrid, wxPropertyGridManager, and wxPropertyGridPage. Note however that wxPGProperty's member functions generally do not refresh the grid.

wxPropertyGridInterface's property operation member functions , such as SetPropertyValue() and DisableProperty(), all accept a special wxPGPropArg id argument, using which you can refer to properties either by their pointer (for performance) or by their name (for convenience). For instance:

```
// Add a file selector property.
wxPGProperty* prop = pg->Append( new wxFileProperty("FileProperty",
                                 wxPG_LABEL,
                                 wxEmptyString) );

// Valid: Set wild card by name
pg->SetPropertyAttribute( "FileProperty",
                          wxPG_FILE_WILDCARD,
                          "All files (*.*)|*.*" );

// Also Valid: Set wild card by property pointer
pg->SetPropertyAttribute( prop,
                          wxPG_FILE_WILDCARD,
                          "All files (*.*)|*.*" );
```

Using pointer is faster, since it doesn't require hash map lookup. Anyway, you can always get property pointer (wxPGProperty*) as return value from Append() or Insert(), or by calling wxPropertyGridInterface::GetPropertyByName() or just plain GetProperty().

10.49.2 Categories

wxPropertyGrid has a hierarchic property storage and display model, which allows property categories to hold child properties and even other categories. Other than that, from the programmer's point of view, categories can be treated exactly the same as "other" properties. For example, despite its name, GetPropertyByName() also returns a category by name. Note however that sometimes the label of a property category may be referred as caption (for example, there is wxPropertyGrid::SetCaptionTextColour() method that sets text colour of property category labels).

When category is added at the top (i.e. root) level of the hierarchy, it becomes a *current category*. This means that all other (non-category) properties after it are automatically appended to it. You may add properties to specific categories by using wxPropertyGridInterface::Insert or wxPropertyGridInterface::AppendIn.

Category code sample:

```
// One way to add category (similar to how other properties are added)
pg->Append( new wxPropertyCategory("Main") );

// All these are added to "Main" category
pg->Append( new wxStringProperty("Name") );
pg->Append( new wxIntProperty("Age",wxPG_LABEL,25) );
pg->Append( new wxIntProperty("Height",wxPG_LABEL,180) );
pg->Append( new wxIntProperty("Weight") );

// Another one
pg->Append( new wxPropertyCategory("Attributes") );

// All these are added to "Attributes" category
pg->Append( new wxIntProperty("Intelligence") );
pg->Append( new wxIntProperty("Agility") );
pg->Append( new wxIntProperty("Strength") );
```

10.49.3 Tree-like Property Structure

Basically any property can have children. There are few limitations, however.

Remarks

- Names of properties with non-category, non-root parents are not stored in global hash map. Instead, they can be accessed with strings like "Parent.Child". For instance, in the sample below, child property named "Max. Speed (mph)" can be accessed by global name "Car.Speeds.Max Speed (mph)".
- If you want to property's value to be a string composed of the child property values, you must use wxStringProperty as parent and use magic string "<composed>" as its value.
- Events (eg. change of value) that occur in parent do not propagate to children. Events that occur in children will propagate to parents, but only if they are wxStringProperties with "<composed>" value.

Sample:

```
wxPGProperty* carProp = pg->Append(new wxStringProperty("Car",
                                   wxPG_LABEL,
                                   "<composed>"));

pg->AppendIn(carProp, new wxStringProperty("Model",
                                           wxPG_LABEL,
                                           "Lamborghini Diablo SV"));

pg->AppendIn(carProp, new wxIntProperty("Engine Size (cc)",
                                        wxPG_LABEL,
                                        5707) );

wxPGProperty* speedsProp = pg->AppendIn(carProp,
                                        new wxStringProperty("Speeds",
                                          wxPG_LABEL,
                                          "<composed>"));

pg->AppendIn( speedsProp, new wxIntProperty("Max. Speed (mph)",
                                            wxPG_LABEL,290) );
pg->AppendIn( speedsProp, new wxFloatProperty("0-100 mph (sec)",
                                              wxPG_LABEL,3.9) );
pg->AppendIn( speedsProp, new wxFloatProperty("1/4 mile (sec)",
                                              wxPG_LABEL,8.6) );
```

```
// This is how child property can be referred to by name
pg->SetPropertyValue( "Car.Speeds.Max. Speed (mph)", 300 );

pg->AppendIn(carProp, new wxIntProperty("Price ($)",
                                        wxPG_LABEL,
                                        300000) );

// Displayed value of "Car" property is now very close to this:
// "Lamborghini Diablo SV; 5707 [300; 3.9; 8.6] 300000"
```

10.49.4 wxEnumProperty and wxFlagsProperty

wxEnumProperty is used when you want property's (integer or string) value to be selected from a popup list of choices.

Creating wxEnumProperty is slightly more complex than those described earlier. You have to provide list of constant labels, and optionally relevant values (if label indexes are not sufficient).

Remarks

- Value wxPG_INVALID_VALUE (equals INT_MAX) is not allowed as list item value.

A very simple example:

```
//
// Using wxArrayString
//
wxArrayString arrDiet;
arr.Add("Herbivore");
arr.Add("Carnivore");
arr.Add("Omnivore");

pg->Append( new wxEnumProperty("Diet",
                               wxPG_LABEL,
                               arrDiet) );

//
// Using wxChar* array
//
const wxChar* arrayDiet[] =
{ wxT("Herbivore"), wxT("Carnivore"), wxT("Omnivore"), NULL };

pg->Append( new wxEnumProperty("Diet",
                               wxPG_LABEL,
                               arrayDiet) );
```

Here's extended example using values as well:

```
//
// Using wxArrayString and wxArrayInt
//
wxArrayString arrDiet;
arr.Add("Herbivore");
arr.Add("Carnivore");
arr.Add("Omnivore");

wxArrayInt arrIds;
arrIds.Add(40);
arrIds.Add(45);
arrIds.Add(50);

// Note that the initial value (the last argument) is the actual value,
// not index or anything like that. Thus, our value selects "Omnivore".
pg->Append( new wxEnumProperty("Diet",
                               wxPG_LABEL,
                               arrDiet,
                               arrIds,
                               50));
```

wxPGChoices is a class where wxEnumProperty, and other properties which require storage for list of items, actually stores strings and values. It is used to facilitate reference counting, and therefore recommended way of adding items when multiple properties share the same set.

You can use wxPGChoices directly as well, filling it and then passing it to the constructor. In fact, if you wish to display bitmaps next to labels, your best choice is to use this approach.

```
wxPGChoices chs;
chs.Add("Herbivore", 40);
chs.Add("Carnivore", 45);
chs.Add("Omnivore", 50);

// Let's add an item with bitmap, too
chs.Add("None of the above", wxBitmap(), 60);

pg->Append( new wxEnumProperty("Primary Diet",
                               wxPG_LABEL,
                               chs) );

// Add same choices to another property as well - this is efficient due
// to reference counting
pg->Append( new wxEnumProperty("Secondary Diet",
                               wxPG_LABEL,
                               chs) );
```

You can later change choices of property by using wxPGProperty::AddChoice(), wxPGProperty::InsertChoice(), wxPGProperty::DeleteChoice(), and wxPGProperty::SetChoices().

wxEditEnumProperty works exactly like wxEnumProperty, except is uses non-read-only combo box as default editor, and value is stored as string when it is not any of the choices.

wxFlagsProperty has similar construction:

```
const wxChar* flags_prop_labels[] = { wxT("wxICONIZE"),
    wxT("wxCAPTION"), wxT("wxMINIMIZE_BOX"), wxT("wxMAXIMIZE_BOX"), NULL };

// this value array would be optional if values matched string indexes
long flags_prop_values[] = { wxICONIZE, wxCAPTION,
      wxMINIMIZE_BOX,
    wxMAXIMIZE_BOX };

pg->Append( new wxFlagsProperty("Window Style",
                                wxPG_LABEL,
                                flags_prop_labels,
                                flags_prop_values,
                                wxDEFAULT_FRAME_STYLE) );
```

wxFlagsProperty can use wxPGChoices just the same way as wxEnumProperty **Note:** When changing "choices" (ie. flag labels) of wxFlagsProperty, you will need to use wxPGProperty::SetChoices() to replace all choices at once - otherwise implicit child properties will not get updated properly.

10.49.5 Specialized Properties

This section describes the use of less often needed property classes. To use them, you have to include <wx/propgrid/advprops.h>.

```
// Necessary extra header file
#include <wx/propgrid/advprops.h>

...

    // Date property.
    pg->Append( new wxDateProperty("MyDateProperty",
                                   wxPG_LABEL,
                                   wxDateTime::Now()) );

    // Image file property. Wild card is auto-generated from available
    // image handlers, so it is not set this time.
    pg->Append( new wxImageFileProperty("Label of ImageFileProperty",
                                        "NameOfImageFileProp") );

    // Font property has sub-properties. Note that we give window's font as
    // initial value.
    pg->Append( new wxFontProperty("Font",
                                   wxPG_LABEL,
                                   GetFont()) );

    // Colour property with arbitrary colour.
    pg->Append( new wxColourProperty("My Colour 1",
```

```
                                    wxPG_LABEL,
                                    wxColour(242,109,0) ) );

    // System colour property.
    pg->Append( new wxSystemColourProperty("My SysColour 1",
                                           wxPG_LABEL,
                                           wxSystemSettings::GetColour(
      wxSYS_COLOUR_WINDOW)) );

    // System colour property with custom colour.
    pg->Append( new wxSystemColourProperty("My SysColour 2",
                                           wxPG_LABEL,
                                           wxColour(0,200,160) ) );

    // Cursor property
    pg->Append( new wxCursorProperty("My Cursor",
                                     wxPG_LABEL,
                                     wxCURSOR_ARROW));
```

10.49.6 Processing Property Values

Properties store their values internally as wxVariant, but is also possible to obtain them as wxAny, using implicit conversion. You can get property values with wxPGProperty::GetValue() and wxPropertyGridInterface::GetProperty↩Value().

Below is a code example which handles wxEVT_PG_CHANGED event:

```
void MyWindowClass::OnPropertyGridChanged(wxPropertyGridEvent& event)
{
    wxPGProperty* property = event.GetProperty();

    // Do nothing if event did not have associated property
    if ( !property )
        return;

    // GetValue() returns wxVariant, but it is converted transparently to
    // wxAny
    wxAny value = property->GetValue();

    // Also, handle the case where property value is unspecified
    if ( value.IsNull() )
        return;

    // Handle changes in values, as needed
    if ( property->GetName() == "MyStringProperty" )
        OnMyStringPropertyChanged(value.As<wxString>());
    else if ( property->GetName() == "MyColourProperty" )
        OnMyColourPropertyChanged(value.As<wxColour>());
}
```

You can get a string-representation of property's value using wxPGProperty::GetValueAsString() or wxProperty↩GridInterface::GetPropertyValueAsString(). This particular function is very safe to use with any kind of property.

Note

There is a one case in which you may want to take extra care when dealing with raw wxVariant values. That is, integer-type properties, such as wxIntProperty and wxUIntProperty, store value internally as wx(U)LongLong when number doesn't fit into standard long type. Using << operator to get wx(U)LongLong from wxVariant is customized to work quite safely with various types of variant data. However, you can also bypass this problem by using wxAny in your code instead of wxVariant.

Note that in some cases property value can be Null variant, which means that property value is unspecified. This usually occurs only when wxPG_EX_AUTO_UNSPECIFIED_VALUES extra window style is defined or when you manually set property value to Null (or unspecified).

10.49.7 Iterating through a property container

You can use somewhat STL'ish iterator classes to iterate through the grid. Here is a simple example of forward iterating through all individual properties (not categories nor private child properties that are normally 'transparent' to application code):

```
wxPropertyGridIterator it;

for ( it = pg->GetIterator();
      !it.AtEnd();
      it++ )
{
    wxPGProperty* p = *it;
    // Do something with the property
}
```

As expected there is also a const iterator:

```
wxPropertyGridConstIterator it;

for ( it = pg->GetIterator();
      !it.AtEnd();
      it++ )
{
    const wxPGProperty* p = *it;
    // Do something with the property
}
```

You can give some arguments to GetIterator to determine which properties get automatically filtered out. For complete list of options, see wxPropertyGridIterator Flags. GetIterator() also accepts other arguments. See wxPropertyGridInterface::GetIterator() for details.

This example reverse-iterates through all visible items:

```
wxPropertyGridIterator it;

for ( it = pg->GetIterator(wxPG_ITERATE_VISIBLE,
      wxBOTTOM);
      !it.AtEnd();
      it-- )
{
    wxPGProperty* p = *it;
    // Do something with the property
}
```

GetIterator() only works with wxPropertyGrid and the individual pages of wxPropertyGridManager. In order to iterate through an arbitrary property container (such as entire wxPropertyGridManager), you need to use wxPropertyGridInterface::GetVIterator(). Note however that this virtual iterator is limited to forward iteration.

```
wxPGVIterator it;

for ( it = manager->GetVIterator(wxPG_ITERATE_ALL);
      !it.AtEnd();
      it.Next() )
{
    wxPGProperty* p = it.GetProperty();
    // Do something with the property
}
```

10.49.8 Populating wxPropertyGrid Automatically

Populating from List of wxVariants

Example of populating an empty wxPropertyGrid from a values stored in an arbitrary list of wxVariants.

```
// This is a static method that initializes *all* built-in type handlers
// available, including those for wxColour and wxFont. Refers to *all*
// included properties, so when compiling with static library, this
// method may increase the executable size noticeably.
pg->InitAllTypeHandlers();

// Get contents of the grid as a wxVariant list
wxVariant all_values = pg->GetPropertyValues();

// Populate the list with values. If a property with appropriate
// name is not found, it is created according to the type of variant.
pg->SetPropertyValues( my_list_variant );
```

Loading Population from a Text-based Storage

Class wxPropertyGridPopulator may be helpful when writing code that loads properties from a text-source. In fact, the wxPropertyGrid xrc-handler (which may not be currently included in wxWidgets, but probably will be in near future) uses it.

Saving and Restoring User-Editable State

You can use wxPropertyGridInterface::SaveEditableState() and wxPropertyGridInterface::RestoreEditableState() to save and restore user-editable state (selected property, expanded/collapsed properties, selected page, scrolled position, and splitter positions).

10.49.9 Event Handling

Probably the most important event is the Changed event which occurs when value of any property is changed by the user. Use EVT_PG_CHANGED(id,func) in your event table to use it.

For complete list of event types, see wxPropertyGrid class reference.

However, one type of event that might need focused attention is EVT_PG_CHANGING, which occurs just prior property value is being changed by user. You can acquire pending value using wxPropertyGridEvent::GetValue(), and if it is not acceptable, call wxPropertyGridEvent::Veto() to prevent the value change from taking place.

```
void MyForm::OnPropertyGridChanging( wxPropertyGridEvent& event )
{
    wxPGProperty* property = event.GetProperty();

    if ( property == m_pWatchThisProperty )
    {
        // GetValue() returns the pending value, but is only
        // supported by wxEVT_PG_CHANGING.
        if ( event.GetValue().GetString() == g_pThisTextIsNotAllowed )
        {
            event.Veto();
            return;
        }
    }
}
```

Remarks

On Child Property Event Handling

- For properties which have private, implicit children (wxFontProperty and wxFlagsProperty), events occur for the main parent property only. For other properties events occur for the children themselves. See Tree-like Property Structure.
- When property's child gets changed, you can use wxPropertyGridEvent::GetMainParent() to obtain its topmost non-category parent (useful, if you have deeply nested properties).

10.49.10 Help String, Hint and Tool Tips

For each property you can specify two different types of help text. First, you can use wxPropertyGridInterface::SetPropertyHelpString() or wxPGProperty::SetHelpString() to set property's help text. Second, you can use wxPGProperty::SetAttribute() to set property's "Hint" attribute.

Difference between hint and help string is that the hint is shown in an empty property value cell, while help string is shown either in the description text box, as a tool tip, or on the status bar, whichever of these is available.

To enable display of help string as tool tips, you must explicitly use the wxPG_EX_HELP_AS_TOOLTIPS extra window style.

10.49.11 Validating Property Values

There are various ways to make sure user enters only correct values. First, you can use wxValidators similar to as you would with ordinary controls. Use wxPropertyGridInterface::SetPropertyValidator() to assign wxValidator to property.

Second, you can subclass a property and override wxPGProperty::ValidateValue(), or handle wxEVT_PG_CHANGING for the same effect. Both of these ways do not actually prevent user from temporarily entering invalid text, but they do give you an opportunity to warn the user and block changed value from being committed in a property.

Various validation failure options can be controlled globally with wxPropertyGrid::SetValidationFailureBehavior(), or on an event basis by calling wxEvent::SetValidationFailureBehavior(). Here's a code snippet of how to handle wxEVT_PG_CHANGING, and to set custom failure behaviour and message.

```
void MyFrame::OnPropertyGridChanging(wxPropertyGridEvent& event)
{
    wxPGProperty* property = event.GetProperty();

    // You must use wxPropertyGridEvent::GetValue() to access
    // the value to be validated.
    wxVariant pendingValue = event.GetValue();

    if ( property->GetName() == "Font" )
    {
        // Make sure value is not unspecified
        if ( !pendingValue.IsNull() )
        {
            wxFont font;
            font << pendingValue;

            // Let's just allow Arial font
            if ( font.GetFaceName() != "Arial" )
            {
                event.Veto();
                event.SetValidationFailureBehavior(wxPG_VFB_STAY_IN_PROPERTY |
                                                   wxPG_VFB_BEEP |
                                                   wxPG_VFB_SHOW_MESSAGEBOX);
            }
        }
    }
}
```

10.49.12 Customizing Individual Cell Appearance

You can control text colour, background colour, and attached image of each cell in the property grid. Use wxPropertyGridInterface::SetPropertyCell() or wxPGProperty::SetCell() for this purpose.

In addition, it is possible to control these characteristics for wxPGChoices list items. See wxPGChoices class reference for more info.

10.49.13 Customizing Keyboard Handling

There is probably one preference for keyboard handling for every developer out there, and as a conveniency control wxPropertyGrid tries to cater for that. By the default arrow keys are used for navigating between properties, and TAB key is used to move focus between the property editor and the first column. When the focus is in the editor, arrow keys usually no longer work for navigation since they are consumed by the editor.

There are mainly two functions which you can use this customize things, wxPropertyGrid::AddActionTrigger() and wxPropertyGrid::DedicateKey(). First one can be used to set a navigation event to occur on a specific key press and the second is used to divert a key from property editors, making it possible for the grid to use keys normally consumed by the focused editors.

For example, let's say you want to have an ENTER-based editing scheme. That is, editor is focused on ENTER press and the next property is selected when the user finishes editing and presses ENTER again. Code like this would accomplish the task:

```
// Have property editor focus on Enter
propgrid->AddActionTrigger( wxPG_ACTION_EDIT, WXK_RETURN );
```

```
// Have Enter work as action trigger even when editor is focused
propgrid->DedicateKey( WXK_RETURN );

// Let Enter also navigate to the next property
propgrid->AddActionTrigger( wxPG_ACTION_NEXT_PROPERTY,
      WXK_RETURN );
```

wxPG_ACTION_EDIT is prioritized above wxPG_ACTION_NEXT_PROPERTY so that the above code can work without conflicts. For a complete list of available actions, see wxPropertyGrid Action Identifiers.

Here's another trick. Normally the up and down cursor keys are consumed by the focused wxTextCtrl editor and as such can't be used for navigating between properties when that editor is focused. However, using DedicateKey() we can change this so that instead of the cursor keys moving the caret inside the wxTextCtrl, they navigate between adjacent properties. As such:

```
propgrid->DedicateKey(WXK_UP);
propgrid->DedicateKey(WXK_DOWN);
```

10.49.14 Customizing Properties (without sub-classing)

In this section are presented miscellaneous ways to have custom appearance and behaviour for your properties without all the necessary hassle of sub-classing a property class etc.

Setting Value Image

Every property can have a small value image placed in front of the actual value text. Built-in example of this can be seen with wxColourProperty and wxImageFileProperty, but for others it can be set using wxPropertyGrid::SetPropertyImage method.

Setting Property's Editor Control(s)

You can set editor control (or controls, in case of a control and button), of any property using wxPropertyGrid::SetPropertyEditor. Editors are passed as wxPGEditor_EditorName, and valid built-in EditorNames are TextCtrl, Choice, ComboBox, CheckBox, TextCtrlAndButton, ChoiceAndButton, SpinCtrl, and DatePickerCtrl. Two last mentioned ones require call to static member function wxPropertyGrid::RegisterAdditionalEditors().

Following example changes wxColourProperty's editor from default Choice to TextCtrlAndButton. wxColourProperty has its internal event handling set up so that button click events of the button will be used to trigger colour selection dialog.

```
wxPGProperty* colProp = new wxColourProperty("Text Colour");
pg->Append(colProp);
pg->SetPropertyEditor(colProp, wxPGEditor_TextCtrlAndButton);
```

Naturally, creating and setting custom editor classes is a possibility as well. For more information, see wxPGEditor class reference.

Property Attributes Recognized by Editors

SpinCtrl editor can make use of property's "Min", "Max", "Step" and "Wrap" attributes.

Adding Multiple Buttons Next to an Editor

See wxPGMultiButton class reference.

Handling Events Passed from Properties

wxEVT_COMMAND_BUTTON_CLICKED (corresponds to event table macro EVT_BUTTON): Occurs when editor button click is not handled by the property itself (as is the case, for example, if you set property's editor to TextCtrl↩ AndButton from the original TextCtrl).

Property Attributes

Miscellaneous values, often specific to a property type, can be set using wxPropertyGridInterface::SetProperty↩ Attribute() and wxPropertyGridInterface::SetPropertyAttributeAll() methods.

Attribute names are strings and values wxVariant. Arbitrary names are allowed in order to store values that are relevant to application only and not property grid. Constant equivalents of all attribute string names are provided. Some of them are defined as cached strings, so using these constants can provide for smaller binary size.

For complete list of attributes, see wxPropertyGrid Property Attribute Identifiers.

10.49.15 Using wxPropertyGridManager

wxPropertyGridManager is an efficient multi-page version of wxPropertyGrid, which can optionally have tool bar for mode and page selection, and a help text box. For more information, see wxPropertyGridManager class reference.

wxPropertyGridPage

wxPropertyGridPage is holder of properties for one page in manager. It is derived from wxEvtHandler, so you can subclass it to process page-specific property grid events. Hand over your page instance in wxPropertyGrid↩ Manager::AddPage().

Please note that the wxPropertyGridPage itself only sports subset of wxPropertyGrid API (but unlike manager, this include item iteration). Naturally it inherits from wxPropertyGridInterface.

For more information, see wxPropertyGridPage class reference.

10.49.16 Sub-classing wxPropertyGrid and wxPropertyGridManager

Few things to note:

- Only a small percentage of member functions are virtual. If you need more, just e-mail to wx-dev mailing list.
- Data manipulation is done in wxPropertyGridPageState class. So, instead of overriding wxPropertyGrid::↩ Insert(), you'll probably want to override wxPropertyGridPageState::DoInsert(). See header file for details.
- Override wxPropertyGrid::CreateState() to instantiate your derivate wxPropertyGridPageState. For wx↩ PropertyGridManager, you'll need to subclass wxPropertyGridPage instead (since it is derived from wx↩ PropertyGridPageState), and hand over instances in wxPropertyGridManager::AddPage() calls.
- You can use a derivate wxPropertyGrid with manager by overriding wxPropertyGridManager::Create↩ PropertyGrid() member function.

10.49.17 Miscellaneous Topics

Property Name Scope

All properties which parent is category or root can be accessed directly by their base name (ie. name given for property in its constructor). Other properties can be accessed via "ParentsName.BaseName" notation, Naturally, all property names should be unique.

Non-unique Labels

It is possible to have properties with identical label under same parent. However, care must be taken to ensure that each property still has unique (base) name.

wxBoolProperty

There are few points about wxBoolProperty that require further discussion:

- wxBoolProperty can be shown as either normal combo box or as a check box. Property attribute wxPG_BOOL_USE_CHECKBOX is used to change this. For example, if you have a wxFlagsProperty, you can set its all items to use check box using the following:

  ```
  pg->SetPropertyAttribute("MyFlagsProperty",
        wxPG_BOOL_USE_CHECKBOX, true, wxPG_RECURSE);
  ```

  Following will set all individual bool properties in your control to use check box:

```
pg->SetPropertyAttributeAll(wxPG_BOOL_USE_CHECKBOX, true);
```

- Default item names for wxBoolProperty are ["False", "True"]. This can be changed using static function wxPropertyGrid::SetBoolChoices(trueChoice, falseChoice).

Updates from wxTextCtrl Based Editor

Changes from wxTextCtrl based property editors are committed (ie. wxEVT_PG_CHANGED is sent etc.) *only* when (1) user presser enter, (2) user moves to edit another property, or (3) when focus leaves the grid.

Because of this, you may find it useful, in some apps, to call wxPropertyGrid::CommitChangesFromEditor() just before you need to do any computations based on property grid values. Note that CommitChangesFromEditor() will dispatch wxEVT_PG_CHANGED with ProcessEvent, so any of your event handlers will be called immediately.

Centering the Splitter

If you need to center the splitter, but only once when the program starts, then do **not** use the wxPG_SPLITTER_AUTO_CENTER window style, but the wxPropertyGrid::CenterSplitter() method. **However, be sure to call it after the sizer setup and SetSize calls!** (ie. usually at the end of the frame/dialog constructor)

Splitter centering behaviour can be customized using wxPropertyGridInterface::SetColumnProportion(). Usually it is used to set non-equal column proportions, which in essence stops the splitter(s) from being 'centered' as such, and instead just auto-resized.

Setting Splitter Position When Creating Property Grid

Splitter position cannot exceed grid size, and therefore setting it during form creation may fail as initial grid size is often smaller than desired splitter position, especially when sizers are being used.

wxColourProperty and wxSystemColourProperty

Through sub-classing, these two property classes provide substantial customization features. Subclass wxSystemColourProperty if you want to use wxColourPropertyValue (which features colour type in addition to wxColour), and wxColourProperty if plain wxColour is enough.

Override wxSystemColourProperty::ColourToString() to redefine how colours are printed as strings.

Override wxSystemColourProperty::GetCustomColourIndex() to redefine location of the item that triggers colour picker dialog (default is last).

Override wxSystemColourProperty::GetColour() to determine which colour matches which choice entry.

10.49.18 Property Class Descriptions

See Supplied Ready-to-use Property Classes

10.49.19 Changes from wxPropertyGrid 1.4

Version of wxPropertyGrid bundled with wxWidgets 2.9+ has various backward- incompatible changes from version 1.4, which had a stable API and will remain as the last separate branch.

Note that in general any behaviour-breaking changes should not compile or run without warnings or errors.

General Changes

- Tab-traversal can no longer be used to travel between properties. Now it only causes focus to move from main grid to editor of selected property. Arrow keys are now your primary means of navigating between properties, with keyboard. This change allowed fixing broken tab traversal on wxGTK (which is open issue in wxPropertyGrid 1.4).
- wxPG_EX_UNFOCUS_ON_ENTER style is removed and is now default behaviour. That is, when enter is pressed, editing is considered done and focus moves back to the property grid from the editor control.
- A few member functions were removed from wxPropertyGridInterface. Please use wxPGProperty's counterparts from now on.
- wxPGChoices now has proper Copy-On-Write behaviour.
- wxPGChoices::SetExclusive() was renamed to AllocExclusive().
- wxPGProperty::SetPropertyChoicesExclusive() was removed. Instead, use GetChoices().AllocExclusive().
- wxPGProperty::ClearModifiedStatus() is removed. Please use SetModifiedStatus() instead.
- wxPropertyGridInterface::GetExpandedProperties() is removed. You should now use wxPropertyGrid↩ Interface::GetEditableState() instead.
- wxPG_EX_DISABLE_TLP_TRACKING is now enabled by default. To get the old behaviour (recommended if you don't use a system that reparents the grid on its own), use the wxPG_EX_ENABLE_TLP_TRACKING extra style.
- Extended window style wxPG_EX_LEGACY_VALIDATORS was removed.
- Default property validation failure behaviour has been changed to (wxPG_VFB_MARK_CELL | wxPG_VF↩ B_SHOW_MESSAGEBOX), which means that the cell is marked red and wxMessageBox is shown. This is more user-friendly than the old behaviour, which simply beeped and prevented leaving the property editor until a valid value was entered.
- wxPropertyGridManager now has same Get/SetSelection() semantics as wxPropertyGrid.
- Various wxPropertyGridManager page-related functions now return pointer to the page object instead of index.
- wxArrayEditorDialog used by wxArrayStringProperty and some sample properties has been renamed to wx↩ PGArrayEditorDialog. Also, it now uses wxEditableListBox for editing.
- Instead of calling wxPropertyGrid::SetButtonShortcut(), use wxPropertyGrid::SetActionTrigger(wxPG_ACT↩ ION_PRESS_BUTTON).
- wxPGProperty::GetCell() now returns a reference. AcquireCell() was removed.
- wxPGMultiButton::FinalizePosition() has been renamed to Finalize(), and it has slightly different argument list.
- wxPropertyGridEvent::HasProperty() is removed. You can use GetProperty() as immediate replacement when checking if event has a property.

- "InlineHelp" property has been replaced with "Hint".
- wxPropertyGrid::CanClose() has been removed. Call wxPropertyGridInterface::EditorValidate() instead.
- wxPGProperty::SetFlag() has been moved to private API. This was done to underline the fact that it was not the preferred method to change a property's state since it never had any desired side-effects. ChangeFlag() still exists for those who really need to achieve the same effect.
- wxArrayStringProperty default delimiter is now comma (','), and it can be changed by setting the new "↩ Delimiter" attribute.

Property and Editor Sub-classing Changes

- Confusing custom property macros have been eliminated.
- Implement wxPGProperty::ValueToString() instead of GetValueAsString().
- wxPGProperty::ChildChanged() must now return the modified value of whole property instead of writing it back into 'thisValue' argument.
- Removed wxPropertyGrid::PrepareValueForDialogEditing(). Use wxPropertyGrid::GetPendingEditedValue() instead.
- wxPGProperty::GetChoiceInfo() is removed, as all properties now carry wxPGChoices instance (protected wxPGProperty::m_choices).
- Connect() should no longer be called in implementations of wxPGEditor::CreateControls(). wxProperty↩ Grid automatically passes all events from editor to wxPGEditor::OnEvent() and wxPGProperty::OnEvent(), as appropriate.
- wxPython: Previously some of the reimplemented member functions needed a 'Py' prefix. This is no longer necessary. For instance, if you previously implemented PyStringToValue() for your custom property, you should now just implement StringToValue().

10.50 Common Dialogs

Common dialog classes and functions encapsulate commonly-needed dialog box requirements.

They are all 'modal', grabbing the flow of control until the user dismisses the dialog, to make them easy to use within an application.

Some dialogs have both platform-dependent and platform-independent implementations, so that if underlying windowing systems do not provide the required functionality, the generic classes and functions can stand in. For example, under MS Windows, wxColourDialog uses the standard colour selector. There is also an equivalent called wxGenericColourDialog for other platforms, and a macro defines wxColourDialog to be the same as wxGeneric↩ ColourDialog on non-MS Windows platforms. However, under MS Windows, the generic dialog can also be used, for testing or other purposes.

See also

Common Dialogs

10.50.1 wxColourDialog Overview

Classes: wxColourDialog, wxColourData

The wxColourDialog presents a colour selector to the user, and returns with colour information.

The MS Windows Colour Selector

Under Windows, the native colour selector common dialog is used. This presents a dialog box with three main regions: at the top left, a palette of 48 commonly-used colours is shown. Under this, there is a palette of 16 'custom colours' which can be set by the application if desired. Additionally, the user may open up the dialog box to show a right-hand panel containing controls to select a precise colour, and add it to the custom colour palette.

The Generic Colour Selector

Under non-MS Windows platforms, the colour selector is a simulation of most of the features of the MS Windows selector. Two palettes of 48 standard and 16 custom colours are presented, with the right-hand area containing three sliders for the user to select a colour from red, green and blue components. This colour may be added to the custom colour palette, and will replace either the currently selected custom colour, or the first one in the palette if none is selected. The RGB colour sliders are not optional in the generic colour selector. The generic colour selector is also available under MS Windows; use the name wxGenericColourDialog.

Example

In the samples/dialogs directory, there is an example of using the wxColourDialog class. Here is an excerpt, which sets various parameters of a wxColourData object, including a grey scale for the custom colours. If the user did not cancel the dialog, the application retrieves the selected colour and uses it to set the background of a window.

```
wxColourData data;
data.SetChooseFull(true);
for (int i = 0; i < 16; i++)
{
    wxColour colour(i*16, i*16, i*16);
    data.SetCustomColour(i, colour);
}

wxColourDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxColourData retData = dialog.GetColourData();
    wxColour col = retData.GetColour();
    wxBrush brush(col, wxSOLID);
    myWindow->SetBackground(brush);
    myWindow->Clear();
    myWindow->Refresh();
}
```

10.50.2 wxFontDialog Overview

Classes: wxFontDialog, wxFontData

The wxFontDialog presents a font selector to the user, and returns with font and colour information.

The MS Windows Font Selector

Under Windows, the native font selector common dialog is used. This presents a dialog box with controls for font name, point size, style, weight, underlining, strikeout and text foreground colour. A sample of the font is shown on a white area of the dialog box. Note that in the translation from full MS Windows fonts to wxWidgets font conventions, strikeout is ignored and a font family (such as Swiss or Modern) is deduced from the actual font name (such as Arial or Courier).

The Generic Font Selector

Under non-MS Windows platforms, the font selector is simpler. Controls for font family, point size, style, weight, underlining and text foreground colour are provided, and a sample is shown upon a white background. The generic font selector is also available under MS Windows; use the name wxGenericFontDialog.

Example

In the samples/dialogs directory, there is an example of using the wxFontDialog class. The application uses the returned font and colour for drawing text on a canvas. Here is an excerpt:

```
wxFontData data;
data.SetInitialFont(canvasFont);
data.SetColour(canvasTextColour);

wxFontDialog dialog(this, &data);
if (dialog.ShowModal() == wxID_OK)
{
    wxFontData retData = dialog.GetFontData();
    canvasFont = retData.GetChosenFont();
    canvasTextColour = retData.GetColour();
    myWindow->Refresh();
}
```

10.50.3 wxPrintDialog Overview

Classes: wxPrintDialog, wxPrintData

This class represents the print and print setup common dialogs. You may obtain a wxPrinterDC device context from a successfully dismissed print dialog.

The samples/printing example shows how to use it: see Printing Framework Overview for an excerpt from this example.

10.50.4 wxFileDialog Overview

Classes: wxFileDialog

Pops up a file selector box. On Windows and GTK 2.4+, this is the common file selector dialog. In X, this is a file selector box with somewhat less functionality. The path and filename are distinct elements of a full file pathname.

If path is "", the current directory will be used. If filename is "", no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of wxFD_OPEN, wxFD_SAVE, wxFD_OVERWRITE_PROMPT, wxFD_HIDE_READONLY, wxFD_FILE_MUST_EXIST, wxFD_MULTIPLE, wxFD_CHANGE_DIR or 0.

Both the X and Windows versions implement a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed. In the X version, supplying no default name will result in the wildcard filter being inserted in the filename text item; the filter is ignored if a default name is supplied.

The wildcard may be a specification for multiple types of file with a description for each, such as:

```
"BMP files (*.bmp)|*.bmp|GIF files (*.gif)|*.gif"
```

10.50.5 wxDirDialog Overview

Classes: wxDirDialog

This dialog shows a directory selector dialog, allowing the user to select a single directory.

10.50.6 wxTextEntryDialog Overview

Classes: wxTextEntryDialog

This is a dialog with a text entry field. The value that the user entered is obtained using wxTextEntryDialog::GetValue().

10.50.7 wxPasswordEntryDialog Overview

Classes: wxPasswordEntryDialog

This is a dialog with a password entry field. The value that the user entered is obtained using wxTextEntryDialog::GetValue().

10.50.8 wxMessageDialog Overview

Classes: wxMessageDialog

This dialog shows a message, plus buttons that can be chosen from OK, Cancel, Yes, and No. Under Windows, an optional icon can be shown, such as an exclamation mark or question mark.

The return value of wxMessageDialog::ShowModal() indicates which button the user pressed.

10.50.9 wxSingleChoiceDialog Overview

Classes: wxSingleChoiceDialog

This dialog shows a list of choices, plus OK and (optionally) Cancel. The user can select one of them. The selection can be obtained from the dialog as an index, a string or client data.

10.50.10 wxMultiChoiceDialog Overview

Classes: wxMultiChoiceDialog

This dialog shows a list of choices, plus OK and (optionally) Cancel. The user can select one or more of them.

10.51 Toolbar Overview

The toolbar family of classes allows an application to use toolbars in a variety of configurations and styles.

The toolbar is a popular user interface component and contains a set of bitmap buttons or toggles. A toolbar gives faster access to an application's facilities than menus, which have to be popped up and selected rather laboriously.

Instead of supplying one toolbar class with a number of different implementations depending on platform, wxWidgets separates out the classes. This is because there are a number of different toolbar styles that you may wish to use simultaneously, and also, future toolbar implementations will emerge which cannot all be shoe-horned into the one class.

For each platform, the symbol wxToolBar is defined to be one of the specific toolbar classes.

The following is a summary of the toolbar classes and their differences:

- wxToolBarBase: This is a base class with pure virtual functions, and should not be used directly.
- wxToolBarSimple: A simple toolbar class written entirely with generic wxWidgets functionality. A simple 3D effect for buttons is possible, but it is not consistent with the Windows look and feel. This toolbar can scroll, and you can have arbitrary numbers of rows and columns.
- wxToolBarMSW: This class implements an old-style Windows toolbar, only on Windows. There are small, three-dimensional buttons, which do not (currently) reflect the current Windows colour settings: the buttons are grey. This is the default wxToolBar on 16-bit windows.
- wxToolBar95: Uses the native Windows 95 toolbar class. It dynamically adjusts it's background and button colours according to user colour settings. CreateTools must be called after the tools have been added. No absolute positioning is supported but you can specify the number of rows, and add tool separators with `AddSeparator`. Tooltips are supported. `OnRightClick` is not supported. This is the default wxToolBar on

Windows 95, Windows NT 4 and above. With the style wxTB_FLAT, the flat toolbar look is used, with a border that is highlighted when the cursor moves over the buttons.

A toolbar might appear as a single row of images under the menubar, or it might be in a separate frame layout in several rows and columns. The class handles the layout of the images, unless explicit positioning is requested.

A tool is a bitmap which can either be a button (there is no 'state', it just generates an event when clicked) or it can be a toggle. If a toggle, a second bitmap can be provided to depict the 'on' state; if the second bitmap is omitted, either the inverse of the first bitmap will be used (for monochrome displays) or a thick border is drawn around the bitmap (for colour displays where inverting will not have the desired result).

The Windows-specific toolbar classes expect 16-colour bitmaps that are 16 pixels wide and 15 pixels high. If you want to use a different size, call `SetToolBitmapSize` as the demo shows, before adding tools to the button bar. Don't supply more than one bitmap for each tool, because the toolbar generates all three images (normal, depressed, and checked) from the single bitmap you give it.

10.51.1 Using the Toolbar Library

Include `"wx/toolbar.h"`, or if using a class directly, one of:

- `"wx/msw/tbarmsw.h"` for wxToolBarMSW
- `"wx/msw/tbar95.h"` for wxToolBar95
- `"wx/tbarsmpl.h"` for wxToolBarSimple

An example of using a toolbar is given in the "toolbar" sample.

10.52 wxGrid Overview

wxGrid and its related classes are used for displaying and editing tabular data.

wxGrid supports custom attributes for the table cells, allowing to completely customize its appearance and uses a separate grid table (wxGridTableBase-derived) class for the data management meaning that it can be used to display arbitrary amounts of data.

10.52.1 Getting Started

For simple applications you need only refer to the wxGrid class in your code. This example shows how you might create a grid in a frame or dialog constructor and illustrates some of the formatting functions.

```
// Create a wxGrid object

grid = new wxGrid( this,
                   -1,
                   wxPoint( 0, 0 ),
                   wxSize( 400, 300 ) );

// Then we call CreateGrid to set the dimensions of the grid
// (100 rows and 10 columns in this example)
grid->CreateGrid( 100, 10 );

// We can set the sizes of individual rows and columns
// in pixels
grid->SetRowSize( 0, 60 );
grid->SetColSize( 0, 120 );

// And set grid cell contents as strings
grid->SetCellValue( 0, 0, "wxGrid is good" );

// We can specify that some cells are read->only
grid->SetCellValue( 0, 3, "This is read->only" );
grid->SetReadOnly( 0, 3 );
```

```
// Colours can be specified for grid cell contents
grid->SetCellValue(3, 3, "green on grey");
grid->SetCellTextColour(3, 3, *wxGREEN);
grid->SetCellBackgroundColour(3, 3, *wxLIGHT_GREY);

// We can specify the some cells will store numeric
// values rather than strings. Here we set grid column 5
// to hold floating point values displayed with width of 6
// and precision of 2
grid->SetColFormatFloat(5, 6, 2);
grid->SetCellValue(0, 6, "3.1415");
```

Here is a list of classes related to wxGrid:

- wxGrid: The main grid control class itself.
- wxGridTableBase: The base class for grid data provider.
- wxGridStringTable: Simple wxGridTableBase implementation supporting only string data items and storing them all in memory (hence suitable for not too large grids only).
- wxGridCellAttr: A cell attribute, allowing to customize its appearance as well as the renderer and editor used for displaying and editing it.
- wxGridCellAttrProvider: The object responsible for storing and retrieving the cell attributes.
- wxGridColLabelWindow: The window showing the grid columns labels.
- wxGridRowLabelWindow: The window showing the grid rows labels.
- wxGridCornerLabelWindow: The window used in the upper left grid corner.
- wxGridWindow: The window representing the main part of the grid.
- wxGridCellRenderer: Base class for objects used to display a cell value.
- wxGridCellStringRenderer: Renderer showing the cell as a text string.
- wxGridCellNumberRenderer: Renderer showing the cell as an integer number.
- wxGridCellFloatRenderer: Renderer showing the cell as a floating point number.
- wxGridCellBoolRenderer: Renderer showing the cell as checked or unchecked box.
- wxGridCellEditor: Base class for objects used to edit the cell value.
- wxGridCellStringEditor: Editor for cells containing text strings.
- wxGridCellNumberEditor: Editor for cells containing integer numbers.
- wxGridCellFloatEditor: Editor for cells containing floating point numbers.
- wxGridCellBoolEditor: Editor for boolean-valued cells.
- wxGridCellChoiceEditor: Editor allowing to choose one of the predefined strings (and possibly enter new one).
- wxGridEvent: The event sent by most of wxGrid actions.
- wxGridSizeEvent: The special event sent when a grid column or row is resized.
- wxGridRangeSelectEvent: The special event sent when a range of cells is selected in the grid.
- wxGridEditorCreatedEvent: The special event sent when a cell editor is created.
- wxGridSelection: The object efficiently representing the grid selection.
- wxGridTypeRegistry: Contains information about the data types supported by the grid.

10.52.2 Column and Row Sizes

NB: This section will discuss the resizing of wxGrid rows only to avoid repetitions but everything in it also applies to grid columns, just replace `Row` in the method names with `Col`.

Initially all wxGrid rows have the same height, which can be modified for all of them at once using wxGrid::SetDefaultRowSize(). However, unlike simpler controls such as wxListBox or wxListCtrl, wxGrid also allows its rows to be individually resized to have their own height using wxGrid::SetRowSize() (as a special case, a row may be hidden entirely by setting its size to 0, which is done by a helper wxGrid::HideRow() method). It is also possible to resize a row to fit its contents with wxGrid::AutoSizeRow() or do it for all rows at once with wxGrid::AutoSizeRows().

Additionally, by default the user can also drag the row separator lines to resize the rows interactively. This can be forbidden completely by calling wxGrid::DisableDragRowSize() or just for the individual rows using wxGrid::DisableRowResize().

If you do allow the user to resize the grid rows, it may be a good idea to save their heights and restore it when the grid is recreated the next time (possibly during a next program execution): the functions wxGrid::GetRowSizes() and wxGrid::SetRowSizes() can help with this, you will just need to serialize wxGridSizesInfo structure returned by the former in some way and deserialize it back before calling the latter.

10.53 wxTreeCtrl Overview

The tree control displays its items in a tree like structure.

Each item has its own (optional) icon and a label. An item may be either collapsed (meaning that its children are not visible) or expanded (meaning that its children are shown). Each item in the tree is identified by its `itemId` which is of opaque data type wxTreeItemId. You can test whether an item is valid by calling wxTreeItemId::IsOk.

See also

wxTreeCtrl, wxImageList

The items text and image may be retrieved and changed with (Get|Set)ItemText and (Get|Set)ItemImage. In fact, an item may even have two images associated with it: the normal one and another one for selected state which is set/retrieved with (Get|Set)ItemSelectedImage functions, but this functionality might be unavailable on some platforms.

Tree items have several attributes: an item may be selected or not, visible or not, bold or not. It may also be expanded or collapsed. All these attributes may be retrieved with the corresponding functions: IsSelected, IsVisible, IsBold and IsExpanded. Only one item at a time may be selected, selecting another one (with SelectItem) automatically unselects the previously selected one.

In addition to its icon and label, a user-specific data structure may be associated with all tree items. If you wish to do it, you should derive a class from wxTreeItemData which is a very simple class having only one function GetId() which returns the id of the item this data is associated with. This data will be freed by the control itself when the associated item is deleted (all items are deleted when the control is destroyed), so you shouldn't delete it yourself (if you do it, you should call SetItemData(NULL) to prevent the tree from deleting the pointer second time). The associated data may be retrieved with GetItemData() function.

Working with trees is relatively straightforward if all the items are added to the tree at the moment of its creation. However, for large trees it may be very inefficient. To improve the performance you may want to delay adding the items to the tree until the branch containing the items is expanded: so, in the beginning, only the root item is created (with AddRoot). Other items are added when EVT_TREE_ITEM_EXPANDING event is received: then all items lying immediately under the item being expanded should be added, but, of course, only when this event is received for the first time for this item - otherwise, the items would be added twice if the user expands/collapses/re-expands the branch.

The tree control provides functions for enumerating its items. There are 3 groups of enumeration functions: for the children of a given item, for the sibling of the given item and for the visible items (those which are currently shown to the user: an item may be invisible either because its branch is collapsed or because it is scrolled out of view). Child enumeration functions require the caller to give them a *cookie* parameter: it is a number which is opaque to

the caller but is used by the tree control itself to allow multiple enumerations to run simultaneously (this is explicitly allowed). The only thing to remember is that the *cookie* passed to GetFirstChild and to GetNextChild should be the same variable (and that nothing should be done with it by the user code).

Among other features of the tree control are: item sorting with SortChildren which uses the user-defined comparison function OnCompareItems (by default the comparison is the alphabetic comparison of tree labels), hit testing (determining to which portion of the control the given point belongs, useful for implementing drag-and-drop in the tree) with HitTest and editing of the tree item labels in place (see EditLabel).

Finally, the tree control has a keyboard interface: the cursor navigation (arrow) keys may be used to change the current selection. HOME and END are used to go to the first/last sibling of the current item. '+', '-' and '∗' expand, collapse and toggle the current branch. Note, however, that DEL and INS keys do nothing by default, but it is common to associate them with deleting an item from a tree and inserting a new one into it.

10.54 wxListCtrl Overview

Todo The wxListCtrl topic overview still needs to be written, sorry.

See also

wxListCtrl, wxImageList, wxListItem

10.55 wxSplitterWindow Overview

See also

wxSplitterWindow

10.55.1 Appearance

The following screenshot shows the appearance of a splitter window with a horizontal split.

The style wxSP_3D has been used to show a 3D border and 3D sash.

10.55.2 Example

The following fragment shows how to create a splitter window, creating two subwindows and hiding one of them.

```
splitter = new wxSplitterWindow(this, -1, wxPoint(0, 0),
                                wxSize(400, 400), wxSP_3D);

leftWindow = new MyWindow(splitter);
leftWindow->SetScrollbars(20, 20, 50, 50);

rightWindow = new MyWindow(splitter);
rightWindow->SetScrollbars(20, 20, 50, 50);
rightWindow->Show(false);

splitter->Initialize(leftWindow);

// Set this to prevent unsplitting
// splitter->SetMinimumPaneSize(20);
```

The next fragment shows how the splitter window can be manipulated after creation.

```
void MyFrame::OnSplitVertical(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )
        splitter->Unsplit();
    leftWindow->Show(true);
    rightWindow->Show(true);
```

```
    splitter->SplitVertically( leftWindow, rightWindow );
}

void MyFrame::OnSplitHorizontal(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )
        splitter->Unsplit();
    leftWindow->Show(true);
    rightWindow->Show(true);
    splitter->SplitHorizontally( leftWindow, rightWindow );
}

void MyFrame::OnUnsplit(wxCommandEvent& event)
{
    if ( splitter->IsSplit() )
        splitter->Unsplit();
}
```

10.56 wxBookCtrl Overview

A book control is a convenient way of displaying multiple pages of information, displayed one page at a time.

wxWidgets has five variants of this control:

- wxChoicebook: controlled by a wxChoice
- wxListbook: controlled by a wxListCtrl
- wxNotebook: uses a row of tabs
- wxSimplebook: doesn't allow the user to change the page at all.
- wxTreebook: controlled by a wxTreeCtrl
- wxToolbook: controlled by a wxToolBar

See the Notebook Sample for an example of wxBookCtrl usage.

Notice that wxSimplebook is special in that it only allows the program to change the selection, thus it's usually used in slightly different circumstances than the other variants.

See also

Book Controls

10.56.1 Best Book

wxBookCtrl is mapped to the class best suited for a given platform. Currently it provides wxChoicebook for smartphones equipped with WinCE, and wxNotebook for all other platforms. The mapping consists of:

wxBookCtrl	wxChoicebook or wxNotebook
wxEVT_COMMAND_BOOKCTRL_PAGE_CHANGED	wxEVT_COMMAND_CHOICEBOOK_PAGE_CHANGED or wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGED
wxEVT_COMMAND_BOOKCTRL_PAGE_CHANGING	wxEVT_COMMAND_CHOICEBOOK_PAGE_CHANGING or wxEVT_COMMAND_NOTEBOOK_PAGE_CHANGING

EVT_BOOKCTRL_PAGE_CHANGED(id, fn)	EVT_CHOICEBOOK_PAGE_CHANGED(id, fn) or EVT_NOTEBOOK_PAGE_CHANGED(id, fn)
EVT_BOOKCTRL_PAGE_CHANGING(id, fn)	EVT_CHOICEBOOK_PAGE_CHANGING(id, fn) or EVT_NOTEBOOK_PAGE_CHANGING(id, fn)

For orientation of the book controller, use following flags in style:

- **wxBK_TOP:** controller above pages
- **wxBK_BOTTOM:** controller below pages
- **wxBK_LEFT:** controller on the left
- **wxBK_RIGHT:** controller on the right
- **wxBK_DEFAULT:** native controller placement

10.57 wxTipProvider Overview

Many "modern" Windows programs have a feature (some would say annoyance) of presenting the user tips at program startup.

While this is probably useless to the advanced users of the program, the experience shows that the tips may be quite helpful for the novices and so more and more programs now do this. For a wxWidgets programmer, implementing this feature is extremely easy. To show a tip, it is enough to just call wxShowTip function like this:

```
if ( ...show tips at startup?... )
{
    wxTipProvider *tipProvider = wxCreateFileTipProvider("tips.txt", 0)
      ;
    wxShowTip(windowParent, tipProvider);
    delete tipProvider;
}
```

Of course, you need to get the text of the tips from somewhere - in the example above, the text is supposed to be in the file tips.txt from where it is read by the *tip provider*. The tip provider is just an object of a class deriving from wxTipProvider. It has to implement one pure virtual function of the base class: GetTip. In the case of the tip provider created by wxCreateFileTipProvider, the tips are just the lines of the text file.

If you want to implement your own tip provider (for example, if you wish to hardcode the tips inside your program), you just have to derive another class from wxTipProvider and pass a pointer to the object of this class to wxShowTip - then you don't need wxCreateFileTipProvider at all.

You will probably want to save somewhere the index of the tip last shown - so that the program doesn't always show the same tip on startup. As you also need to remember whether to show tips or not (you shouldn't do it if the user unchecked "Show tips on startup" checkbox in the dialog), you will probably want to store both the index of the last shown tip (as returned by wxTipProvider::GetCurrentTip and the flag telling whether to show the tips at startup at all.

In a tips.txt file, lines that begin with a # character are considered comments and are automatically skipped. Blank lines and lines only having spaces are also skipped.

You can easily add runtime-translation capacity by placing each line of the tips.txt file inside the usual translation macro. For example, your tips.txt file would look like this:

```
_("This is my first tip")
_("This is my second tip")
```

Now add your tips.txt file into the list of files that gettext searches for translatable strings. The tips will thus get included into your generated .po file catalog and be translated at runtime along with the rest of your application's translatable strings.

Note

Each line in the tips.txt file needs to strictly begin with exactly the 3 characters of underscore-parenthesis-doublequote, and end with doublequote-parenthesis, as shown above. Also, remember to escape any double-quote characters within the tip string with a backslash-doublequote.

See the dialogs program in your samples folder for a working example inside a program.

10.58 Document/View Framework

The document/view framework is found in most application frameworks, because it can dramatically simplify the code required to build many kinds of application.

The idea is that you can model your application primarily in terms of *documents* to store data and provide interface-independent operations upon it, and *views* to visualise and manipulate the data. Documents know how to do input and output given stream objects, and views are responsible for taking input from physical windows and performing the manipulation on the document data.

If a document's data changes, all views should be updated to reflect the change. The framework can provide many user-interface elements based on this model.

Once you have defined your own classes and the relationships between them, the framework takes care of popping up file selectors, opening and closing files, asking the user to save modifications, routing menu commands to appropriate (possibly default) code, even some default print/preview functionality and support for command undo/redo.

The framework is highly modular, allowing overriding and replacement of functionality and objects to achieve more than the default behaviour.

These are the overall steps involved in creating an application based on the document/view framework:

- Define your own document and view classes, overriding a minimal set of member functions e.g. for input/output, drawing and initialization.
- Define any subwindows (such as a scrolled window) that are needed for the view(s). You may need to route some events to views or documents, for example, "OnPaint" needs to be routed to wxView::OnDraw.
- Decide what style of interface you will use: Microsoft's MDI (multiple document child frames surrounded by an overall frame), SDI (a separate, unconstrained frame for each document), or single-window (one document open at a time, as in Windows Write).
- Use the appropriate wxDocParentFrame and wxDocChildFrame classes. Construct an instance of wxDocParentFrame in your wxApp::OnInit, and a wxDocChildFrame (if not single-window) when you initialize a view. Create menus using standard menu ids (such as wxID_OPEN, wxID_PRINT).
- Construct a single wxDocManager instance at the beginning of your wxApp::OnInit, and then as many wxDocTemplate instances as necessary to define relationships between documents and views. For a simple application, there will be just one wxDocTemplate.

If you wish to implement Undo/Redo, you need to derive your own class(es) from wxCommand and use wxCommandProcessor::Submit instead of directly executing code. The framework will take care of calling Undo and Do functions as appropriate, so long as the wxID_UNDO and wxID_REDO menu items are defined in the view menu.

Here are a few examples of the tailoring you can do to go beyond the default framework behaviour:

- Override wxDocument::OnCreateCommandProcessor to define a different Do/Undo strategy, or a command history editor.
- Override wxView::OnCreatePrintout to create an instance of a derived wxPrintout class, to provide multi-page document facilities.
- Override wxDocManager::SelectDocumentPath to provide a different file selector.

- Limit the maximum number of open documents and the maximum number of undo commands.

Note that to activate framework functionality, you need to use some or all of the wxWidgets Predefined Command Identifiers in your menus.

wxPerl Note: The document/view framework is available in wxPerl. To use it, you will need the following statements in your application code:

```
1 use Wx::DocView;
2 use Wx ':docview';   # import constants (optional)
```

See also

Document/View Framework,

10.58.1 wxDocument Overview

The wxDocument class can be used to model an application's file-based data. It is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocTemplate and wxDocManager classes. Using this framework can save a lot of routine user-interface programming, since a range of menu commands – such as open, save, save as – are supported automatically.

The programmer just needs to define a minimal set of classes and member functions for the framework to call when necessary. Data, and the means to view and edit the data, are explicitly separated out in this model, and the concept of multiple *views* onto the same data is supported.

Note that the document/view model will suit many but not all styles of application. For example, it would be overkill for a simple file conversion utility, where there may be no call for *views* on *documents* or the ability to open, edit and save files. But probably the majority of applications are document-based.

See the example application in `samples/docview`. To use the abstract wxDocument class, you need to derive a new class and override at least the member functions SaveObject and LoadObject. SaveObject and LoadObject will be called by the framework when the document needs to be saved or loaded.

Use the macros DECLARE_DYNAMIC_CLASS and IMPLEMENT_DYNAMIC_CLASS in order to allow the framework to create document objects on demand. When you create a wxDocTemplate object on application initialization, you should pass CLASSINFO(YourDocumentClass) to the wxDocTemplate constructor so that it knows how to create an instance of this class.

If you do not wish to use the wxWidgets method of creating document objects dynamically, you must override wxDocTemplate::CreateDocument to return an instance of the appropriate class.

10.58.2 wxView Overview

The wxView class can be used to model the viewing and editing component of an application's file-based data. It is part of the document/view framework supported by wxWidgets, and cooperates with the wxDocument, wxDoc↩ Template and wxDocManager classes.

See the example application in `samples/docview`.

To use the abstract wxView class, you need to derive a new class and override at least the member functions On↩ Create, OnDraw, OnUpdate and OnClose. You will probably want to respond to menu commands from the frame containing the view.

Use the macros DECLARE_DYNAMIC_CLASS and IMPLEMENT_DYNAMIC_CLASS in order to allow the framework to create view objects on demand. When you create a wxDocTemplate object on application initialization, you should pass CLASSINFO(YourViewClass) to the wxDocTemplate constructor so that it knows how to create an instance of this class.

If you do not wish to use the wxWidgets method of creating view objects dynamically, you must override wxDoc↩ Template::CreateView to return an instance of the appropriate class.

10.58.3 wxDocTemplate Overview

The wxDocTemplate class is used to model the relationship between a document class and a view class. The application creates a document template object for each document/view pair. The list of document templates managed by the wxDocManager instance is used to create documents and views. Each document template knows what file filters and default extension are appropriate for a document/view combination, and how to create a document or view.

For example, you might write a small doodling application that can load and save lists of line segments. If you had two views of the data – graphical, and a list of the segments – then you would create one document class DoodleDocument, and two view classes (DoodleGraphicView and DoodleListView). You would also need two document templates, one for the graphical view and another for the list view. You would pass the same document class and default file extension to both document templates, but each would be passed a different view class. When the user clicks on the Open menu item, the file selector is displayed with a list of possible file filters – one for each wxDocTemplate. Selecting the filter selects the wxDocTemplate, and when a file is selected, that template will be used for creating a document and view.

For the case where an application has one document type and one view type, a single document template is constructed, and dialogs will be appropriately simplified.

wxDocTemplate is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocument and wxDocManager classes.

See the example application in `samples/docview`.

To use the wxDocTemplate class, you do not need to derive a new class. Just pass relevant information to the constructor including CLASSINFO(YourDocumentClass) and CLASSINFO(YourViewClass) to allow dynamic instance creation.

If you do not wish to use the wxWidgets method of creating document objects dynamically, you must override wxDocTemplate::CreateDocument and wxDocTemplate::CreateView to return instances of the appropriate class.

Note

The document template has nothing to do with the C++ template construct.

10.58.4 wxDocManager Overview

The wxDocManager class is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocument and wxDocTemplate classes.

A wxDocManager instance coordinates documents, views and document templates. It keeps a list of document and template instances, and much functionality is routed through this object, such as providing selection and file dialogs. The application can use this class 'as is' or derive a class and override some members to extend or change the functionality.

Create an instance of this class near the beginning of your application initialization, before any documents, views or templates are manipulated.

There may be multiple wxDocManager instances in an application. See the example application in `samples/docview`.

10.58.5 Event Propagation in Document/View framework

While wxDocument, wxDocManager and wxView are abstract objects, with which the user can't interact directly, all of them derive from wxEvtHandler class and can handle events arising in the windows showing the document with which the user does interact. This is implemented by adding additional steps to the event handling process described in How Events are Processed, so the full list of the handlers searched for an event occurring directly in wxDocChildFrame is:

1. wxDocument opened in this frame.

2. wxView shown in this frame.
3. wxDocManager associated with the parent wxDocParentFrame.
4. wxDocChildFrame itself.
5. wxDocParentFrame, as per the usual event bubbling up to parent rules.
6. wxApp, again as the usual fallback for all events.

This is mostly useful to define handlers for some menu commands directly in wxDocument or wxView and is also used by the framework itself to define the handlers for several standard commands, such as wxID_NEW or wx↩ ID_SAVE, in wxDocManager itself. Notice that due to the order of the event handler search detailed above, the handling of these commands can *not* be overridden at wxDocParentFrame level but must be done at the level of wxDocManager itself.

10.58.6 wxCommand Overview

wxCommand is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other means provided by the application to change the data or view.

Instead of the application functionality being scattered around switch statements and functions in a way that may be hard to read and maintain, the functionality for a command is explicitly represented as an object which can be manipulated by a framework or application.

When a user interface event occurs, the application *submits* a command to a wxCommandProcessor object to execute and store.

The wxWidgets document/view framework handles Undo and Redo by use of wxCommand and wxCommand↩ Processor objects. You might find further uses for wxCommand, such as implementing a macro facility that stores, loads and replays commands.

An application can derive a new class for every command, or, more likely, use one class parameterized with an integer or string command identifier.

10.58.7 wxCommandProcessor Overview

wxCommandProcessor is a class that maintains a history of wxCommand instances, with undo/redo functionality built-in. Derive a new class from this if you want different behaviour.

10.58.8 wxFileHistory Overview

wxFileHistory encapsulates functionality to record the last few files visited, and to allow the user to quickly load these files using the list appended to the File menu. Although wxFileHistory is used by wxDocManager, it can be used independently. You may wish to derive from it to allow different behaviour, such as popping up a scrolling list of files.

By calling wxFileHistory::UseMenu() you can associate a file menu with the file history. The menu will then be used for appending filenames that are added to the history.

Please notice that currently if the history already contained filenames when UseMenu() is called (e.g. when initializing a second MDI child frame), the menu is not automatically initialized with the existing filenames in the history and so you need to call wxFileHistory::AddFilesToMenu() after UseMenu() explicitly in order to initialize the menu with the existing list of MRU files (otherwise an assertion failure is raised in debug builds).

The filenames are appended using menu identifiers in the range `wxID_FILE1` to `wxID_FILE9`.

In order to respond to a file load command from one of these identifiers, you need to handle them using an event handler, for example:

```
BEGIN_EVENT_TABLE(wxDocParentFrame, wxFrame)
    EVT_MENU(wxID_EXIT, wxDocParentFrame::OnExit)
    EVT_MENU_RANGE(wxID_FILE1, wxID_FILE9, wxDocParentFrame::OnMRUFile)
END_EVENT_TABLE()

void wxDocParentFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    Close();
}

void wxDocParentFrame::OnMRUFile(wxCommandEvent& event)
{
    wxString f(m_docManager->GetHistoryFile(event.GetId() -
      wxID_FILE1));
    if (!f.empty())
        (void)m_docManager-CreateDocument(f, wxDOC_SILENT);
}
```

10.58.9 Predefined Command Identifiers

To allow communication between the application's menus and the document/view framework, several command identifiers are predefined for you to use in menus.

```
wxID_OPEN (5000)
wxID_CLOSE (5001)
wxID_NEW (5002)
wxID_SAVE (5003)
wxID_SAVEAS (5004)
wxID_REVERT (5005)
wxID_EXIT (5006)
wxID_UNDO (5007)
wxID_REDO (5008)
wxID_HELP (5009)
wxID_PRINT (5010)
wxID_PRINT_SETUP (5011)
wxID_PREVIEW (5012)
```

10.59 Backwards Compatibility

Many of the GUIs and platforms supported by wxWidgets are continuously evolving, and some of the new platforms wxWidgets now supports were quite unimaginable even a few years ago.

In this environment wxWidgets must also evolve in order to support these new features and platforms.

However the goal of wxWidgets is not only to provide a consistent programming interface across many platforms, but also to provide an interface that is reasonably stable over time, to help protect its users from some of the uncertainty of the future.

10.59.1 The Version Numbering Scheme

wxWidgets version numbers can have up to four components, with trailing zeros sometimes omitted:

```
major.minor.release.sub-release
```

A stable release of wxWidgets will have an even number for *minor*, e.g. 2.6.0. Stable, in this context, means that the API is not changing. In truth, some changes are permitted, but only those that are backward compatible. For example, you can expect later 2.6.x releases, such as 2.6.1 and 2.6.2 to be backward compatible with their predecessor.

When it becomes necessary to make changes which are not wholly backward compatible, the stable branch is forked, creating a new development branch of wxWidgets. This development branch will have an odd number for *minor*, for example 2.7.x. Releases from this branch are known as development snapshots.

The stable branch and the development branch will then be developed in parallel for some time. When it is no longer useful to continue developing the stable branch, the development branch is renamed and becomes a new stable

branch, for example: 2.8.0. And the process begins again. This is how the tension between keeping the interface stable, and allowing the library to evolve is managed.

You can expect the versions with the same major and even minor version number to be compatible, but between minor versions there will be incompatibilities. Compatibility is not broken gratuitously however, so many applications will require no changes or only small changes to work with the new version.

10.59.2 Source Level Compatibility

Later releases from a stable branch are backward compatible with earlier releases from the same branch at the source level. This means that, for example, if you develop your application using wxWidgets 2.8.0 then it should also compile fine with all later 2.8.x versions.

The converse is also true providing you avoid any new features not present in the earlier version. For example if you develop using 2.6.1 your program will compile fine with wxWidgets 2.8.0 providing you don't use any 2.8.1 specific features.

For some platforms binary compatibility is also supported, see Library Binary Compatibility below.

Between minor versions, for example between 2.4.x, 2.6.x and 2.8.x, there will be some incompatibilities. Wherever possible the old way of doing something is kept alongside the new for a time wrapped inside:

```
#if WXWIN_COMPATIBILITY_2_6
    // deprecated feature
    ...
#endif
```

By default the `WXWIN_COMPATIBILITY_X_X` macro is set to 1 for the previous stable branch, for example in 2.8.x, `WXWIN_COMPATIBILITY_2_6` = 1. For the next earlier stable branch the default is 0, so `WXWIN_COMPATIBILITY_2_4` = 0 for 2.8.x. Earlier than that, obsolete features are removed.

These macros can be changed in `setup.h`. Or on UNIX-like systems you can set them using the `-disable-compat26` and `-enable-compat24` options to configure.

They can be useful in two ways:

- Changing `WXWIN_COMPATIBILITY_2_6` to 0 can be useful to find uses of deprecated features in your program that should eventually be removed.
- Changing `WXWIN_COMPATIBILITY_2_4` to 1 can be useful to compile a program developed using 2.4.x that no longer compiles with 2.8.x.

A program requiring one of these macros to be 1 will become incompatible with some future version of wxWidgets, and you should consider updating it.

10.59.3 Library Binary Compatibility

For some platforms, releases from a stable branch are not only source level compatible but can also be binary compatible.

Binary compatibility makes it possible to get the maximum benefit from using shared libraries, also known as dynamic link libraries (DLLs) on Windows or dynamic shared libraries on OS X.

For example, suppose several applications are installed on a system requiring wxWidgets 2.6.0, 2.6.1 and 2.6.2. Since 2.6.2 is backward compatible with the earlier versions, it should be enough to install just wxWidgets 2.6.2 shared libraries, and all the applications should be able to use them. If binary compatibility is not supported, then all the required versions 2.6.0, 2.6.1 and 2.6.2 must be installed side by side.

Achieving this, without the user being required to have the source code and recompile everything, places many extra constraints on the changes that can be made within the stable branch. So it is not supported for all platforms, and not for all versions of wxWidgets. To date it has mainly been supported by wxGTK for UNIX-like platforms.

Another practical consideration is that for binary compatibility to work, all the applications and libraries must have been compiled with compilers that are capable of producing compatible code; that is, they must use the same ABI (Application Binary Interface). Unfortunately most different C++ compilers do not produce code compatible with each other, and often even different versions of the same compiler are not compatible.

10.59.4 Application Binary Compatibility

The most important aspect of binary compatibility is that applications compiled with one version of wxWidgets, e.g. 2.6.1, continue to work with shared libraries of a later binary compatible version, for example 2.6.2. The converse can also be useful however. That is, it can be useful for a developer using a later version, e.g. 2.6.2 to be able to create binary application packages that will work with all binary compatible versions of the shared library starting with, for example 2.6.0.

To do this the developer must, of course, avoid any features not available in the earlier versions. However this is not necessarily enough; in some cases an application compiled with a later version may depend on it even though the same code would compile fine against an earlier version.

To help with this, a preprocessor symbol `wxABI_VERSION` can be defined during the compilation of the application (this would usually be done in the application's makefile or project settings). It should be set to the lowest version that is being targeted, as a number with two decimal digits for each component, for example `wxABI_VERSIO↩N=20600` for 2.6.0.

Setting `wxABI_VERSION` should prevent the application from implicitly depending on a later version of wxWidgets, and also disables any new features in the API, giving a compile time check that the source is compatible with the versions of wxWidgets being targeted.

Uses of `wxABI_VERSION` are stripped out of the wxWidgets sources when each new development branch is created. Therefore it is only useful to help achieve compatibility with earlier versions with the same major and even minor version numbers. It won't, for example, help you write code compatible with 2.6.x using wxWidgets 2.8.x.

10.60 C++ Exceptions

wxWidgets had been started long before the exceptions were introduced in C++ so it is not very surprising that it is not built around using them as some more modern C++ libraries are.

For instance, the library doesn't throw exceptions to signal about the errors. Moreover, up to (and including) the version 2.4 of wxWidgets, even using the exceptions in the user code was dangerous because the library code wasn't exception-safe and so an exception propagating through it could result in memory and/or resource leaks, and also not very convenient.

However the recent wxWidgets versions are exception-friendly. This means that while the library still doesn't use the exceptions by itself, it should be now safe to use the exceptions in the user code and the library tries to help you with this.

10.60.1 Strategies for Exception Handling

There are several choice for using the exceptions in wxWidgets programs. First of all, you may not use them at all. As stated above, the library doesn't throw any exceptions by itself and so you don't have to worry about exceptions at all unless your own code throws them. This is, of course, the simplest solution but may be not the best one to deal with all possible errors.

The next simplest strategy is to only use exceptions inside non-GUI code, i.e. never let unhandled exceptions escape the event handler in which it happened. In this case using exceptions in wxWidgets programs is not different from using them in any other C++ program.

Things get more interesting if you decide to let (at least some) exceptions escape from the event handler in which they occurred. Such exceptions will be caught by wxWidgets and the special wxApp::OnExceptionInMainLoop() method will be called from the `catch` clause. This allows you to decide in a single place what to do about such

exceptions: you may want to handle the exception somehow or terminate the program. In this sense, OnExceptionInMainLoop() is equivalent to putting a `try/catch` block around the entire `main()` function body in the traditional console programs. However notice that, as its name indicates, this method won't help you with the exceptions thrown before the main loop is started or after it is over, so you may still want to have `try/catch` in your overridden wxApp::OnInit() and wxApp::OnExit() methods too, otherwise wxApp::OnUnhandledException() will be called.

Finally, notice that even if you decide to not let any exceptions escape in this way, this still may happen unexpectedly in a program using exceptions as a result of a bug. So consider always overriding OnExceptionInMainLoop() in your wxApp-derived class if you use exceptions in your program, whether you expect it to be called or not. In the latter case you may simple re-throw the exception and let it bubble up to OnUnhandledException() as well.

To summarize, when you use exceptions in your code, you may handle them in the following places, in order of priority:

1. In a `try/catch` block inside an event handler.
2. In wxApp::OnExceptionInMainLoop().
3. In wxApp::OnUnhandledException().

In the first two cases you may decide whether you want to handle the exception and continue execution or to exit the program. In the last one the program is about to exit already so you can just try to save any unsaved data and notify the user about the problem (while being careful not to throw any more exceptions as otherwise `std::terminate()` will be called).

10.60.2 Handling Exception Inside wxYield()

In some, relatively rare cases, using wxApp::OnExceptionInMainLoop() may not be sufficiently flexible. The most common example is using automated GUI tests, when test failures are signaled by throwing an exception and these exceptions can't be caught in a single central method because their handling depends on the test logic, e.g. sometimes an exception is expected while at other times it is an actual error. Typically this results in writing code like the following:

```
void TestNewDocument()
{
    wxUIActionSimulator ui;
    ui.Char('n', wxMOD_CONTROL); // simulate creating a new file

    // Let wxWidgets dispatch Ctrl+N event, invoke the handler and create the
    // new document.
    try {
        wxYield();
    } catch ( ... ) {
        // Handle exceptions as failure in the new document creation test.
    }
}
```

Unfortunately, by default this example only works when using a C++11 compiler because the exception can't be safely propagated back to the code handling it in `TestNewDocument()` through the system event dispatch functions which are not compatible with C++ exceptions and needs to be stored by wxWidgets when it is first caught and rethrown later, when it is safe to do it. And such storing and rethrowing of exceptions is only possible in C++11, so while everything just works if you do use C++11, there is an extra step if you are using C++98: In this case you need to override wxApp::StoreCurrentException() and wxApp::RethrowStoredException() to help wxWidgets to do this, please see the documentation of these functions for more details.

10.60.3 Technicalities

To use any kind of exception support in the library you need to build it with `wxUSE_EXCEPTIONS` set to 1. It is turned on by default but you may wish to check `include/wx/msw/setup.h` file under Windows or run `configure` with explicit `-enable-exceptions` argument under Unix.

On the other hand, if you do not plan to use exceptions, setting this flag to 0 or using `-disable-exceptions` could result in a leaner and slightly faster library.

As for any other library feature, there is a sample (`except`) showing how to use it. Please look at its sources for further information.

10.61 Runtime Type Information (RTTI)

One of the failings of C++ used to be that no runtime information was provided about a class and its position in the inheritance hierarchy.

Another, which still persists, is that instances of a class cannot be created just by knowing the name of a class, which makes facilities such as persistent storage hard to implement.

Most C++ GUI frameworks overcome these limitations by means of a set of macros and functions and wxWidgets is no exception. As it originated before the addition of RTTI to the C++ standard and as support for it is still missing from some (albeit old) compilers, wxWidgets doesn't (yet) use it, but provides its own macro-based RTTI system.

In the future, the standard C++ RTTI will be used though and you're encouraged to use whenever possible the wxDynamicCast macro which, for the implementations that support it, is defined just as dynamic_cast and uses wxWidgets RTTI for all the others. This macro is limited to wxWidgets classes only and only works with pointers (unlike the real dynamic_cast which also accepts references).

Each class that you wish to be known to the type system should have a macro such as DECLARE_DYNAMIC_CL↩ ASS just inside the class declaration. The macro IMPLEMENT_DYNAMIC_CLASS should be in the implementation file. Note that these are entirely optional; use them if you wish to check object types, or create instances of classes using the class name. However, it is good to get into the habit of adding these macros for all classes.

Variations on these macros are used for multiple inheritance, and abstract classes that cannot be instantiated dynamically or otherwise.

DECLARE_DYNAMIC_CLASS inserts a static wxClassInfo declaration into the class, initialized by IMPLEMEN↩ T_DYNAMIC_CLASS. When initialized, the wxClassInfo object inserts itself into a linked list (accessed through wxClassInfo::first and wxClassInfo::next pointers). The linked list is fully created by the time all global initialisation is done.

IMPLEMENT_DYNAMIC_CLASS is a macro that not only initialises the static wxClassInfo member, but defines a global function capable of creating a dynamic object of the class in question. A pointer to this function is stored in wxClassInfo, and is used when an object should be created dynamically.

wxObject::IsKindOf uses the linked list of wxClassInfo. It takes a wxClassInfo argument, so use CLASSINFO(class↩ Name) to return an appropriate wxClassInfo pointer to use in this function.

The function wxCreateDynamicObject can be used to construct a new object of a given type, by supplying a string name. If you have a pointer to the wxClassInfo object instead, then you can simply call wxClassInfo::CreateObject.

See also

wxObject

10.61.1 wxClassInfo

This class stores meta-information about classes. An application may use macros such as DECLARE_DYNAMI↩ C_CLASS and IMPLEMENT_DYNAMIC_CLASS to record runtime information about a class, including:

- Its position in the inheritance hierarchy.
- The base class name(s) (up to two base classes are permitted).
- A string representation of the class name.
- A function that can be called to construct an instance of this class.

The DECLARE_... macros declare a static wxClassInfo variable in a class, which is initialized by macros of the form IMPLEMENT_... in the implementation C++ file. Classes whose instances may be constructed dynamically are given a global constructor function which returns a new object.

You can get the wxClassInfo for a class by using the CLASSINFO macro, e.g. CLASSINFO(wxFrame). You can get the wxClassInfo for an object using wxObject::GetClassInfo.

10.61.2 Example

In a header file frame.h:

```
class wxFrame : public wxWindow
{
    DECLARE_DYNAMIC_CLASS(wxFrame)

private:
    wxString m_title;

public:
...
};
```

In a C++ file frame.cpp:

```
IMPLEMENT_DYNAMIC_CLASS(wxFrame, wxWindow)

wxFrame::wxFrame()
{
...
}
```

10.62 Caveats When Not Using C++ RTTI

Note

C++ RTTI is usually enabled by default in most wxWidgets builds. If you do not know if your build has C++ RTTI enabled or not, then it probably is enabled, and you should not worry about anything mentioned in this section.

While in general wxWidgets standard Runtime Type Information (RTTI) is used throughout the library, there are some places where it won't work. One of those places is template classes.

When available, C++ RTTI is used to address this issue. If you have built the library with C++ RTTI disabled, an internal RTTI system is substituted. However, this system is not perfect and one proven scenario where it may break is a shared library or DLL build. More specifically, a template class instance created in one physical binary may not be recognized as its correct type when used in another one.

See also

Runtime Type Information (RTTI), wxEvtHandler::Bind(), wxAny

10.62.1 Bind() Issues

wxWidgets 2.9.0 introduced a new Dynamic Event Handling system, using wxEvtHandler::Bind<>() and Unbind<>(). This functionality uses templates behind the scenes and therefore is vulnerable to breakage in shared library builds, as described above.

Currently only Unbind<>() needs the type information, so you should be immune to this problem simply if you only need to use Bind<>() and not Unbind<>().

Also, if you only bind and unbind same event handler inside same binary, you should be fine.

10.62.2 wxAny Issues

wxAny is a dynamic type class which transparently uses templates to generate data type handlers, and therefore is vulnerable to breakage in shared library builds, as described above

You should be fine if you only create and use wxAny instances inside same physical binary. However, if you do need to be able to use wxAny freely across binary boundaries, (and for sake of code-safety, you probably do), then specializations for wxAnyValueTypeImpl<> templates need to be defined in one of your shared library (DLL) files. One specialization is required for every data type you use with wxAny. Easiest way to do this is using macros provided in wx/any.h. Note that you **do not** need to define specializations for C built-in types, nor for wxString or wxDateTime, because these are already provided in wxBase. However, you **do** need to define specializations for all pointer types except char* and wchar_t*.

Let's define a specialization for imaginary type 'MyClass'. In your shared library source code you will need to have this line:

```
WX_IMPLEMENT_ANY_VALUE_TYPE(wxAnyValueTypeImpl<MyClass>)
```

In your header file you will need the following:

```
wxDECLARE_ANY_TYPE(MyClass, WXIMPORT_OR_WXEXPORT)
```

Where WXIMPORT_OR_WXEXPORT is WXEXPORT when being included from the shared library that called the WX_IMPLEMENT_ANY_VALUE_TYPE() macro, and WXIMPORT otherwise.

10.63 Reference Counting

Many wxWidgets objects use a technique known as *reference counting*, also known as *copy on write* (COW).

This means that when an object is assigned to another, no copying really takes place. Only the reference count on the shared object data is incremented and both objects share the same data (a very fast operation).

But as soon as one of the two (or more) objects is modified, the data has to be copied because the changes to one of the objects shouldn't be seen in the others. As data copying only happens when the object is written to, this is known as COW.

What is important to understand is that all this happens absolutely transparently to the class users and that whether an object is shared or not is not seen from the outside of the class - in any case, the result of any operation on it is the same.

10.63.1 Object Comparison

The == and != operators of the reference counted classes always do a *deep comparison*. This means that the equality operator will return true if two objects are identical and not only if they share the same data.

Note that wxWidgets follows the *STL philosophy*: when a comparison operator cannot be implemented efficiently (like for e.g. wxImage's == operator which would need to compare the entire image's data, pixel-by-pixel), it's not implemented at all. That's why not all reference counted classes provide comparison operators.

Also note that if you only need to do a `shallow` comparison between two wxObject derived classes, you should not use the == and != operators but rather the wxObject::IsSameAs() function.

10.63.2 Object Destruction

When a COW object destructor is called, it may not delete the data: if it's shared, the destructor will just decrement the shared data's reference count without destroying it. Only when the destructor of the last object owning the data is called, the data is really destroyed. Just like all other COW-things, this happens transparently to the class users so that you shouldn't care about it.

10.63.3 List of Reference Counted Classes

The following classes in wxWidgets have efficient (i.e. fast) assignment operators and copy constructors since they are reference-counted:

- wxAcceleratorTable
- wxAnimation
- wxBitmap
- wxBrush
- wxCursor
- wxFont
- wxGraphicsBrush
- wxGraphicsContext
- wxGraphicsFont
- wxGraphicsMatrix
- wxGraphicsPath
- wxGraphicsPen
- wxIcon
- wxImage
- wxMetafile
- wxPalette
- wxPen
- wxRegion
- wxString
- wxVariant
- wxVariantData

Note that the list above reports the objects which are reference counted in all ports of wxWidgets; some ports may use this technique also for other classes.

All the objects implement a function **IsOk()** to test if they are referencing valid data; when the objects are in uninitialized state, you can only use the **IsOk()** getter; trying to call any other getter, e.g. wxBrush::GetStyle() on the wxNullBrush object, will result in an assert failure in debug builds.

10.63.4 Making Your Own Reference Counted Class

Reference counting can be implemented easily using wxObject or using the intermediate wxRefCounter class directly. Alternatively, you can also use the wxObjectDataPtr<T> template.

First, derive a new class from wxRefCounter (or wxObjectRefData when using a wxObject derived class) and put the memory-consuming data in it.

Then derive a new class from wxObject and implement there the public interface which will be seen by the user of your class. You'll probably want to add a function to your class which does the cast from wxObjectRefData to your class-specific shared data. For example:

```
MyClassRefData* GetData() const
{
    return wx_static_cast(MyClassRefData*, m_refData);
}
```

In fact, any time you need to read the data from your wxObject-derived class, you will need to call this function.

Note

Any time you need to actually modify the data placed inside your wxObject derived class, you must first call the wxObject::UnShare() function to ensure that the modifications won't affect other instances which are eventually sharing your object's data.

10.64 wxMBConv Overview

The wxMBConv classes in wxWidgets enable an Unicode-aware application to easily convert between Unicode and the variety of 8-bit encoding systems still in use.

See also

Text Conversion

10.64.1 Background: The Need for Conversion

As programs are becoming more and more globalized, and users exchange documents across country boundaries as never before, applications increasingly need to take into account all the different character sets in use around the world. It is no longer enough to just depend on the default byte-sized character set that computers have traditionally used.

A few years ago, a solution was proposed: the Unicode standard. Able to contain the complete set of characters in use in one unified global coding system, it would resolve the character set problems once and for all.

But it hasn't happened yet, and the migration towards Unicode has created new challenges, resulting in "compatibility encodings" such as UTF-8. A large number of systems out there still depends on the old 8-bit encodings, hampered by the huge amounts of legacy code still widely deployed. Even sending Unicode data from one Unicode-aware system to another may need encoding to an 8-bit multibyte encoding (UTF-7 or UTF-8 is typically used for this purpose), to pass unhindered through any traditional transport channels.

10.64.2 Background: The wxString Class

Todo rewrite this overview; it's not up2date with wxString changes

If you have compiled wxWidgets in Unicode mode, the wxChar type will become identical to wchar_t rather than char, and a wxString stores wxChars. Hence, all wxString manipulation in your application will then operate on Unicode strings, and almost as easily as working with ordinary char strings (you just need to remember to use the wxT() macro to encapsulate any string literals).

But often, your environment doesn't want Unicode strings. You could be sending data over a network, or processing a text file for some other application. You need a way to quickly convert your easily-handled Unicode data to and from a traditional 8-bit encoding. And this is what the wxMBConv classes do.

10.64.3 wxMBConv Classes

The base class for all these conversions is the wxMBConv class (which itself implements standard libc locale conversion). Derived classes include wxMBConvLibc, several different wxMBConvUTFxxx classes, and wxCSConv, which implement different kinds of conversions. You can also derive your own class for your own custom encoding and use it, should you need it. All you need to do is override the MB2WC and WC2MB methods.

10.64.4 wxMBConv Objects

Several of the wxWidgets-provided wxMBConv classes have predefined instances (wxConvLibc, wxConvFileName, wxConvUTF7, wxConvUTF8, wxConvLocal). You can use these predefined objects directly, or you can instantiate your own objects.

A variable, wxConvCurrent, points to the conversion object that the user interface is supposed to use, in the case that the user interface is not Unicode-based (like with GTK+ 1.2). By default, it points to wxConvLibc or wxConv↩ Local, depending on which works best on the current platform.

10.64.5 wxCSConv

The wxCSConv class is special because when it is instantiated, you can tell it which character set it should use, which makes it meaningful to keep many instances of them around, each with a different character set (or you can create a wxCSConv instance on the fly).

The predefined wxCSConv instance, wxConvLocal, is preset to use the default user character set, but you should rarely need to use it directly, it is better to go through wxConvCurrent.

10.64.6 Converting Strings

Once you have chosen which object you want to use to convert your text, here is how you would use them with wxString. These examples all assume that you are using a Unicode build of wxWidgets, although they will still compile in a non-Unicode build (they just won't convert anything).

Example 1: Constructing a wxString from input in current encoding.

```
wxString str(input_data, *wxConvCurrent);
```

Example 2: Input in UTF-8 encoding.

```
wxString str(input_data, wxConvUTF8);
```

Example 3: Input in KOI8-R. Construction of wxCSConv instance on the fly.

```
wxString str(input_data, wxCSConv(wxT("koi8-r")));
```

Example 4: Printing a wxString to stdout in UTF-8 encoding.

```
puts(str.mb_str(wxConvUTF8));
```

Example 5: Printing a wxString to stdout in custom encoding. Using preconstructed wxCSConv instance.

```
wxCSConv cust(user_encoding);
printf("Data: %s\n", (const char*) str.mb_str(cust));
```

Note

Since mb_str() returns a temporary wxCharBuffer to hold the result of the conversion, you need to explicitly cast it to const char* if you use it in a vararg context (like with printf).

10.64.7 Converting Buffers

If you have specialized needs, or just don't want to use wxString, you can also use the conversion methods of the conversion objects directly. This can even be useful if you need to do conversion in a non-Unicode build of wxWidgets; converting a string from UTF-8 to the current encoding should be possible by doing this:

```
wxString str(wxConvUTF8.cMB2WC(input_data), *wxConvCurrent);
```

Here, cMB2WC of the UTF8 object returns a wxWCharBuffer containing a Unicode string. The wxString constructor then converts it back to an 8-bit character set using the passed conversion object, *wxConvCurrent. (In a Unicode build of wxWidgets, the constructor ignores the passed conversion object and retains the Unicode data.)

This could also be done by first making a wxString of the original data:

```
wxString input_str(input_data);
wxString str(input_str.wc_str(wxConvUTF8), *wxConvCurrent);
```

To print a wxChar buffer to a non-Unicode stdout:

```
printf("Data: %s\n", (const char*) wxConvCurrent->cWX2MB(unicode_data));
```

If you need to do more complex processing on the converted data, you may want to store the temporary buffer in a local variable:

```
const wxWX2MBbuf tmp_buf = wxConvCurrent->cWX2MB(unicode_data);
const char *tmp_str = (const char*) tmp_buf;
printf("Data: %s\n", tmp_str);
process_data(tmp_str);
```

If a conversion had taken place in cWX2MB (i.e. in a Unicode build), the buffer will be deallocated as soon as tmp_buf goes out of scope. The macro wxWX2MBbuf reflects the correct return value of cWX2MB (either char* or wxCharBuffer), except for the const.

10.65 Writing Non-English Applications

This article describes how to write applications that communicate with the user in a language other than English.

Unfortunately many languages use different charsets under Unix and Windows (and other platforms, to make the situation even more complicated). These charsets usually differ in so many characters that it is impossible to use the same texts under all platforms.

The wxWidgets library provides a mechanism that helps you avoid distributing many identical, only differently encoded, packages with your application (e.g. help files and menu items in iso8859-13 and windows-1257). Thanks to this mechanism you can, for example, distribute only iso8859-13 data and it will be handled transparently under all systems.

Please read the Internationalization which describes the locales concept.

In the following text, wherever *iso8859-2* and *windows-1250* are used, any encodings are meant and any encodings may be substituted there.

10.65.1 Locales

The best way to ensure correctly displayed texts in a GUI across platforms is to use locales. Write your in-code messages in English or without diacritics and put real messages into the message catalog (see Internationalization).

A standard .po file begins with a header like this:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 1999-02-19 16:03+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: ENCODING\n"
```

Note this particular line:

```
"Content-Type: text/plain; charset=CHARSET\n"
```

It specifies the charset used by the catalog. All strings in the catalog are encoded using this charset.

You have to fill in proper charset information. Your .po file may look like this after doing so:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR Free Software Foundation, Inc.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 1999-02-19 16:03+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=iso8859-2\n"
"Content-Transfer-Encoding: 8bit\n"
```

(Make sure that the header is **not** marked as *fuzzy*.)

wxWidgets is able to use this catalog under any supported platform (although iso8859-2 is a Unix encoding and is normally not understood by Windows).

How is this done? When you tell the wxLocale class to load a message catalog that contains a correct header, it checks the charset. The catalog is then converted to the charset used (see wxLocale::GetSystemEncoding and wxLocale::GetSystemEncodingName) by the user's operating system.

10.65.2 Non-English Strings or 8-bit Characters in Source

By convention, you should only use characters without diacritics (i.e. 7-bit ASCII strings) for msgids in the source code and write them in English.

If you port software to wxWidgets, you may be confronted with legacy source code containing non-English string literals. Instead of translating the strings in the source code to English and putting the original strings into message catalog, you may configure wxWidgets to use non-English msgids and translate to English using message catalogs:

- If you use the program `xgettext` to extract the strings from the source code, specify the option `-from-code=<source code charset>`.
- Specify the source code language and charset as arguments to wxLocale::AddCatalog. For example:

```
locale.AddCatalog(wxT("myapp"), wxLANGUAGE_GERMAN, wxT("iso-8859-1"));
```

10.65.3 Font Mapping

You can use wxMBConv Overview and wxFontMapper to display text:

```
if (!wxFontMapper::Get()->IsEncodingAvailable(enc, facename))
{
    wxFontEncoding alternative;
    if (wxFontMapper::Get()->GetAltForEncoding(enc, &alternative,
                                               facename, false))
    {
        wxCSConv convFrom(wxFontMapper::Get()->GetEncodingName(enc));
        wxCSConv convTo(wxFontMapper::Get()->GetEncodingName(alternative));
        text = wxString(text.mb_str(convFrom), convTo);
    }
    else
        ...failure (or we may try iso8859-1/7bit ASCII)...
}
...display text...
```

10.65.4 Converting Data

You may want to store all program data (created documents etc.) in the same encoding, let's say `utf-8`. You can use wxCSConv to convert data to the encoding used by the system your application is running on (see wxLocale::GetSystemEncoding).

10.65.5 Help Files

If you're using wxHtmlHelpController there is no problem at all. You only need to make sure that all the HTML files contain the META tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso8859-2">
```

Also, the hhp project file needs one additional line in the OPTIONS section:

```
Charset=iso8859-2
```

This additional entry tells the HTML help controller what encoding is used in contents and index tables.

10.66 Debugging

Various classes, functions and macros are provided in wxWidgets to help you debug your application.

Assertion macros allow you to insert various checks in your application which can be compiled out or disabled in release builds but are extremely useful while developing. Logging functions are also provided which are useful for inserting traces into your application code as well as debugging. Both assertions and debug logging are also used by wxWidgets itself so you may encounter them even if you don't use either of these features yourself.

See also

wxLog, Logging, Debugging macros

10.66.1 Configuring Debug Support

Starting with wxWidgets 2.9.1 debugging features are always available by default (and not only in a special "debug" build of the library) and you need to predefine wxDEBUG_LEVEL symbol as 0 when building both the library and your application to remove them completely from the generated object code. However the debugging features are disabled by default when the application itself is built with NDEBUG defined (i.e. in "release" or "production" mode) so there is no need to do this, unless the resources of the system your application will be running on are unusually constrained (notice that when asserts are disabled their condition is not even evaluated so the only run-time cost is a single condition check and the extra space taken by the asserts in the code).

This automatic deactivation of debugging code is done by IMPLEMENT_APP() macro so if you don't use you may need to explicitly call wxDISABLE_DEBUG_SUPPORT() yourself.

Also notice that it is possible to build your own application with a different value of wxDEBUG_LEVEL than the one which was used for wxWidgets itself. E.g. you may be using an official binary version of the library which will have been compiled with default

```
wxDEBUG_LEVEL == 1
```

but still predefine wxDEBUG_LEVEL as 0 for your own code.

On the other hand, if you do want to keep the asserts even in production builds, you will probably want to override the handling of assertion failures as the default behaviour which pops up a message box notifying the user about the problem is usually inappropriate. Use wxSetAssertHandler() to set up your own custom function which should be called instead of the standard assertion failure handler. Such function could log an appropriate message in the application log file or maybe notify the user about the problem in some more user-friendly way.

10.66.2 Assertion Macros

wxASSERT(), wxFAIL(), wxCHECK() as well as their other variants (see Debugging macros) are similar to the standard assert() macro but are more flexible and powerful. The first of them is equivalent to assert() itself, i.e. it

simply checks a condition and does nothing if it is true. The second one is equivalent to checking an always false condition and is supposed to be used for code paths which are supposed to be inaccessible (e.g. `default` branch of a `switch` statement which should never be executed). Finally, the wxCHECK() family of macros verifies the condition just as wxASSERT() does and performs some action such returning from the function if it fails – thus, it is useful for checking the functions preconditions.

All of the above functions exist in `_MSG` variants which allow you to provide a custom message which will be shown (or, more generally, passed to the assert handler) if the assertion fails, in addition to the usual file and line number information and the condition itself.

Example of using an assertion macro:

```
void GetTheAnswer(int *p)
{
    wxCHECK_RET( p, "pointer can't be NULL in GetTheAnswer()" );

    *p = 42;
};
```

If the condition is false, i.e. `p` is NULL, the assertion handler is called and, in any case (even when wxDEBUG_L↩ EVEL is 0), the function returns without dereferencing the NULL pointer on the next line thus avoiding a crash.

The default assertion handler behaviour depends on whether the application using wxWidgets was compiled in release build (with `NDEBUG` defined) or debug one (without) but may be changed in either case as explained above. If it wasn't changed, then nothing will happen in the release build and a message box showing the information about the assert as well as allowing to stop the program, ignore future asserts or break into the debugger is shown. On the platforms where wxStackWalker is supported the message box will also show the stack trace at the moment when the assert failed often allowing you to diagnose the problem without using the debugger at all. You can see an example of such message box in the Exception Sample.

10.66.3 Logging Functions

You can use the wxLogDebug and wxLogTrace functions to output debugging information in debug mode; it will do nothing for non-debugging code.

10.67 Window Styles

Window styles are used to specify alternative behaviour and appearances for windows, when they are created.

The symbols are defined in such a way that they can be combined in a 'bit-list' using the C++ *bitwise-or* operator.

For example:

```
wxCAPTION | wxMINIMIZE_BOX | wxMAXIMIZE_BOX |
      wxRESIZE_BORDER
```

For the window styles specific to each window class, please see the documentation for the window.

Most windows can use the generic styles listed for wxWindow in addition to their own styles.

10.68 Window Deletion

Window deletion can be a confusing subject, so this overview is provided to help make it clear when and how you delete windows, or respond to user requests to close windows.

See also

wxCloseEvent, wxWindow

10.68.1 Sequence of Events During Window Deletion

When the user clicks on the system close button or system close command, in a frame or a dialog, wxWidgets calls wxWindow::Close. This in turn generates an EVT_CLOSE event: see wxCloseEvent.

It is the duty of the application to define a suitable event handler, and decide whether or not to destroy the window. If the application is for some reason forcing the application to close (wxCloseEvent::CanVeto returns false), the window should always be destroyed, otherwise there is the option to ignore the request, or maybe wait until the user has answered a question before deciding whether it is safe to close. The handler for EVT_CLOSE should signal to the calling code if it does not destroy the window, by calling wxCloseEvent::Veto. Calling this provides useful information to the calling code.

The wxCloseEvent handler should only call wxWindow::Destroy to delete the window, and not use the `delete` operator. This is because for some window classes, wxWidgets delays actual deletion of the window until all events have been processed, since otherwise there is the danger that events will be sent to a non-existent window.

As reinforced in the next section, calling Close does not guarantee that the window will be destroyed. Call wx↩ Window::Destroy if you want to be certain that the window is destroyed.

10.68.2 Closing Windows

Your application can either use wxWindow::Close event just as the framework does, or it can call wxWindow::↩ Destroy directly. If using Close(), you can pass a true argument to this function to tell the event handler that we definitely want to delete the frame and it cannot be vetoed.

The advantage of using Close instead of Destroy is that it will call any clean-up code defined by the EVT_CLO↩ SE handler; for example it may close a document contained in a window after first asking the user whether the work should be saved. Close can be vetoed by this process (return false), whereas Destroy definitely destroys the window.

10.68.3 Default Window Close Behaviour

The default close event handler for wxDialog simulates a Cancel command, generating a wxID_CANCEL event. Since the handler for this cancel event might itself call Close, there is a check for infinite looping. The default handler for wxID_CANCEL hides the dialog (if modeless) or calls EndModal(wxID_CANCEL) (if modal). In other words, by default, the dialog *is* not destroyed (it might have been created on the stack, so the assumption of dynamic creation cannot be made).

The default close event handler for wxFrame destroys the frame using Destroy().

10.68.4 User Calls to Exit From a Menu

What should I do when the user calls up Exit from a menu? You can simply call wxWindow::Close on the frame. This will invoke your own close event handler which may destroy the frame.

You can do checking to see if your application can be safely exited at this point, either from within your close event handler, or from within your exit menu command handler. For example, you may wish to check that all files have been saved. Give the user a chance to save and quit, to not save but quit anyway, or to cancel the exit command altogether.

10.68.5 Exiting the Application Gracefully

A wxWidgets application automatically exits when the last top level window (wxFrame or wxDialog), is destroyed. Put any application-wide cleanup code in wxApp::OnExit (this is a virtual function, not an event handler).

10.68.6 Automatic Deletion of Child Windows

Child windows are deleted from within the parent destructor. This includes any children that are themselves frames or dialogs, so you may wish to close these child frame or dialog windows explicitly from within the parent close handler.

10.68.7 Other Kinds of Windows

So far we've been talking about 'managed' windows, i.e. frames and dialogs. Windows with parents, such as controls, don't have delayed destruction and don't usually have close event handlers, though you can implement them if you wish. For consistency, continue to use the wxWindow::Destroy function instead of the `delete` operator when deleting these kinds of windows explicitly.

10.69 Environment Variables

This section describes all environment variables that affect execution of wxWidgets programs.

WXTRACE	(Debug build only.) This variable can be set to a comma-separated list of trace masks used in wxLogTrace calls; wxLog::AddTraceMask is called for every mask in the list during wxWidgets initialization.
WXPREFIX	(Unix only.) Overrides installation prefix. Normally, the prefix is hard-coded and is the same as the value passed to `configure` via the `-prefix` switch when compiling the library (typically `/usr/local` or `/usr`). You can set WXPREFIX if you are for example distributing a binary version of an application and you don't know in advance where it will be installed.

10.70 Creating a Custom Widget

Typically combining the existing Controls controls in wxDialogs and wxFrames is sufficient to fullfill any GUI design.

Using the wxWidgets standard controls makes your GUI looks native on all ports and is obviously easier and faster.

However there are situations where you need to show some particular kind of data which is not suited to any existing control. In these cases rather than hacking an existing control for something it has not been conceived for, it's better to write a new widget.

10.70.1 Writing a Custom Widget

There are at least two very different ways to implement a new widget.

The first is to build it upon wxWidgets existing classes, thus deriving it from wxControl or wxWindow. In this way you'll get a **generic** widget. This method has the advantage that writing a single implementation works on all ports; the disadvantage is that it the widget will look the same on all platforms, and thus it may not integrate well with the native look and feel.

The second method is to build it directly upon the native toolkits of the platforms you want to support (e.g. GTK+, Carbon and GDI). In this way you'll get a **native** widget. This method in fact has the advantage of a native look and feel but requires different implementations and thus more work.

In both cases you'll want to better explore some hot topics like:

- Window Sizing Overview
- Custom Event Summary to implement your custom widget's events.

You will probably need also to gain some familiarity with the wxWidgets sources, since you'll need to interface with some undocumented wxWidgets internal mechanisms.

Writing a Generic Widget

Generic widgets are typically derived from wxControl or wxWindow. They are easy to write. The typical "template" is as follows:

```
enum MySpecialWidgetStyles
{
    SWS_LOOK_CRAZY = 1,
    SWS_LOOK_SERIOUS = 2,
    SWS_SHOW_BUTTON = 4,

    SWS_DEFAULT_STYLE = (SWS_SHOW_BUTTON|SWS_LOOK_SERIOUS)
};

class MySpecialWidget : public wxControl
{
public:
    MySpecialWidget() { Init(); }

    MySpecialWidget(wxWindow *parent,
                    wxWindowID winid,
                    const wxString& label,
                    const wxPoint& pos = wxDefaultPosition,
                    const wxSize& size = wxDefaultSize,
                    long style = SWS_DEFAULT_STYLE,
                    const wxValidator& val = wxDefaultValidator,
                    const wxString& name = "MySpecialWidget")
    {
        Init();

        Create(parent, winid, label, pos, size, style, val, name);
    }

    bool Create(wxWindow *parent,
                wxWindowID winid,
                const wxString& label,
                const wxPoint& pos = wxDefaultPosition,
                const wxSize& size = wxDefaultSize,
                long style = SWS_DEFAULT_STYLE,
                const wxValidator& val = wxDefaultValidator,
                const wxString& name = wxCollapsiblePaneNameStr);

    // accessors...

protected:

    void Init() {
        // init widget's internals...
    }

    virtual wxSize DoGetBestSize() const {
        // we need to calculate and return the best size of the widget...
    }

    void OnPaint(wxPaintEvent&) {
        // draw the widget on a wxDC...
    }

private:
    DECLARE_DYNAMIC_CLASS(MySpecialWidget)
    DECLARE_EVENT_TABLE()
};
```

Writing a Native Widget

Writing a native widget is typically more difficult as it requires you to know the APIs of the platforms you want to support. See Native Toolkit Documentation for links to the documentation manuals of the various toolkits.

The organization used by wxWidgets consists in:

- declaring the common interface of the control in a generic header, using the 'Base' postfix; e.g. MySpecial↩ WidgetBase. See for example the wxWidgets' `"wx/button.h"` file.

- declaring the real widget class inheriting from the Base version in platform-specific headers; see for example the wxWidgets' "`wx/gtk/button.h`" file.
- separating the different implementations in different source files, putting all common stuff in a separate source. See for example the wxWidgets' "`src/common/btncmn.cpp`", "`src/gtk/button.cpp`" and "`src/msw/button.cpp`" files.

Chapter 11

Translations to Other Languages

wxWidgets uses a certain number of user-readable strings such as "help" or "Load file" which should be translated to the users language if it is different from English.

wxWidgets has built in support for internationalization (*i18n* from now on) which allows for this to happen automatically if the translations to the current language are available.

You may find here the list of all existing translations with the addresses of the official translators whom you should contact if you would like to submit any corrections to the translations for your language.

Also, please see How to Help if you would like to translate wxWidgets to your language if it is not mentioned here (or to help with one which already is - it is quite helpful to have several translators for one language at least for proof reading).

11.1 Available Translations

Below is the table containing the list of languages supported by wxWidgets. The columns of this table have the obvious meaning: in each row you will see the language, the official translator (if any) for it and the status of the translations.

Please note that email addresses in the table below are intentionally invalid to foil spam robots, remove one @ from them.

Language	Status	Translator(s)
Afrikaans	68%	Petri Jooste
Albanian	75%	Besnik Bleta
Arabic	45%	Abdullah Abouzekry
Basque	100%	3ARRANO Euskalgintza Taldea, Xabier Aramendi
Catalan	65%	Pau Bosch i Crespo, Robert Millan
Chinese (simplified)	100%	mrfx, Liu XiaoXi, Huang Jiawei, William Jiang

Chinese (traditional)	100%	pal.tw
Czech	100%	Vaclav Slavik, Herbert Breunung, Zbyněk Schwarz
Danish	74%	Leif Jensen, Henrik Ræder Clausen, Morten Råbjerg Ulrich
Dutch	100%	Patrick Hubers, Gideon van Melle, Thomas De Rocker
English (UK)	100%	N/A
Finnish	91%	Kaj G Backas, Lauri Nurmi, Jaakko Salli, Elias Julkunen, Jani Kinnunen
French (standard)	96%	Stephane Junique, Lionel Allorge, Gilles Guyot
Galician	76%	Leandro Regueiro, Adrián González Alba
German	100%	Daniel Reith, Gerhard Gruber, Stefan Hedemann, Dr. Detlev Reymann, Mark Johnson, Martin Jost, Herbert Breunung, Ch. Buck, Max Christian Pohle, Thomas Krebs
Greek	73%	Tsolakos Stavros, Nassos Yiannopoulos
Hindi	83%	Dhananjaya Sharma, Priyank Bolia
Hungarian	76%	Végh János Dr.
Indonesian	100%	Bambang Purnomosidi D. P., Rahmat Bambang
Italian	100%	Mattia Barbon, Marco Cavallini, (Koan Software), Stefano

Japanese	97%	James Bishop, Hiroshi Saito, Suzumizaki-Kimitaka, Y. KABA.
Korean	85%	Sungkee Jung
Latvian	90%	Lauris Bukshis
Lithuanian	15%	Pieter
Malay	89%	Mahrazi Mohd Kamal
Nepali	100%	Him Prasad Gautam
Norwegian Bokmal	74%	Hans F. Nordhaug
Polish	94%	Piotr Mackowiak, Janusz Piwowarski, ABX, Michał Trzebiatowski, Grzegorz Zlotowicz
Portuguese (pt)	89%	Bernardo Santos Wernesback, Mario Pereira, Antonio Cardoso Martins, Carlos Gonçalves
Portuguese (pt_BR)	100%	E.A. Tacao, José Eduardo de Carvalho Diniz, Adiel Mittmann, Allann Jones, Felipe
Romanian	100%	Cătălin Răceanu, Adrian Hăisan, Manuel Ciosici
Russian	74%	Dennis Prochko, Roman Rolinsky, Vadim Zeitlin, Andrew V. Samoilov
Slovak	74%	Ivan Masar
Slovenian	97%	Roman Plevel, Martin Srebotnjak
Spanish	89%	Guillermo Rodriguez Garcia, JSJ, Francisco Vila, Adrián González Alba

Swedish	98%	Jonas Rydberg, Kaj G Backas
Tamil	100%	DINAKAR T.D.
Turkish	100%	Hakki Dogusan, Kaya Zeren
Ukrainian	100%	Eugene Manko, Yuri Chornoivan, Ylia K
Valencian (ca@valencia)	65%	Robert Millan
Vietnamese	100%	Tran Ngoc Quan

11.2 How to Help

wxWidgets uses the standard GNU gettext tools for i18n so if you are already familiar with them you shouldn't have any problems with working on wxWidgets translations.

Here are the steps you should follow:

1. Get the latest version of the file `locale/wxstd.pot` from the wxWidgets source tree: if you're using Subversion or the daily snapshots you should already have it. Otherwise you can always retrieve it directly from the Subversion repository via the Web interface.
2. Rename it to `XY.po` where `"XY"` is the 2 letter ISO 639-2 language code for your language.
3. Translate the strings in this file using either your favourite text editor or a specialized tool such as Vaclav Slavik's excellent poEdit utility.
4. Verify that your translations can at least be compiled (even if they are yet incomplete) by running `msgfmt -v XY.po` command: please note that you *must* use the `-v` option. In particular, please fill the header fields because `msgfmt` doesn't accept the default values for them.
5. Send the finished translation to Vadim Zeitlin and it will be added to the next wxWidgets release or snapshot.

In addition, please consider subscribing to the very low volume wxWidgets translators mailing list on which the news especially important for the translators are announced.

Thank you in advance for your help!

Chapter 12

Utilities Overview

In addition to the wxWidgets libraries (see Library List), some utilities are available to the users in the `utils` hierarchy (even if some of them are explicitly conceived for wxWidgets maintainance and will probably be of little use to others).

Please note that these utilities do represent only the utilities developed and maintained by the wxWidgets team. There are lots of other user-contributed and user-maintained packages; see the wxWidgets download page: `http://www.wxwidgets.org/downloads` or directly `http://wxcode.sourceforge.net` or `http://www.wxcommunity.com/` .

12.1 Emulator

Xnest-based display emulator for X11-based PDA applications.

This program can be found in `utils/emulator`.

12.2 Help Viewer

Helpview is a program for displaying wxWidgets HTML Help files. In many cases, you may wish to use the wxWidgets HTML Help classes from within your application, but this provides a handy stand-alone viewer. See wxHTML Overview for more details.

You can find Helpview in `utils/helpview`.

12.3 HHP2Cached

This utility creates a "cached" version of a .hhp file; using cached .hhp files in wxHtmlHelpController can dramatically improve the performance of the help viewer. See wxHtmlHelpController for more details.

You can find HHP2Cached in `utils/hhp2cached`.

12.4 Interface Checker

This utility compares the wxWidgets real interface contained in the `include` hierarchy with the wxWidgets interface used for documentation purposes and kept in the `interface` hierarchy.

Ifacecheck warns about incoherences (mainly wrong prototype signatures) and can even correct them automatically. It uses the XML outputs of the gccxml utility (see `http://www.gccxml.org`) and of the Doxygen utility (see `http://www.doxygen.org`) to do the comparisons.

It's explicitly designed for wxWidgets documentation needs and is probably of little use for anything else than wx↩ Widgets docs reviewing.

You can find it in `utils/ifacecheck`.

12.5 Screenshot Generator

This utility automates the process of taking screenshots of various GUI components for use in the HTML documentation of wxWidgets.

You can find it in `utils/screenshotgen`.

12.6 wxWidgets XML Resource Compiler

This utility allows the user to compile *binary* versions of their XRC files, which are compressed and can be loaded faster than plain XRC files. See XML Based Resource System (XRC) for more info.

You can find it under `utils/wxrc`.

Chapter 13

Changes Since wxWidgets 2.8

This topic describes backwards-incompatible changes in wxWidgets 3.0 compared to the last stable release and is very important to read if you are updating from the 2.8 or an older version.

And even if you hadn't used any previous version of wxWidgets and are starting directly with 3.0, it can still be useful to have at least a quick look at it just to know that some of the older examples and tutorials may not be applicable any more to wxWidgets 3.0.

The incompatible changes can be grouped into the following categories:

- Unicode-related Changes
- Miscellaneous Other Changes

13.1 Unicode-related Changes

If you used Unicode build of wxWidgets 2.8 or previous version, please read Unicode Support in wxWidgets for the details about how the API changed in 3.0 as a lot of the information which was correct before doesn't apply any longer.

For example, the notorious (due to the confusion they created) macros `wxT()` and `_T()` are not needed at all any longer. Basically, you can remove them from any code which used them. On the other hand, there is no particular harm in leaving them neither as the code will still compile and work correctly – you only need to remove them if you think that your code looks tidier without them. You also don't need to use `wxChar` any longer but can directly use the standard `wchar_t` type even if, again, `wxChar` continues to work.

The most serious backwards-incompatible change is related to the change of return type of wxString::c_str() method: it returns a special proxy object instead of a simple `char*` or `wchar_t*` now. Because of this, you cannot pass its result to any standard vararg functions such as `printf()` any more as described in Unicode-Related Compilation Errors. All wxWidgets functions, such as wxPrintf(), wxLogMessage() &c still work with it, but passing it to `printf()` will now result in a crash. It is strongly advised to recompile your code with a compiler warning about passing non-POD objects to vararg functions, such as g++.

The change of the type of wxString::c_str() can also result in compilation errors when passing its result to a function overloaded to take both narrow and wide strings and in this case you must select the version which you really want to use, e.g.:

```
void OpenLogFile(const char *filename);
void OpenLogFile(const wchar_t *filename);

wxString s;
OpenLogFile(s);             // ERROR: ambiguity
OpenLogFile(s.c_str());     // ERROR: ambiguity
OpenLogFile(s.wx_str());    // OK: function called depends on the build
OpenLogFile(s.mb_str());    // OK: always calls narrow string overload
OpenLogFile(s.wc_str());    // OK: always calls wide string overload
```

A common example of such problem arises with `std::fstream` class constructor in Microsoft Visual C++ standard library implementation. In addition to a constructor from `const char *` which this class must have, it also provides a constructor taking a wide character file name. Because of this, code like the following

```
#include <fstream>

void MyFunc(const wxString& filename)
{
    std::ifstream ifs(filename.c_str());
    ...
}
```

does not compile when using Microsoft Visual C++ and needs to be changed to use mb_str() (which will not work for file names containing Unicode characters, consider using wxWidgets classes and functions to work with such file names as they are not supported by standard C++ library).

The other class of incompatible changes is due to modifying some virtual methods to use `wxString` parameters instead of `const wxChar*` ones to make them accept both narrow and wide strings. This is not a problem if you simply call these functions but you need to change the signature of the derived class versions if you override them as otherwise they wouldn't be called any more. Again, the best way to ensure that this problem doesn't arise is to rebuild your code using a compiler which warns about function signature mismatch (you can use `-Woverloaded-virtual` g++ option).

Finally, a few structure fields, notable `wxCmdLineEntryDesc::shortName`, `longName` and `description` fields have been changed to be of type `const char*` instead of `const wxChar*` so you will need to remove `wxT()` or `_T()` if you used it with their initializers.

13.2 Miscellaneous Other Changes

- Default location of wxFileConfig files has changed under Windows, you will need to update your code if you access these files directly.
- wxWindow::IsEnabled() now returns false if a window parent (and not necessarily the window itself) is disabled, new function IsThisEnabled() with the same behaviour as old IsEnabled() was added.
- Generating wxNavigationKeyEvent events doesn't work any more under wxGTK (and other platforms in the future), use wxWindow::Navigate() or NavigateIn() instead.
- Sizers distribute only the extra space between the stretchable items according to their proportions and not all available space. We believe the new behaviour corresponds better to user expectations but if you did rely on the old behaviour you will have to update your code to set the minimal sizes of the sizer items to be in the same proportion as the items proportions to return to the old behaviour.
- wxWindow::Freeze/Thaw() are not virtual any more, if you overrode them in your code you need to override DoFreeze/Thaw() instead now.
- wxCalendarCtrl has native implementation in wxGTK, but it has less features than the generic one. The native implementation is used by default, but you can still use wxGenericCalendarCtrl instead of wxCalendarCtrl in your code if you need the extra features.
- wxDocument::FileHistoryLoad() and wxFileHistory::Load() now take const reference to wxConfigBase argument and not just a reference, please update your code if you overrode these functions and change the functions in the derived classes to use const reference as well.
- Calling wxConfig::Write() with an enum value will fail to compile because wxConfig now tries to convert all unknown types to wxString automatically using wxToString() function.

  The simplest solution is to cast the enum value to int, e.g.

  ```
  enum Colour { Red, Green, Blue };

  wxConfig conf;
  conf.Write("MyFavouriteColour", Red);       // ERROR: no match
  conf.Write("MyFavouriteColour", int(Red));  // OK
  ```

Another possibility which exists now is to provide an overload of wxToString() (and wxFromString()) for your own type, e.g.

```
wxString wxToString(Colour col)
{
    return col == Red ? "R" : col == Green ? "G" : "B";
}

bool wxFromString(const wxString& s, Colour* col)
{
    if ( s.length() != 1 )
        return false;

    switch ( s[0].GetValue() )
    {
        case 'R': *col = Red; return true;
        case 'G': *col = Green; return true;
        case 'B': *col = Blue; return true;
    }

    return false;
}
```

Of course, this will change the format of the wxConfig output which may be undesirable.

- wxTE_AUTO_SCROLL style is deprecated as it's always on by default anyhow in the ports which support it so you should simply remove any mentions of it from your code.
- If you use wxScrolled<T>::SetTargetWindow() you must override wxScrolled<T>::GetSizeAvailableFor↩ ScrollTarget() method to compute the size available for the scroll target as function of the main window size, please see the documentation of this method for more details.
- Signature of wxDataViewCustomRenderer::StartDrag() virtual method changed. You will need to change it in your derived renderer class too if you override it.
- wxDataViewCustomRenderer::Activate() and wxDataViewCustomRenderer::LeftClick() were replaced with the new wxDataViewCustomRenderer::ActivateCell() method. You will need to change it in your derived renderer class accordingly.

Chapter 14

Todo List

Page Date and Time

WRITE THIS DOC PARAGRAPH.

Page Samples Overview

Write descriptions for the samples who description started with "This sample demonstrates", they are semi-auto generated.

This sample isn't very didactive; it's more than a set of tests rather than a sample and thus should be rewritten with CppUnit and moved under "tests"

Member wxAuiManagerDock

wxAuiPaneInfo dock direction types used with wxAuiManager.

Member wxAuiManagerEvent::GetDC ()

What is this?

Member wxAuiManagerEvent::SetDC (wxDC *pdc)

What is this?

Member wxBitmap::SetDepth (int depth)

since these functions do not affect the bitmap data, why they exist??

Class wxBitmapComboBox

create wxCB_PROCESS_ENTER rather than reusing wxTE_PROCESS_ENTER!

Member wxClipboard::IsSupported (const wxDataFormat &format)

The name of this function is misleading. This should be renamed to something that more accurately indicates what it does.

Class wxConnectionBase

Document this class.

Class wxDatagramSocket

docme

Class wxDateTimeHolidayAuthority

Write wxDateTimeHolidayAuthority documentation.

Class wxDateTimeWorkDays

Write wxDateTimeWorkDays documentation.

Class wxDC

Precise definition of default/initial state.

Pixelwise definition of operations (e.g. last point of a line not drawn).

Member wxDefaultDateTime

Would it be better to rename this wxNullDateTime so it's consistent with the rest of the "empty/invalid/null" global objects?

Member wxEvtHandler::SearchEventTable (wxEventTable &table, wxEvent &event)

this function in the header is listed as an "implementation only" function; are we sure we want to document it?

Member wxFFile::Eof () const

THIS METHOD MAY CRASH? DOESN'T SOUND GOOD

Member wxFFile::Error () const

THIS METHOD MAY CRASH? DOESN'T SOUND GOOD

Member wxFont::SetNativeFontInfoUserDesc (const wxString &info)

add an example for wxMac

Member wxGridCellAttr::SetDefAttr (wxGridCellAttr *defAttr)

Needs documentation.

Member wxHelpControllerBase::SetViewer (const wxString &viewer, long flags=wxHELP_NETSCAPE)

modernize this function with wxLaunchDefaultBrowser

Member wxHtmlDCRenderer::Render (int x, int y, wxArrayInt &known_pagebreaks, int from=0, int dont_↩ render=false, int to=INT_MAX)

docme

Parameters

from	y-coordinate of the very first visible cell.
dont_render	if true then this method only returns y coordinate of the next page and does not output anything.
to	y-coordinate of the last visible cell.

Member wxHtmlTag::GetBeginPos () const

provide deprecation description

Member wxHtmlTag::GetEndPos1 () const

provide deprecation description

Member wxHtmlTag::GetEndPos2 () const

provide deprecation description

Class wxHtmlTagHandler

describe me

Page wxListCtrl Overview

The wxListCtrl topic overview still needs to be written, sorry.

Page wxMBConv Overview

rewrite this overview; it's not up2date with wxString changes

Member wxMediaCtrl::Seek (wxFileOffset where, wxSeekMode mode=wxFromStart)

Document the wxSeekMode parameter *mode*, and perhaps also the wxFileOffset and wxSeekMode themselves.

Member wxPenCap

use wxPENCAP_ prefix

Member wxPenJoin

use wxPENJOIN_ prefix

Member wxPENSTYLE_STIPPLE_MASK

WHAT's this?

Member wxPENSTYLE_STIPPLE_MASK_OPAQUE

WHAT's this?

Member wxPGProperty::SetValueFromString (const wxString &text, int flags=0)

docme

Member wxPreviewControlBar::CreateButtons ()

which flags??

Member wxPrintout::GetTitle () const

the python note here was wrong

Member wxPropertyGridInterface::GetPropertiesWithFlag (wxArrayPGProperty *targetArr, wxPGProperty::FlagType flags, bool inverse=false, int iterFlags=(wxPG_ITERATE_PROPERTIES|wxPG_ITERATE_HIDDEN|wxPG_ITERATE_CATEGORIES)) const

docme

Parameters

flags	Property flags to use.
inverse	

docme

Parameters

iterFlags	Iterator flags to use. Default is everything expect private children. See wxPropertyGridIterator Flags.

Member wxPropertyGridInterface::GetPropertyValues (const wxString &listname=wxEmptyString, wxPGProperty *baseparent=NULL, long flags=0) const

docme

Parameters

baseparent	

docme

Parameters

flags	Use wxPG_KEEP_STRUCTURE to retain category structure; each sub category will be its own wxVariantList of wxVariant.

Member wxPropertyGridInterface::HideProperty (wxPGPropArg id, bool hide=true, int flags=wxPG_RECURSE)

docme

Parameters

hide	If true, hides property, otherwise reveals it.
flags	By default changes are applied recursively. Set this parameter wxPG_DONT_RECURSE to prevent this.

Member wxPropertyGridInterface::SetPropertyAttribute (wxPGPropArg id, const wxString &attrName, wxVariant value, long argFlags=0)

docme

Parameters

attrName	Text identifier of attribute. See wxPropertyGrid Property Attribute Identifiers.
value	Value of attribute.
argFlags	Optional. Use wxPG_RECURSE to set the attribute to child properties recursively.

Member wxPropertyGridInterface::SetPropertyEditor (wxPGPropArg id, const wxPGEditor *editor)

docme

Parameters

editor	For builtin editors, use wxPGEditor_X, where X is builtin editor's name (TextCtrl, Choice, etc. see wxPGEditor documentation for full list).

Class wxScrolled< T >

review docs for this class replacing SetVirtualSizeHints() with SetMinClientSize().

Member wxScrollWinEvent::GetOrientation () const

wxHORIZONTAL and wxVERTICAL should go in their own enum

Member wxSizerItem::SetInitSize (int x, int y)

docme.

Member wxSizerItem::SetWindow (wxWindow *window)

provide deprecation description

Class wxSocketClient

describe me.

Class wxSocketServer

describe me.

Class wxStyledTextEvent

list styled text ctrl events.

Member wxVListBox::OnDrawBackground (wxDC &dc, const wxRect &rect, size_t n) const

Change this function signature to non-const.

Member wxVListBox::OnDrawItem (wxDC &dc, const wxRect &rect, size_t n) const =0

Change this function signature to non-const.

Member wxVListBox::OnDrawSeparator (wxDC &dc, wxRect &rect, size_t n) const

Change this function signature to non-const.

Member wxWizard::SetBitmapPlacement (int placement)

describe this

Chapter 15

Deprecated List

Member wxApp::MacOpenFile (const wxString &fileName)

This function is kept mostly for backwards compatibility. Please override wxApp::MacOpenFiles method instead in any new code.

Member wxArtProvider::Insert (wxArtProvider *provider)

Use PushBack() instead.

Member wxCalendarCtrl::EnableYearChange (bool enable=true)

Member wxComboCtrl::GetTextIndent () const

Use GetMargins() instead.

Member wxComboCtrl::HidePopup (bool generateEvent=false)

Use Dismiss() instead.

Member wxComboCtrl::SetTextIndent (int indent)

Use SetMargins() instead.

Member wxComboCtrl::ShowPopup ()

Use Popup() instead.

Member wxDataViewCustomRenderer::Activate (wxRect cell, wxDataViewModel *model, const wxDataViewItem &item, unsigned int col)

Use ActivateCell instead.

Member wxDataViewCustomRenderer::LeftClick (wxPoint cursor, wxRect cell, wxDataViewModel *model, const wxDataViewItem &item, unsigned int col)

Use ActivateCell instead.

Member wxDebugContext::GetLevel ()

This is obsolete, replaced by wxLog functionality.

Member wxDebugContext::SetLevel (int level)

This is obsolete, replaced by wxLog functionality.

Member wxDos2UnixFilename (wxChar *s)

Construct a wxFileName with wxPATH_DOS and then use wxFileName::GetFullPath(wxPATH_UNIX) instead.

Member wxFileNameFromPath (const wxString &path)

This function is obsolete, please use wxFileName::SplitPath() instead.

Member wxFindWindowByLabel (const wxString &label, wxWindow *parent=NULL)

Replaced by wxWindow::FindWindowByLabel().

Member wxFindWindowByName (const wxString &name, wxWindow *parent=NULL)

Replaced by wxWindow::FindWindowByName().

Member wxGetEmailAddress (char *buf, int sz)

Use wxGetEmailAddress() instead.

Member wxGetHostName (char *buf, int sz)

Use wxGetHostName() instead.

Member wxGetTempFileName (const wxString &prefix, char *buf=NULL)

This function is obsolete, please use wxFileName::CreateTempFileName() instead.

Member wxGetUserId (char *buf, int sz)

Use wxGetUserId() instead.

Member wxGetUserName (char *buf, int sz)

Use wxGetUserName() instead.

Member wxGetWorkingDirectory (char *buf=NULL, int sz=1000)

This function is deprecated, use wxGetCwd() instead.

Member wxGrid::SetCellAlignment (int align, int row, int col)

Please use SetCellAlignment(row, col, horiz, vert) instead.

Member wxGrid::SetCellTextColour (const wxColour &val, int row, int col)

Please use SetCellTextColour(row, col, colour)

Member wxGrid::SetCellTextColour (const wxColour &colour)

Please use SetDefaultCellTextColour(colour) instead.

Member wxGrid::SetCellValue (const wxString &val, int row, int col)

Please use SetCellValue(int,int,const wxString&) or SetCellValue(const wxGridCellCoords&,const wxString&) instead.

Class wxHashTable

Please note that this class is retained for backward compatibility reasons; you should use wxHashMap.

Member wxHelpControllerBase::DisplayBlock (long blockNo)=0

This function is for backward compatibility only, and applications should use DisplaySection() instead.

Member wxHtmlTag::GetBeginPos () const

Member wxHtmlTag::GetEndPos1 () const

Member wxHtmlTag::GetEndPos2 () const

Member wxIconizeEvent::Iconized () const

This function is deprecated in favour of IsIconized().

Member wxList< T >::Nth (int n) const

This function is deprecated, use Item() instead.

Member wxList< T >::Number () const

This function is deprecated, use wxList::GetCount instead. Returns the number of elements in the list.

Member wxLocale::Init (const wxString &name, const wxString &shortName=wxEmptyString, const wxString &locale=wxEmptyString, bool bLoadDefault=true)

This form is deprecated, use the other one unless you know what you are doing.

Member wxLogTrace (wxTraceMask mask, const char *formatString,...)

This version of wxLogTrace() only logs the message if all the bits corresponding to the *mask* are set in the wxLog trace mask which can be set by calling wxLog::SetTraceMask(). This version is less flexible than wxLogTrace(const char*,const char*,...) because it doesn't allow defining the user trace masks easily. This is why it is deprecated in favour of using string trace masks.

Member wxMBConv::MB2WC (wchar_t *out, const char *in, size_t outLen) const

This function is deprecated, please use ToWChar() instead.

Member wxMBConv::WC2MB (char *buf, const wchar_t *psz, size_t n) const

This function is deprecated, please use FromWChar() instead.

Member wxMenu::Append (int id, const wxString &item, wxMenu *subMenu, const wxString &help↩ String=wxEmptyString)

This function is deprecated, use AppendSubMenu() instead.

Member wxMenuBar::GetLabelTop (size_t pos) const

This function is deprecated in favour of GetMenuLabel() and GetMenuLabelText().

Member wxMenuBar::SetLabelTop (size_t pos, const wxString &label)

This function has been deprecated in favour of SetMenuLabel().

Member wxMenuItem::GetLabel () const

This function is deprecated in favour of GetItemLabelText().

Member wxMenuItem::GetLabelFromText (const wxString &text)

This function is deprecated; please use GetLabelText() instead.

Member wxMenuItem::GetName () const

This function is deprecated. Please use GetItemLabel() or GetItemLabelText() instead.

Member wxMenuItem::GetText () const

This function is deprecated in favour of GetItemLabel().

Member wxMenuItem::SetText (const wxString &text)

This function is deprecated in favour of SetItemLabel().

Member wxNewId ()

Ids generated by it can conflict with the Ids defined by the user code, use `wxID_ANY` to assign ids which are guaranteed to not conflict with the user-defined ids for the controls and menu items you create instead of using this function.

Member wxPG_ATTR_INLINE_HELP

Use "Hint" (wxPG_ATTR_HINT) instead.

Member wxPGProperty::wxDEPRECATED (void AddChild(wxPGProperty *prop))

Use AddPrivateChild() instead.

Member wxPGProperty::wxDEPRECATED (wxString GetValueString(int argFlags=0) const)

Use GetValueAsString() instead.

Member wxPostDelete (wxObject *object)

Replaced by wxWindow::Close(). See the window deletion overview.

Member wxPrintDialogData::SetSetupDialog (bool flag)

This function has been deprecated since version 2.5.4.

Member wxShowEvent::GetShow () const

This function is deprecated in favour of IsShown().

Member wxSizer::Remove (wxWindow *window)

The overload of this method taking a wxWindow* parameter is deprecated as it does not destroy the window as would usually be expected from Remove(). You should use Detach() in new code instead. There is currently no wxSizer method that will both detach and destroy a wxWindow item.

Member wxSizer::SetVirtualSizeHints (wxWindow *window)

This is exactly the same as FitInside() in wxWidgets 2.9 and later, please replace calls to it with FitInside().

Member wxSizerItem::SetSizer (wxSizer *sizer)

This function does not free the old sizer which may result in memory leaks, use AssignSizer() which does free it instead.

Member wxSizerItem::SetSpacer (const wxSize &size)

This function does not free the old spacer which may result in memory leaks, use AssignSpacer() which does free it instead.

Member wxSizerItem::SetWindow (wxWindow *window)

Member wxSocketBase::LastCount () const

This function is kept mostly for backwards compatibility. Use LastReadCount() or LastWriteCount() instead. LastCount() is still needed for use with less commonly used functions: Discard(), Peek(), and Unread().

Member wxSplitPath (const wxString &fullname, wxString *path, wxString *name, wxString *ext)

This function is obsolete, please use wxFileName::SplitPath() instead.

Member wxStreamBuffer::Stream ()

use GetStream() instead

Member wxStyledTextEvent::GetDragText ()

Use GetString() instead.

Member wxStyledTextEvent::GetText () const

Use GetString() instead.

Member wxTextInputStream::ReadString ()

Use ReadLine() or ReadWord() instead.

Member wxThreadHelper::Create (unsigned int stackSize=0)

Use CreateThread() instead.

Member wxToolBar::OnRightClick (int toolId, long x, long y)

This is the old way of detecting tool right clicks; although it will still work, you should use the EVT_TOOL_RC↩ LICKED() macro instead.

Member WXTRACE (format,...)

Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Member wxTrace (const wxString &format,...)

Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Member wxTraceLevel (int level, const wxString &format,...)

Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Member WXTRACELEVEL (level, format,...)

Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Member wxUnix2DosFilename (wxChar *s)

Construct a wxFileName with wxPATH_UNIX and then use wxFileName::GetFullPath(wxPATH_DOS) instead.

Member wxUsleep (unsigned long milliseconds)

This function is deprecated because its name is misleading: notice that the argument is in milliseconds, not microseconds. Please use either wxMilliSleep() or wxMicroSleep() depending on the resolution you need.

Member wxWindow::SetInitialBestSize (const wxSize &size)

Use SetInitialSize() instead.

Member wxWindow::SetPalette (const wxPalette &pal)

use wxDC::SetPalette instead.

Member wxYield ()

This function is kept only for backwards compatibility. Please use the wxAppConsole::Yield method instead in any new code.

Chapter 16

Module Index

16.1 Categories

Here is a list of all modules:

Chapter 17

Hierarchical Index

17.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 18

Class Index

18.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 19

File Index

19.1 File List

Here is a list of all files with brief descriptions:

Chapter 20

Module Documentation

20.1 Application Initialization and Termination

20.1.1 Detailed Description

The functions in this section are used on application startup/shutdown and also to control the behaviour of the main event loop of the GUI programs.

Related macros/global-functions group: Application and Process Management.

Functions

- wxAppDerivedClass & wxGetApp ()

  This function doesn't exist in wxWidgets but it is created by using the wxIMPLEMENT_APP() macro.
- bool wxHandleFatalExceptions (bool doIt=true)

  If doIt is true, the fatal exceptions (also known as general protection faults under Windows or segmentation violations in the Unix world) will be caught and passed to wxApp::OnFatalException.
- bool wxInitialize ()

  This function is used in wxBase only and only if you don't create wxApp object at all.
- void wxUninitialize ()

  This function is for use in console (wxBase) programs only.
- void wxWakeUpIdle ()

  This function wakes up the (internal and platform dependent) idle system, i.e.
- bool wxYield ()

  Calls wxAppConsole::Yield.
- bool wxSafeYield (wxWindow *win=NULL, bool onlyIfNeeded=false)

  Calls wxApp::SafeYield.
- int wxEntry (int &argc, wxChar **argv)

  This function initializes wxWidgets in a platform-dependent way.
- int wxEntry (HINSTANCE hInstance, HINSTANCE hPrevInstance=NULL, char *pCmdLine=NULL, int nCmdShow=SW_SHOWNORMAL)

  *See wxEntry(int&,wxChar**) for more info about this function.*
- void wxInitAllImageHandlers ()

  Initializes all available image handlers.
- bool wxEntryStart (int &argc, wxChar **argv)

  This function can be used to perform the initialization of wxWidgets if you can't use the default initialization code for any reason.
- bool wxEntryStart (HINSTANCE hInstance, HINSTANCE hPrevInstance=NULL, char *pCmdLine=NULL, int nCmdShow=SW_SHOWNORMAL)

*See wxEntryStart(int&,wxChar**) for more info about this function.*

- void wxEntryCleanup ()

  Free resources allocated by a successful call to wxEntryStart().

- bool wxInitialize (int argc=0, wxChar **argv=NULL)

  Initialize the library (may be called as many times as needed, but each call to wxInitialize() must be matched by wxUninitialize()).

20.1.2 Function Documentation

int wxEntry (int & *argc*, wxChar ** *argv*)

This function initializes wxWidgets in a platform-dependent way.

Use this if you are not using the default wxWidgets entry code (e.g. main or WinMain).

For example, you can initialize wxWidgets from an Microsoft Foundation Classes (MFC) application using this function.

Note

This overload of wxEntry is available under all platforms.

See also

wxEntryStart()

Include file:

```
#include <wx/app.h>
```

int wxEntry (HINSTANCE *hInstance*, HINSTANCE *hPrevInstance* = NULL, char * *pCmdLine* = NULL, int *nCmdShow* = SW_SHOWNORMAL)

See wxEntry(int&,wxChar**) for more info about this function.

Notice that under Windows CE platform, and only there, the type of *pCmdLine* is `wchar_t *`, otherwise it is `char *`, even in Unicode build.

Remarks

To clean up wxWidgets, call wxApp::OnExit followed by the static function wxApp::CleanUp. For example, if exiting from an MFC application that also uses wxWidgets:

```
1 int CTheApp::ExitInstance()
2 {
3     // OnExit isn't called by CleanUp so must be called explicitly.
4     wxTheApp->OnExit();
5     wxApp::CleanUp();
6
7     return CWinApp::ExitInstance();
8 }
```

Include file:

```
#include <wx/app.h>
```

void wxEntryCleanup ( )

Free resources allocated by a successful call to wxEntryStart().

Include file:

```
#include <wx/init.h>
```

bool wxEntryStart (int & *argc*, wxChar ** *argv*)

This function can be used to perform the initialization of wxWidgets if you can't use the default initialization code for any reason.

If the function returns true, the initialization was successful and the global wxApp object wxTheApp has been created. Moreover, wxEntryCleanup() must be called afterwards. If the function returns false, a catastrophic initialization error occurred and (at least the GUI part of) the library can't be used at all.

Notice that parameters `argc` and `argv` may be modified by this function.

Include file:

```
#include <wx/init.h>
```

bool wxEntryStart (HINSTANCE *hInstance*, HINSTANCE *hPrevInstance* = `NULL`, char * *pCmdLine* = `NULL`, int *nCmdShow* = `SW_SHOWNORMAL`)

See wxEntryStart(int&,wxChar**) for more info about this function.

This is an additional overload of wxEntryStart() provided under MSW only. It is meant to be called with the parameters passed to WinMain().

Note

Under Windows CE platform, and only there, the type of *pCmdLine* is `wchar_t *`, otherwise it is `char *`, even in Unicode build.

Availability: only available for the wxMSW port.

Include file:

```
#include <wx/init.h>
```

wxAppDerivedClass& wxGetApp ( )

This function doesn't exist in wxWidgets but it is created by using the wxIMPLEMENT_APP() macro.

Thus, before using it anywhere but in the same module where this macro is used, you must make it available using wxDECLARE_APP().

The advantage of using this function compared to directly using the global wxTheApp pointer is that the latter is of type wxApp* and so wouldn't allow you to access the functions specific to your application class but not present in wxApp while wxGetApp() returns the object of the right type.

Include file:

```
#include <wx/app.h>
```

bool wxHandleFatalExceptions (bool *doIt* = `true`)

If *doIt* is true, the fatal exceptions (also known as general protection faults under Windows or segmentation violations in the Unix world) will be caught and passed to wxApp::OnFatalException.

By default, i.e. before this function is called, they will be handled in the normal way which usually just means that the application will be terminated. Calling wxHandleFatalExceptions() with *doIt* equal to false will restore this default behaviour.

Notice that this function is only available if `wxUSE_ON_FATAL_EXCEPTION` is 1 and under Windows platform this requires a compiler with support for SEH (structured exception handling) which currently means only Microsoft Visual C++ or a recent Borland C++ version.

Include file:

```
#include <wx/app.h>
```

void wxInitAllImageHandlers ( )

Initializes all available image handlers.

This function calls wxImage::AddHandler() for all the available image handlers (see Available image handlers for the full list). Calling it is the simplest way to initialize wxImage but it creates and registers even the handlers your program may not use. If you want to avoid the overhead of doing this you need to call wxImage::AddHandler() manually just for the handlers that you do want to use.

See also

wxImage, wxImageHandler

Include file:

```
#include <wx/image.h>
```

bool wxInitialize (int *argc* = 0, wxChar ** *argv* = NULL)

Initialize the library (may be called as many times as needed, but each call to wxInitialize() must be matched by wxUninitialize()).

With this function you may avoid wxDECLARE_APP() and wxIMPLEMENT_APP() macros and use wxInitialize() and wxUninitialize() dynamically in the program startup and termination.

Include file:

```
#include <wx/init.h>
```

bool wxInitialize ( )

This function is used in wxBase only and only if you don't create wxApp object at all.

In this case you must call it from your `main()` function before calling any other wxWidgets functions.

If the function returns false the initialization could not be performed, in this case the library cannot be used and wxUninitialize() shouldn't be called neither.

This function may be called several times but wxUninitialize() must be called for each successful call to this function.

Include file:

```
#include <wx/app.h>
```

bool wxSafeYield (wxWindow * *win* = NULL, bool *onlyIfNeeded* = false)

Calls wxApp::SafeYield.

Include file:

```
#include <wx/app.h>
```

void wxUninitialize ( )

This function is for use in console (wxBase) programs only.

Clean up; the library can't be used any more after the last call to wxUninitialize().

It must be called once for each previous successful call to wxInitialize().

Include file:

```
#include <wx/app.h>
```

See wxInitialize() for more info.

Include file:

```
#include <wx/init.h>
```

void wxWakeUpIdle ( )

This function wakes up the (internal and platform dependent) idle system, i.e.

it will force the system to send an idle event even if the system currently *is* idle and thus would not send any idle event until after some other event would get sent. This is also useful for sending events between two threads and is used by the corresponding functions wxPostEvent() and wxEvtHandler::AddPendingEvent().

Include file:

```
#include <wx/app.h>
```

bool wxYield ( )

Calls wxAppConsole::Yield.

Deprecated This function is kept only for backwards compatibility. Please use the wxAppConsole::Yield method instead in any new code.

Include file:

```
#include <wx/app.h>
```

20.2 Application and Process Management

20.2.1 Detailed Description

The classes in this section represent the application (see wxApp) or parts of it (e.g. wxEventLoopBase, wxModule). They can be used for initialization/shutdown of the application itself.

Related macros/global-functions group: Application Initialization and Termination.

Classes

- class wxAppConsole

  This class is essential for writing console-only or hybrid apps without having to define `wxUSE_GUI=0`.
- class wxApp

  The wxApp class represents the application itself when `wxUSE_GUI=1`.
- class wxCmdLineParser

  wxCmdLineParser is a class for parsing the command line.
- class wxDynamicLibraryDetails

  This class is used for the objects returned by the wxDynamicLibrary::ListLoaded() method and contains the information about a single module loaded into the address space of the current process.
- class wxDynamicLibrary

  wxDynamicLibrary is a class representing dynamically loadable library (Windows DLL, shared library under Unix etc).
- class wxEventLoopBase

  Base class for all event loop implementations.
- class wxEventLoopActivator

  Makes an event loop temporarily active.
- class wxGUIEventLoop

  A generic implementation of the GUI event loop.
- class wxInitializer

  Create an object of this class on the stack to initialize/cleanup the library automatically.
- class wxModule

  The module system is a very simple mechanism to allow applications (and parts of wxWidgets itself) to define initialization and cleanup functions that are automatically called on wxWidgets startup and exit.
- class wxProcess

  The objects of this class are used in conjunction with the wxExecute() function.
- class wxSingleInstanceChecker

  wxSingleInstanceChecker class allows to check that only a single instance of a program is running.

20.3 Application and System configuration

20.3.1 Detailed Description

The classes in this section are used to handle application-wide settings and system-wide settings.

Classes

- class wxAppTraits

  The wxAppTraits class defines various configurable aspects of a wxApp.
- class wxConfigBase

  wxConfigBase defines the basic interface of all config classes.
- class wxConfigPathChanger

  A handy little class which changes the current path in a wxConfig object and restores it in dtor.
- class wxDisplay

  Determines the sizes and locations of displays connected to the system.
- class wxFileConfig

  wxFileConfig implements wxConfigBase interface for storing and retrieving configuration information using plain text files.
- class wxFontMapper

  wxFontMapper manages user-definable correspondence between logical font names and the fonts present on the machine.
- class wxLocale

  wxLocale class encapsulates all language-dependent settings and is a generalization of the C locale concept.
- class wxMimeTypesManager

  This class allows the application to retrieve information about all known MIME types from a system-specific location and the filename extensions to the MIME types and vice versa.
- class wxRegConfig

  wxRegConfig implements the wxConfigBase interface for storing and retrieving configuration information using Windows registry.
- class wxRegKey

  wxRegKey is a class representing the Windows registry (it is only available under Windows).
- class wxPlatformInfo

  This class holds information about the operating system, the toolkit and the basic architecture of the machine where the application is currently running.
- class wxSystemSettings

  wxSystemSettings allows the application to ask for details about the system.
- class wxSystemOptions

  wxSystemOptions stores option/value pairs that wxWidgets itself or applications can use to alter behaviour at runtime.
- struct wxVideoMode

  Determines the sizes and locations of displays connected to the system.
- class wxIdManager

  wxIdManager is responsible for allocating and releasing window IDs.
- class wxXLocale

  This class represents a locale object used by so-called xlocale API.

20.4 Archive support

20.4.1 Detailed Description

Classes for managing (eventually compressed) archives.

Classes

- class wxArchiveInputStream

  This is an abstract base class which serves as a common interface to archive input streams such as wxZipInputStream.
- class wxArchiveOutputStream

  This is an abstract base class which serves as a common interface to archive output streams such as wxZipOutputStream.
- class wxArchiveEntry

  This is an abstract base class which serves as a common interface to archive entry classes such as wxZipEntry.
- class wxArchiveClassFactory

  Allows the creation of streams to handle archive formats such as zip and tar.
- class wxArchiveNotifier

  If you need to know when a wxArchiveInputStream updates a wxArchiveEntry object, you can create a notifier by deriving from this abstract base class, overriding wxArchiveNotifier::OnEntryUpdated.
- class wxArchiveIterator

  An input iterator template class that can be used to transfer an archive's catalogue to a container.
- class wxTarInputStream

  Input stream for reading tar files.
- class wxTarClassFactory

  Class factory for the tar archive format.
- class wxTarEntry

  Holds the meta-data for an entry in a tar.
- class wxZipNotifier

  If you need to know when a wxZipInputStream updates a wxZipEntry, you can create a notifier by deriving from this abstract base class, overriding wxZipNotifier::OnEntryUpdated().
- class wxZipEntry

  Holds the meta-data for an entry in a zip.
- class wxZipInputStream

  Input stream for reading zip files.
- class wxZipClassFactory

  Class factory for the zip archive format.
- class wxZipOutputStream

  Output stream for writing zip files.
- class wxZlibOutputStream

  This stream compresses all data written to it.
- class wxZlibInputStream

  This filter stream decompresses a stream that is in zlib or gzip format.

20.5 Atomic Operations

20.5.1 Detailed Description

When using multi-threaded applications, it is often required to access or modify memory which is shared between threads.

Atomic integer and pointer operations are an efficient way to handle this issue (another, less efficient, way is to use a wxMutex or wxCriticalSection). A native implementation exists for Windows, Linux, Solaris and Mac OS X; for others, a wxCriticalSection is used to protect the data.

One particular application is reference counting (used by so-called smart pointers).

You should define your variable with the type wxAtomicInt in order to apply atomic operations to it.

Functions

- void wxAtomicInc (wxAtomicInt &value)

  This function increments value in an atomic manner.
- wxInt32 wxAtomicDec (wxAtomicInt &value)

  This function decrements value in an atomic manner.

20.5.2 Function Documentation

wxInt32 wxAtomicDec (wxAtomicInt & *value*)

This function decrements value in an atomic manner.

Returns 0 if value is 0 after decrement or any non-zero value (not necessarily equal to the value of the variable) otherwise.

See also

wxAtomicInc

Include file:

```
#include <wx/atomic.h>
```

void wxAtomicInc (wxAtomicInt & *value*)

This function increments *value* in an atomic manner.

Whenever possible wxWidgets provides an efficient, CPU-specific, implementation of this function. If such implementation is available, the symbol wxHAS_ATOMIC_OPS is defined. Otherwise this function still exists but is implemented in a generic way using a critical section which can be prohibitively expensive for use in performance-sensitive code.

Include file:

```
#include <wx/atomic.h>
```

20.6 Book Controls

20.6.1 Detailed Description

A book control contains pages of other controls.

Related overview: wxBookCtrl Overview

Classes

- class wxAuiNotebookEvent

  This class is used by the events generated by wxAuiNotebook.
- class wxBookCtrlBase

  A book control is a convenient way of displaying multiple pages of information, displayed one page at a time.
- class wxBookCtrlEvent

  This class represents the events generated by book controls (wxNotebook, wxListbook, wxChoicebook, wxTreebook, wxAuiNotebook).
- class wxChoicebook

  wxChoicebook is a class similar to wxNotebook, but uses a wxChoice control to show the labels instead of the tabs.
- class wxListbook

  wxListbook is a class similar to wxNotebook but which uses a wxListCtrl to show the labels instead of the tabs.
- class wxNotebook

  This class represents a notebook control, which manages multiple windows with associated tabs.
- class wxSimplebook

  wxSimplebook is a control showing exactly one of its several pages.
- class wxToolbook

  wxToolbook is a class similar to wxNotebook but which uses a wxToolBar to show the labels instead of the tabs.
- class wxTreebook

  This class is an extension of the wxNotebook class that allows a tree structured set of pages to be shown in a control.

20.7 Byte Order

20.7.1 Detailed Description

The endian-ness issues (that is the difference between big-endian and little-endian architectures) are important for the portable programs working with the external binary data (for example, data files or data coming from network) which is usually in some fixed, platform-independent format.

The macros are helpful for transforming the data to the correct format.

Macros

- #define wxINT32_SWAP_ALWAYS(wxInt32_value)

  This macro will swap the bytes of the value variable from little endian to big endian or vice versa unconditionally, i.e.
- #define wxUINT32_SWAP_ALWAYS(wxUint32_value)
- #define wxINT16_SWAP_ALWAYS(wxInt16_value)
- #define wxUINT16_SWAP_ALWAYS(wxUint16_value)
- #define wxINT32_SWAP_ON_BE(wxInt32_value)

  This macro will swap the bytes of the value variable from little endian to big endian or vice versa if the program is compiled on a big-endian architecture (such as Sun work stations).
- #define wxUINT32_SWAP_ON_BE(wxUint32_value)
- #define wxINT16_SWAP_ON_BE(wxInt16_value)
- #define wxUINT16_SWAP_ON_BE(wxUint16_value)
- #define wxINT32_SWAP_ON_LE(wxInt32_value)

  This macro will swap the bytes of the value variable from little endian to big endian or vice versa if the program is compiled on a little-endian architecture (such as Intel PCs).
- #define wxUINT32_SWAP_ON_LE(wxUint32_value)
- #define wxINT16_SWAP_ON_LE(wxInt16_value)
- #define wxUINT16_SWAP_ON_LE(wxUint16_value)
- #define wxFORCE_LINK_THIS_MODULE(moduleName)

  This macro can be used in conjunction with the wxFORCE_LINK_MODULE() macro to force the linker to include in its output a specific object file.
- #define wxFORCE_LINK_MODULE(moduleName)

  This macro can be used in conjunction with the wxFORCE_LINK_THIS_MODULE() macro to force the linker to include in its output a specific object file.

20.7.2 Macro Definition Documentation

#define wxFORCE_LINK_MODULE(*moduleName*)

This macro can be used in conjunction with the wxFORCE_LINK_THIS_MODULE() macro to force the linker to include in its output a specific object file.

In particular, you should use this macro in a source file which you know for sure is linked in the output (e.g. the source file containing the `main()` of your app). The `moduleName` is the name of the module you want to forcefully link (i.e. the name you used in the relative wxFORCE_LINK_THIS_MODULE() macro.

Include file:

```
#include <wx/link.h>
```

#define wxFORCE_LINK_THIS_MODULE(*moduleName*)

This macro can be used in conjunction with the wxFORCE_LINK_MODULE() macro to force the linker to include in its output a specific object file.

In particular, you should use this macro in the source file which you want to force for inclusion. The `moduleName` needs to be a name not already in use in other wxFORCE_LINK_THIS_MODULE() macros, but is not required to be e.g. the same name of the source file (even if it's a good choice).

Include file:

```
#include <wx/link.h>
```

#define wxINT16_SWAP_ALWAYS(*wxInt16_value*)

#define wxINT16_SWAP_ON_BE(*wxInt16_value*)

#define wxINT16_SWAP_ON_LE(*wxInt16_value*)

#define wxINT32_SWAP_ALWAYS(*wxInt32_value*)

This macro will swap the bytes of the *value* variable from little endian to big endian or vice versa unconditionally, i.e.

independently of the current platform.

Include file:

```
#include <wx/defs.h>
```

#define wxINT32_SWAP_ON_BE(*wxInt32_value*)

This macro will swap the bytes of the *value* variable from little endian to big endian or vice versa if the program is compiled on a big-endian architecture (such as Sun work stations).

If the program has been compiled on a little-endian architecture, the value will be unchanged.

Use these macros to read data from and write data to a file that stores data in little-endian (for example Intel i386) format.

Include file:

```
#include <wx/defs.h>
```

#define wxINT32_SWAP_ON_LE(*wxInt32_value*)

This macro will swap the bytes of the *value* variable from little endian to big endian or vice versa if the program is compiled on a little-endian architecture (such as Intel PCs).

If the program has been compiled on a big-endian architecture, the value will be unchanged.

Use these macros to read data from and write data to a file that stores data in big-endian format.

Include file:

```
#include <wx/defs.h>
```

#define wxUINT16_SWAP_ALWAYS(*wxUint16_value*)

#define wxUINT16_SWAP_ON_BE(*wxUint16_value*)

#define wxUINT16_SWAP_ON_LE(*wxUint16_value* **)**

#define wxUINT32_SWAP_ALWAYS(*wxUint32_value* **)**

#define wxUINT32_SWAP_ON_BE(*wxUint32_value* **)**

#define wxUINT32_SWAP_ON_LE(*wxUint32_value* **)**

20.8 Class List by Category

20.8.1 Detailed Description

This group contains all full class list groups.

The Overview of Available Classes provides a quick summary of these groups on one page.

Modules

- Application and Process Management

  The classes in this section represent the application (see wxApp) or parts of it (e.g.
- Application and System configuration

  The classes in this section are used to handle application-wide settings and system-wide settings.
- Archive support

  Classes for managing (eventually compressed) archives.
- Book Controls

  A book control contains pages of other controls.
- Clipboard and Drag & Drop

  Related Overviews: Drag and Drop Overview.
- Common Dialogs

  Common dialogs are ready-made dialog classes which are frequently used in an application.
- Containers

  These are classes, templates and class macros are used by wxWidgets.
- Controls

  Typically, these are small windows which provide interaction with the user.
- Data Structures

  These are the data structure classes provided by wxWidgets.
- Debugging

  wxWidgets supports some aspects of debugging an application through classes, functions and macros.
- Device Contexts

  Device contexts are surfaces that may be drawn on, and provide an abstraction that allows parameterisation of your drawing code by passing different device contexts.
- Document/View Framework

  wxWidgets supports a document/view framework which provides housekeeping for a document-centric application.
- Events

  An event object contains information about a specific event.
- File Handling

  wxWidgets has several small classes to work with disk files and directories.
- Graphics Device Interface (GDI)

  The following are classes related to GDI (Graphics Device Interface) access.
- Grid Related Classes

  Classes related to the wxGrid generic widget.
- HTML

  wxWidgets provides a set of classes to display text in HTML format.
- Help

  Classes for loading and displaying help manuals or help informations in general.
- Interprocess Communication

  wxWidgets provides simple interprocess communications facilities based on Windows DDE, but they are available on most platforms using TCP.
- Logging

wxWidgets provides several classes and functions for message logging.

- Managed Windows

  There are several types of window that are directly controlled by the window manager (such as MS Windows, or the Motif Window Manager).

- Menus

  Group of classes for handling menu bars and items.

- Miscellaneous

  Group of miscellaneous classes.

- Miscellaneous Windows

  The following are a variety of classes that are derived from wxWindow.

- Multimedia

  Classes for showing multimedia contents.

- Networking

  wxWidgets provides its own classes for socket based networking.

- OpenGL

  Classes interfacing wxWidgets with OpenGL (http://opengl.org/).

- Picker Controls

  A picker control is a control whose appearance and behaviour is highly platform-dependent.

- Printing Framework

  A printing and previewing framework is implemented to make it relatively straightforward to provide document printing facilities.

- Ribbon User Interface

  The wxRibbon library is a set of classes for writing a ribbon user interface.

- Rich Text

  wxWidgets provides a set of generic classes to edit and print simple rich text with character and paragraph formatting.

- Runtime Type Information (RTTI)

  wxWidgets supports runtime manipulation of class information, and dynamic creation of objects given class names.

- Scintilla Text Editor

  wxWidgets also provides a wrapper around the Scintilla text editor control, which is a control for plain-text editing with support for highlighting, smart indentation, etc.

- Smart Pointers

  wxWidgets provides a few smart pointer class templates.

- Streams

  wxWidgets has its own set of stream classes, as an alternative to often buggy standard stream libraries, and to provide enhanced functionality.

- Text Conversion

  These are the classes used for conversions between different text encodings.

- Threading

  wxWidgets provides a set of classes to make use of the native thread capabilities of the various platforms.

- Validators

  These are the window validators, used for filtering and validating user input.

- Virtual File System

  wxWidgets provides a set of classes that implement an extensible virtual file system, used internally by the HTML classes.

- WebView

  The wxWebView library is a set of classes for viewing complex web documents and for internet browsing.

- Window Docking (wxAUI)

  wxAUI is a set classes for writing a customizable application interface with built-in docking, floatable panes and a flexible MDI-like interface.

- Window Layout

  wxWidgets makes window layout and sizing easy and painless using a set of classes known as "sizers".

- XML

Group of classes loading and saving XML documents (http://www.w3.org/XML/).

- XML Based Resource System (XRC)

  Resources allow your application to create controls and other user interface elements from specifications stored in an XML format.

- wxDataViewCtrl Related Classes

  These are all classes used or provided for use with wxDataViewCtrl.

- wxPropertyGrid

  wxPropertyGrid is a specialized grid for editing properties (that is, name=value pairs).

20.9 Clipboard and Drag & Drop

20.9.1 Detailed Description

Related Overviews: Drag and Drop Overview.

Classes

- class wxClipboard

  A class for manipulating the clipboard.
- class wxDataFormat

  A wxDataFormat is an encapsulation of a platform-specific format handle which is used by the system for the clipboard and drag and drop operations.
- class wxDataObject

  A wxDataObject represents data that can be copied to or from the clipboard, or dragged and dropped.
- class wxCustomDataObject

  wxCustomDataObject is a specialization of wxDataObjectSimple for some application-specific data in arbitrary (either custom or one of the standard ones).
- class wxDataObjectComposite

  wxDataObjectComposite is the simplest wxDataObject derivation which may be used to support multiple formats.
- class wxDataObjectSimple

  This is the simplest possible implementation of the wxDataObject class.
- class wxBitmapDataObject

  wxBitmapDataObject is a specialization of wxDataObject for bitmap data.
- class wxURLDataObject

  wxURLDataObject is a wxDataObject containing an URL and can be used e.g.
- class wxTextDataObject

  wxTextDataObject is a specialization of wxDataObjectSimple for text data.
- class wxFileDataObject

  wxFileDataObject is a specialization of wxDataObject for file names.
- class wxHTMLDataObject

  wxHTMLDataObject is used for working with HTML-formatted text.
- class wxDropTarget

  This class represents a target for a drag and drop operation.
- class wxDropSource

  This class represents a source for a drag and drop operation.
- class wxTextDropTarget

  A predefined drop target for dealing with text data.
- class wxFileDropTarget

  This is a drop target which accepts files (dragged from File Manager or Explorer).
- class wxDragImage

  This class is used when you wish to drag an object on the screen, and a simple cursor is not enough.

20.10 Common Dialogs

20.10.1 Detailed Description

Common dialogs are ready-made dialog classes which are frequently used in an application.

Related Overviews: Common Dialogs

Classes

- class wxAboutDialogInfo

  wxAboutDialogInfo contains information shown in the standard About dialog displayed by the wxAboutBox() function.
- class wxBusyInfo

  This class makes it easy to tell your user that the program is temporarily busy.
- class wxMultiChoiceDialog

  This class represents a dialog that shows a list of strings, and allows the user to select one or more.
- class wxSingleChoiceDialog

  This class represents a dialog that shows a list of strings, and allows the user to select one.
- class wxPrintDialogData

  This class holds information related to the visual characteristics of wxPrintDialog.
- class wxColourDialog

  This class represents the colour chooser dialog.
- class wxColourData

  This class holds a variety of information related to colour dialogs.
- class wxDialog

  A dialog box is a window with a title bar and sometimes a system menu, which can be moved around the screen.
- class wxDirDialog

  This class represents the directory chooser dialog.
- class wxFindReplaceData

  wxFindReplaceData holds the data for wxFindReplaceDialog.
- class wxFindReplaceDialog

  wxFindReplaceDialog is a standard modeless dialog which is used to allow the user to search for some text (and possibly replace it with something else).
- class wxFileDialog

  This class represents the file chooser dialog.
- class wxFontData

  This class holds a variety of information related to font dialogs.
- class wxFontDialog

  This class represents the font chooser dialog.
- class wxGenericAboutDialog

  This class defines a customizable About dialog.
- class wxMessageDialog

  This class represents a dialog that shows a single or multi-line message, with a choice of OK, Yes, No and Cancel buttons.
- class wxGenericProgressDialog

  This class represents a dialog that shows a short message and a progress bar.
- class wxRearrangeDialog

  A dialog allowing the user to rearrange the specified items.
- class wxRichMessageDialog

  Extension of wxMessageDialog with additional functionality.
- class wxSymbolPickerDialog

wxSymbolPickerDialog presents the user with a choice of fonts and a grid of available characters.

- class wxPasswordEntryDialog

  This class represents a dialog that requests a one-line password string from the user.

- class wxTextEntryDialog

  This class represents a dialog that requests a one-line text string from the user.

- class wxWizard

  wxWizard is the central class for implementing 'wizard-like' dialogs.

20.11 Containers

20.11.1 Detailed Description

These are classes, templates and class macros are used by wxWidgets.

Most of these classes provide a subset or almost complete STL API.

Related Overviews: Container Classes

Classes

- class wxArray< T >

  This section describes the so called "dynamic arrays".
- class wxList< T >

  The wxList<T> class provides linked list functionality.
- class wxStack< T >

  wxStack<T> is similar to `std::stack` and can be used exactly like it.
- class wxVector< T >

  wxVector<T> is a template class which implements most of the `std::vector` class and can be used like it.
- class wxArrayString

  wxArrayString is an efficient container for storing wxString objects.
- class wxSortedArrayString

  wxSortedArrayString is an efficient container for storing wxString objects which always keeps the string in alphabetical order.
- class wxClientDataContainer

  This class is a mixin that provides storage and management of "client data".
- class wxClientData

  All classes deriving from wxEvtHandler (such as all controls and wxApp) can hold arbitrary data which is here referred to as "client data".
- class wxStringClientData

  Predefined client data class for holding a string.
- class wxHashTable
- class wxHashMap

  This is a simple, type-safe, and reasonably efficient hash map class, whose interface is a subset of the interface of STL containers.
- class wxHashSet

  This is a simple, type-safe, and reasonably efficient hash set class, whose interface is a subset of the interface of STL containers.
- class wxTreeItemData

  wxTreeItemData is some (arbitrary) user class associated with some item.

20.12 Controls

20.12.1 Detailed Description

Typically, these are small windows which provide interaction with the user.

Controls that are not static can have wxValidator associated with them.

Classes

- class wxAnimationCtrl

  This is a static control which displays an animation.
- class wxBitmapButton

  A bitmap button is a control that contains a bitmap.
- class wxBitmapComboBox

  A combobox that displays bitmap in front of the list items.
- class wxButton

  A button is a control that contains a text string, and is one of the most common elements of a GUI.
- class wxCalendarCtrl

  The calendar control allows the user to pick a date.
- class wxCheckBox

  A checkbox is a labelled box which by default is either on (checkmark is visible) or off (no checkmark).
- class wxCheckListBox

  A wxCheckListBox is like a wxListBox, but allows items to be checked or unchecked.
- class wxChoice

  A choice item is used to select one of a list of strings.
- class wxCollapsiblePane

  A collapsible pane is a container with an embedded button-like control which can be used by the user to collapse or expand the pane's contents.
- class wxComboPopup

  In order to use a custom popup with wxComboCtrl, an interface class must be derived from wxComboPopup.
- class wxComboCtrl

  A combo control is a generic combobox that allows totally custom popup.
- class wxComboBox

  A combobox is like a combination of an edit control and a listbox.
- class wxCommandLinkButton

  Objects of this class are similar in appearance to the normal wxButtons but are similar to the links in a web page in functionality.
- class wxControl

  This is the base class for a control or "widget".
- class wxItemContainerImmutable

  wxItemContainer defines an interface which is implemented by all controls which have string subitems each of which may be selected.
- class wxItemContainer

  This class is an abstract base class for some wxWidgets controls which contain several items such as wxListBox, wxCheckListBox, wxComboBox or wxChoice.
- class wxControlWithItems

  This is convenience class that derives from both wxControl and wxItemContainer.
- class wxDataViewCtrl

  wxDataViewCtrl is a control to display data either in a tree like fashion or in a tabular form or both.
- class wxDataViewListCtrl

This class is a wxDataViewCtrl which internally uses a wxDataViewListStore and forwards most of its API to that class.

- class wxDataViewTreeCtrl

  This class is a wxDataViewCtrl which internally uses a wxDataViewTreeStore and forwards most of its API to that class.
- class wxGenericDirCtrl

  This control can be used to place a directory listing (with optional files) on an arbitrary window.
- class wxEditableListBox

  An editable listbox is composite control that lets the user easily enter, delete and reorder a list of strings.
- class wxFileCtrl

  This control allows the user to select a file.
- class wxGauge

  A gauge is a horizontal or vertical bar which shows a quantity (often time).
- class wxHeaderColumn

  Represents a column header in controls displaying tabular data such as wxDataViewCtrl or wxGrid.
- class wxSettableHeaderColumn

  Adds methods to set the column attributes to wxHeaderColumn.
- class wxHeaderColumnSimple

  Simple container for the information about the column.
- class wxHeaderCtrl

  wxHeaderCtrl is the control containing the column headings which is usually used for display of tabular data.
- class wxHeaderCtrlSimple

  wxHeaderCtrlSimple is a concrete header control which can be used directly, without inheriting from it as you need to do when using wxHeaderCtrl itself.
- class wxHtmlListBox

  wxHtmlListBox is an implementation of wxVListBox which shows HTML content in the listbox rows.
- class wxSimpleHtmlListBox

  wxSimpleHtmlListBox is an implementation of wxHtmlListBox which shows HTML content in the listbox rows.
- class wxHyperlinkCtrl

  This class shows a static text element which links to an URL.
- class wxListBox

  A listbox is used to select one or more of a list of strings.
- class wxListCtrl

  A list control presents lists in a number of formats: list view, report view, icon view and small icon view.
- class wxListView

  This class currently simply presents a simpler to use interface for the wxListCtrl – it can be thought of as a façade for that complicated class.
- class wxActiveXContainer

  wxActiveXContainer is a host for an ActiveX control on Windows (and as such is a platform-specific class).
- class wxOwnerDrawnComboBox

  wxOwnerDrawnComboBox is a combobox with owner-drawn list items.
- class wxRadioBox

  A radio box item is used to select one of number of mutually exclusive choices.
- class wxRadioButton

  A radio button item is a button which usually denotes one of several mutually exclusive options.
- class wxRearrangeList

  A listbox-like control allowing the user to rearrange the items and to enable or disable them.
- class wxRearrangeCtrl

  A composite control containing a wxRearrangeList and the buttons allowing to move the items in it.
- class wxScrollBar

  A wxScrollBar is a control that represents a horizontal or vertical scrollbar.

- class wxSlider

  A slider is a control with a handle which can be pulled back and forth to change the value.
- class wxSpinButton

  A wxSpinButton has two small up and down (or left and right) arrow buttons.
- class wxSpinCtrl

  wxSpinCtrl combines wxTextCtrl and wxSpinButton in one control.
- class wxSpinCtrlDouble

  wxSpinCtrlDouble combines wxTextCtrl and wxSpinButton in one control and displays a real number.
- class wxSearchCtrl

  A search control is a composite control with a search button, a text control, and a cancel button.
- class wxStaticBitmap

  A static bitmap control displays a bitmap.
- class wxStaticBox

  A static box is a rectangle drawn around other windows to denote a logical grouping of items.
- class wxStaticLine

  A static line is just a line which may be used in a dialog to separate the groups of controls.
- class wxStaticText

  A static text control displays one or more lines of read-only text.
- class wxTextCtrl

  A text control allows text to be displayed and edited.
- class wxTextEntry

  Common base class for single line text entry fields.
- class wxToggleButton

  wxToggleButton is a button that stays pressed when clicked by the user.
- class wxBitmapToggleButton

  wxBitmapToggleButton is a wxToggleButton that contains a bitmap instead of text.
- class wxTreeCtrl

  A tree control presents information as a hierarchy, with items that may be expanded to show further items.
- class wxTreeListItem

  Unique identifier of an item in wxTreeListCtrl.
- class wxTreeListItemComparator

  Class defining sort order for the items in wxTreeListCtrl.
- class wxTreeListCtrl

  A control combining wxTreeCtrl and wxListCtrl features.
- class wxVListBox

  wxVListBox is a wxListBox-like control with the following two main differences from a regular wxListBox: it can have an arbitrarily huge number of items because it doesn't store them itself but uses the OnDrawItem() callback to draw them (so it is a virtual listbox) and its items can have variable height as determined by OnMeasureItem() (so it is also a listbox with the lines of variable height).
- class wxWebView

  This control may be used to render web (HTML / CSS / javascript) documents.

20.13 Data Structures

20.13.1 Detailed Description

These are the data structure classes provided by wxWidgets.

Some of them are used to store generic data (e.g. wxPoint, wxSize, etc), others are mainly helpers of other classes (e.g. wxListItem, wxCalendarDateAttr, wxFindReplaceDialogData, etc).

Classes

- class wxScopedCharTypeBuffer< T >

  wxScopedCharTypeBuffer<T> is a template class for storing characters.
- class wxCharTypeBuffer< T >

  wxCharTypeBuffer<T> is a template class for storing characters.
- class wxCharBuffer

  This is a specialization of wxCharTypeBuffer<T> for `char` type.
- class wxWCharBuffer

  This is a specialization of wxCharTypeBuffer<T> for `wchar_t` type.
- class wxNode< T >

  wxNode<T> is the node structure used in linked lists (see wxList) and derived classes.
- class wxAboutDialogInfo

  wxAboutDialogInfo contains information shown in the standard About dialog displayed by the wxAboutBox() function.
- class wxAcceleratorEntry

  An object used by an application wishing to create an accelerator table (see wxAcceleratorTable).
- class wxAcceleratorTable

  An accelerator table allows the application to specify a table of keyboard shortcuts for menu or button commands.
- class wxAny

  The wxAny class represents a container for any type.
- class wxAnyValueType

  wxAnyValueType is base class for value type functionality for C++ data types used with wxAny.
- class wxMemoryBuffer

  *A **wxMemoryBuffer** is a useful data structure for storing arbitrary sized blocks of memory.*
- class wxCalendarDateAttr

  wxCalendarDateAttr is a custom attributes for a calendar date.
- class wxPageSetupDialogData

  This class holds a variety of information related to wxPageSetupDialog.
- class wxPrintData

  This class holds a variety of information related to printers and printer device contexts.
- class wxPrintDialogData

  This class holds information related to the visual characteristics of wxPrintDialog.
- class wxColourData

  This class holds a variety of information related to colour dialogs.
- class wxConvAuto

  This class implements a Unicode to/from multibyte converter capable of automatically recognizing the encoding of the multibyte text on input.
- class wxDateTime

  wxDateTime class represents an absolute moment in time.
- class wxDateTimeWorkDays
- class wxDateSpan

This class is a "logical time span" and is useful for implementing program logic for such things as "add one month to the date" which, in general, doesn't mean to add 60∗60∗24∗31 seconds to it, but to take the same date the next month (to understand that this is indeed different consider adding one month to Feb, 15 – we want to get Mar, 15, of course).

- class wxTimeSpan

  wxTimeSpan class represents a time interval.
- class wxDateTimeHolidayAuthority
- class wxFindReplaceData

  wxFindReplaceData holds the data for wxFindReplaceDialog.
- class wxFontData

  This class holds a variety of information related to font dialogs.
- class wxRealPoint

  A wxRealPoint is a useful data structure for graphics operations.
- class wxRect

  A class for manipulating rectangles.
- class wxPoint

  A wxPoint is a useful data structure for graphics operations.
- class wxSize

  A wxSize is a useful data structure for graphics operations.
- class wxListItemAttr

  Represents the attributes (color, font, ...) of a wxListCtrl's wxListItem.
- class wxListItem

  This class stores information about a wxListCtrl item or column.
- class wxLongLong

  This class represents a signed 64 bit long number.
- class wxULongLong

  This class represents an unsigned 64 bit long number.
- class wxFileType

  This class holds information about a given file type.
- class wxVariantDataCurrency

  This class represents a thin wrapper for Microsoft Windows CURRENCY type.
- class wxVariantDataErrorCode

  This class represents a thin wrapper for Microsoft Windows SCODE type (which is the same as HRESULT).
- class wxVariantDataSafeArray

  This class represents a thin wrapper for Microsoft Windows SAFEARRAY type.
- class wxAutomationObject

  *The **wxAutomationObject** class represents an OLE automation object containing a single data member, an IDispatch pointer.*
- class wxPosition

  This class represents the position of an item in any kind of grid of rows and columns such as wxGridBagSizer, or wxHVScrolledWindow.
- class wxRegEx

  wxRegEx represents a regular expression.
- class wxRegion

  A wxRegion represents a simple or complex region on a device context or window.
- class wxStatusBarPane

  A status bar pane data container used by wxStatusBar.
- class wxString

  String class for passing textual data to or receiving it from wxWidgets.
- class wxStringBufferLength

  This tiny class allows you to conveniently access the wxString internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later, and allows the user to set the internal length of the string.

- class wxStringBuffer

  This tiny class allows you to conveniently access the wxString internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later.
- class wxStringTokenizer

  wxStringTokenizer helps you to break a string up into a number of tokens.
- class wxTreeItemId

  An opaque reference to a tree item.
- class wxUniChar

  This class represents a single Unicode character.
- class wxUniCharRef

  Writeable reference to a character in wxString.
- class wxUString

  wxUString is a class representing a Unicode character string where each character is stored using a 32-bit value.
- class wxVariant

  The wxVariant class represents a container for any type.
- class wxVariantData

  The wxVariantData class is used to implement a new type for wxVariant.
- class wxVersionInfo

  wxVersionInfo contains version information.

20.14 Debugging

20.14.1 Detailed Description

wxWidgets supports some aspects of debugging an application through classes, functions and macros.

Related Overviews: Debugging

Related macros/global-functions group: Debugging macros

Classes

- class wxDebugReportPreview

  This class presents the debug report to the user and allows him to veto report entirely or remove some parts of it.
- class wxDebugReportCompress

  wxDebugReportCompress is a wxDebugReport which compresses all the files in this debug report into a single ZIP file in its wxDebugReport::Process() function.
- class wxDebugReport

  wxDebugReport is used to generate a debug report, containing information about the program current state.
- class wxDebugReportPreviewStd

  wxDebugReportPreviewStd is a standard debug report preview window.
- class wxDebugReportUpload

  This class is used to upload a compressed file using HTTP POST request.
- class wxDebugContext

  A class for performing various debugging and memory tracing operations.
- class wxStackWalker

  wxStackWalker allows an application to enumerate, or walk, the stack frames (the function callstack).
- class wxStackFrame

  wxStackFrame represents a single stack frame, or a single function in the call stack, and is used exclusively together with wxStackWalker, see there for a more detailed discussion.

20.15 Debugging macros

20.15.1 Detailed Description

Useful macros and functions for error checking and defensive programming.

Starting with wxWidgets 2.9.1, debugging support in wxWidgets is always compiled in by default, you need to explicitly define wxDEBUG_LEVEL as 0 to disable it completely. However, by default debugging macros are dormant in the release builds, i.e. when the main program is compiled with the standard `NDEBUG` symbol being defined. You may explicitly activate the debugging checks in the release build by calling wxSetAssertHandler() with a custom function if needed.

When debugging support is active, failure of both wxASSERT() and wxCHECK() macros conditions result in a debug alert. When debugging support is inactive or turned off entirely at compilation time, wxASSERT() and wxFAIL() macros don't do anything while wxCHECK() still checks its condition and returns if it fails, even if no alerts are shown to the user.

Finally, the compile time assertions don't happen during the run-time but result in the compilation error messages if the condition they check fail. They are always enabled and are not affected by wxDEBUG_LEVEL.

Related class group: Debugging.

Macros

- #define wxDISABLE_DEBUG_SUPPORT()

  Use this macro to disable all debugging code in release build when not using wxIMPLEMENT_APP().
- #define wxDEBUG_LEVEL

  Preprocessor symbol defining the level of debug support available.
- #define __WXDEBUG__

  Compatibility macro indicating presence of debug support.
- #define wxASSERT(condition)

  Assert macro.
- #define wxASSERT_LEVEL_2(condition)

  Assert macro for expensive run-time checks.
- #define wxASSERT_LEVEL_2_MSG(condition, msg)

  Assert macro with a custom message for expensive run-time checks.
- #define wxASSERT_MIN_BITSIZE(type, size)

  This macro results in a compile time assertion failure if the size of the given `type` is less than `size` bits.
- #define wxASSERT_MSG(condition, message)

  Assert macro with message.
- #define wxASSERT_MSG_AT(condition, message, file, line, func)

  Assert macro pretending to assert at the specified location.
- #define wxCHECK(condition, retValue)

  Checks that the condition is true, returns with the given return value if not (stops execution in debug mode).
- #define wxCHECK_MSG(condition, retValue, message)

  Checks that the condition is true, returns with the given return value if not (stops execution in debug mode).
- #define wxCHECK_RET(condition, message)

  Checks that the condition is true, and returns if not (stops execution with the given error message in debug mode).
- #define wxCHECK2(condition, operation)

  Checks that the condition is true, and if not, it will wxFAIL() and execute the given `operation` if it is not.
- #define wxCHECK2_MSG(condition, operation, message)

  This is the same as wxCHECK2(), but wxFAIL_MSG() with the specified `message` is called instead of wxFAIL() if the `condition` is false.
- #define wxCOMPILE_TIME_ASSERT(condition, message)

  Using wxCOMPILE_TIME_ASSERT() results in a compilation error if the specified `condition` is false.

- #define wxCOMPILE_TIME_ASSERT2(condition, message, name)

  This macro is identical to wxCOMPILE_TIME_ASSERT() except that it allows you to specify a unique `name` *for the struct internally defined by this macro to avoid getting the compilation errors described for wxCOMPILE_TIME_ASSERT().*
- #define wxDISABLE_ASSERTS_IN_RELEASE_BUILD() wxDisableAsserts()

  Use this macro to disable asserts in release build when not using wxIMPLEMENT_APP().
- #define wxFAIL

  Will always generate an assert error if this code is reached (in debug mode).
- #define wxFAIL_MSG(message)

  Will always generate an assert error with specified message if this code is reached (in debug mode).
- #define wxFAIL_MSG_AT(message, file, line, func)

  Assert failure macro pretending to assert at the specified location.
- #define wxDISABLE_DEBUG_LOGGING_IN_RELEASE_BUILD()

  Use this macro to disable logging at debug and trace levels in release build when not using wxIMPLEMENT_APP().
- #define WXDEBUG_NEW(arg)

  This is defined in debug mode to be call the redefined new operator with filename and line number arguments.

Typedefs

- typedef void(∗ wxAssertHandler_t)(const wxString &file, int line, const wxString &func, const wxString &cond, const wxString &msg)

  Type for the function called in case of assert failure.

Functions

- void wxAbort ()

  Exits the program immediately.
- void wxDisableAsserts ()

  Disable the condition checks in the assertions.
- bool wxIsDebuggerRunning ()

  Returns true if the program is running under debugger, false otherwise.
- wxAssertHandler_t wxSetAssertHandler (wxAssertHandler_t handler)

  Sets the function to be called in case of assertion failure.
- void wxSetDefaultAssertHandler ()

  Reset the assert handler to default function which shows a message box when an assert happens.
- void wxTrap ()

  Generate a debugger exception meaning that the control is passed to the debugger if one is attached to the process.

20.15.2 Macro Definition Documentation

#define __WXDEBUG__

Compatibility macro indicating presence of debug support.

This symbol is defined if wxDEBUG_LEVEL is greater than 0 and undefined otherwise.

Include file:

```
#include <wx/debug.h>
```

#define wxASSERT(*condition*)

Assert macro.

An error message will be generated if the condition is false in debug mode, but nothing will be done in the release build.

Please note that the condition in wxASSERT() should have no side effects because it will not be executed in release mode at all.

This macro should be used to catch (in debug builds) logical errors done by the programmer.

See also

wxASSERT_MSG(), wxCOMPILE_TIME_ASSERT()

Include file:

```
#include <wx/debug.h>
```

#define wxASSERT_LEVEL_2(*condition*)

Assert macro for expensive run-time checks.

This macro does nothing unless wxDEBUG_LEVEL is 2 or more and is meant to be used for the assertions with noticeable performance impact and which, hence, should be disabled during run-time.

If wxDEBUG_LEVEL is 2 or more, it becomes the same as wxASSERT().

Include file:

```
#include <wx/debug.h>
```

#define wxASSERT_LEVEL_2_MSG(*condition, msg*)

Assert macro with a custom message for expensive run-time checks.

If wxDEBUG_LEVEL is 2 or more, this is the same as wxASSERT_MSG(), otherwise it doesn't do anything at all.

See also

wxASSERT_LEVEL_2()

Include file:

```
#include <wx/debug.h>
```

#define wxASSERT_MIN_BITSIZE(*type, size*)

This macro results in a compile time assertion failure if the size of the given `type` is less than `size` bits.

This macro should be used to catch (in debug builds) logical errors done by the programmer.

You may use it like this, for example:

```
1 // we rely on the int being able to hold values up to 2^32
2 wxASSERT_MIN_BITSIZE(int, 32);
3
4 // can't work with the platforms using UTF-8 for wchar_t
5 wxASSERT_MIN_BITSIZE(wchar_t, 16);
```

Include file:

```
#include <wx/debug.h>
```

#define wxASSERT_MSG(*condition, message*)

Assert macro with message.

An error message will be generated if the condition is false.

This macro should be used to catch (in debug builds) logical errors done by the programmer.

See also

wxASSERT(), wxCOMPILE_TIME_ASSERT()

Include file:

```
#include <wx/debug.h>
```

#define wxASSERT_MSG_AT(*condition, message, file, line, func*)

Assert macro pretending to assert at the specified location.

This macro is the same as wxASSERT_MSG(), but the assert message will use the specified source file, line number and function name instead of the values corresponding to the current location as done by wxASSERT_MSG() by default.

It is mostly useful for asserting inside functions called from macros, as by passing the usual **__FILE__**, **__LINE__** and **__FUNCTION__** values to a function, it's possible to pretend that the assert happens at the location of the macro in the source code (which can be useful) instead of inside the function itself (which is never useful as these values are always the same for the given assertion).

Since

3.1.0

Include file:

```
#include <wx/debug.h>
```

#define wxCHECK(*condition, retValue*)

Checks that the condition is true, returns with the given return value if not (stops execution in debug mode).

This check is done even in release mode.

This macro should be used to catch (both in debug and release builds) logical errors done by the programmer.

Include file:

```
#include <wx/debug.h>
```

#define wxCHECK2(*condition, operation*)

Checks that the condition is true, and if not, it will wxFAIL() and execute the given `operation` if it is not.

This is a generalisation of wxCHECK() and may be used when something else than just returning from the function must be done when the `condition` is false. This check is done even in release mode.

This macro should be used to catch (both in debug and release builds) logical errors done by the programmer.

Include file:

```
#include <wx/debug.h>
```

#define wxCHECK2_MSG(*condition, operation, message*)

This is the same as wxCHECK2(), but wxFAIL_MSG() with the specified `message` is called instead of wxFAIL() if the `condition` is false.

This macro should be used to catch (both in debug and release builds) logical errors done by the programmer.

Include file:

```
#include <wx/debug.h>
```

#define wxCHECK_MSG(*condition, retValue, message*)

Checks that the condition is true, returns with the given return value if not (stops execution in debug mode).

This check is done even in release mode.

This macro may be only used in non-void functions, see also wxCHECK_RET().

This macro should be used to catch (both in debug and release builds) logical errors done by the programmer.

Include file:

```
#include <wx/debug.h>
```

#define wxCHECK_RET(*condition, message*)

Checks that the condition is true, and returns if not (stops execution with the given error message in debug mode).

This check is done even in release mode.

This macro should be used in void functions instead of wxCHECK_MSG().

This macro should be used to catch (both in debug and release builds) logical errors done by the programmer.

Include file:

```
#include <wx/debug.h>
```

#define wxCOMPILE_TIME_ASSERT(*condition, message*)

Using wxCOMPILE_TIME_ASSERT() results in a compilation error if the specified `condition` is false.

The compiler error message should include the `message` identifier - please note that it must be a valid C++ identifier and not a string unlike in the other cases.

This macro is mostly useful for testing the expressions involving the `sizeof` operator as they can't be tested by the preprocessor but it is sometimes desirable to test them at the compile time.

Note that this macro internally declares a struct whose name it tries to make unique by using the **`LINE`** in it but it may still not work if you use it on the same line in two different source files. In this case you may either change the line in which either of them appears on or use the wxCOMPILE_TIME_ASSERT2() macro.

Also note that Microsoft Visual C++ has a bug which results in compiler errors if you use this macro with 'Program Database For Edit And Continue' (`/ZI`) option, so you shouldn't use it ('Program Database' (`/Zi`) is ok though) for the code making use of this macro.

This macro should be used to catch misconfigurations at compile-time.

See also

wxASSERT_MSG(), wxASSERT_MIN_BITSIZE()

Include file:

```
#include <wx/debug.h>
```

#define wxCOMPILE_TIME_ASSERT2(*condition, message, name*)

This macro is identical to wxCOMPILE_TIME_ASSERT() except that it allows you to specify a unique `name` for the struct internally defined by this macro to avoid getting the compilation errors described for wxCOMPILE_TIME_ASSERT().

This macro should be used to catch misconfigurations at compile-time.

Include file:

```
#include <wx/debug.h>
```

#define wxDEBUG_LEVEL

Preprocessor symbol defining the level of debug support available.

This symbol is defined to 1 by default meaning that asserts are compiled in (although they may be disabled by a call to wxDisableAsserts()). You may predefine it as 0 prior to including any wxWidgets headers to omit the calls to wxASSERT() and related macros entirely in your own code and you may also predefine it as 0 when building wxWidgets to also avoid including any asserts in wxWidgets itself.

Alternatively, you may predefine it as 2 to include wxASSERT_LEVEL_2() and similar macros which are used for asserts which have non-trivial run-time costs and so are disabled by default.

Since

2.9.1

Include file:

```
#include <wx/debug.h>
```

#define WXDEBUG_NEW(*arg*)

This is defined in debug mode to be call the redefined new operator with filename and line number arguments.

The definition is:

```
1 #define WXDEBUG_NEW new(__FILE__,__LINE__)
```

In non-debug mode, this is defined as the normal new operator.

Include file:

```
#include <wx/object.h>
```

#define wxDISABLE_ASSERTS_IN_RELEASE_BUILD( ) wxDisableAsserts()

Use this macro to disable asserts in release build when not using wxIMPLEMENT_APP().

By default, assert message boxes are suppressed in release build by wxIMPLEMENT_APP() which uses this macro. If you don't use wxIMPLEMENT_APP() because your application initializes wxWidgets directly (e.g. calls wxEntry() or wxEntryStart() itself) but still want to suppress assert notifications in release build you need to use this macro directly.

See also

wxDISABLE_DEBUG_SUPPORT()

Since

2.9.1

Include file:

```
#include <wx/debug.h>
```

#define wxDISABLE_DEBUG_LOGGING_IN_RELEASE_BUILD( )

Use this macro to disable logging at debug and trace levels in release build when not using wxIMPLEMENT_APP().

See also

wxDISABLE_DEBUG_SUPPORT(), wxDISABLE_ASSERTS_IN_RELEASE_BUILD(), Debugging

Since

2.9.1

Include file:

```
#include <wx/log.h>
```

#define wxDISABLE_DEBUG_SUPPORT( )

Value:

```
wxDISABLE_ASSERTS_IN_RELEASE_BUILD(); \
    wxDISABLE_DEBUG_LOGGING_IN_RELEASE_BUILD()
```

Use this macro to disable all debugging code in release build when not using wxIMPLEMENT_APP().

Currently this macro disables assert checking and debug and trace level logging messages in release build (i.e. when `NDEBUG` is defined). It is used by wxIMPLEMENT_APP() macro so you only need to use it explicitly if you don't use this macro but initialize wxWidgets directly (e.g. calls wxEntry() or wxEntryStart() itself).

If you do not want to disable debugging code even in release build of your application, you can use wxSetDefaultAssertHandler() and wxLog::SetLogLevel() with `wxLOG_Max` parameter to enable assertions and debug logging respectively.

See also

wxDISABLE_ASSERTS_IN_RELEASE_BUILD(), wxDISABLE_DEBUG_LOGGING_IN_RELEASE_BUILD(), Debugging

Since

2.9.1

Include file:

```
#include <wx/app.h>
```

#define wxFAIL

Will always generate an assert error if this code is reached (in debug mode).

Note that you don't have to (and cannot) use brackets when invoking this macro:

```
1 if (...some condition...) {
2     wxFAIL;
3 }
```

This macro should be used to catch (in debug builds) logical errors done by the programmer.

See also

wxFAIL_MSG()

Include file:

```
#include <wx/debug.h>
```

#define wxFAIL_MSG(*message*)

Will always generate an assert error with specified message if this code is reached (in debug mode).

This macro is useful for marking "unreachable" code areas, for example it may be used in the "default:" branch of a switch statement if all possible cases are processed above.

This macro should be used to catch (in debug builds) logical errors done by the programmer.

See also

wxFAIL()

Include file:

```
#include <wx/debug.h>
```

#define wxFAIL_MSG_AT(*message, file, line, func*)

Assert failure macro pretending to assert at the specified location.

This is a cross between wxASSERT_MSG_AT() and wxFAIL_MSG(), see their documentation for more details.

Since

3.1.0

Include file:

```
#include <wx/debug.h>
```

20.15.3 Typedef Documentation

typedef void(∗ wxAssertHandler_t)(const wxString &file, int line, const wxString &func, const wxString &cond, const wxString &msg)

Type for the function called in case of assert failure.

See also

wxSetAssertHandler()

20.15.4 Function Documentation

void wxAbort ( )

Exits the program immediately.

This is a simple wrapper for the standard abort() function which is not available under all platforms (currently only Windows CE doesn't provide it).

Since

2.9.4

void wxDisableAsserts ( )

Disable the condition checks in the assertions.

This is the same as calling wxSetAssertHandler() with NULL handler.

Since

2.9.0

Include file:

```
#include <wx/debug.h>
```

bool wxIsDebuggerRunning ( )

Returns true if the program is running under debugger, false otherwise.

Please note that this function is currently only implemented for Win32 and always returns false elsewhere.

Include file:

```
#include <wx/debug.h>
```

wxAssertHandler_t wxSetAssertHandler (wxAssertHandler_t *handler*)

Sets the function to be called in case of assertion failure.

The default assert handler forwards to wxApp::OnAssertFailure() whose default behaviour is, in turn, to show the standard assertion failure dialog if a wxApp object exists or shows the same dialog itself directly otherwise.

While usually it is enough – and more convenient – to just override OnAssertFailure(), to handle all assertion failures, including those occurring even before wxApp object creation of after its destruction you need to provide your assertion handler function.

This function also provides a simple way to disable all asserts: simply pass NULL pointer to it. Doing this will result in not even evaluating assert conditions at all, avoiding almost all run-time cost of asserts.

Notice that this function is not MT-safe, so you should call it before starting any other threads.

The return value of this function is the previous assertion handler. It can be called after any pre-processing by your handler and can also be restored later if you uninstall your handler.

Parameters

handler	The function to call in case of assertion failure or NULL.

Returns

The previous assert handler which is not NULL by default but could be NULL if it had been previously set to this value using this function.

Since

2.9.0

Include file:

```
#include <wx/debug.h>
```

void wxSetDefaultAssertHandler ( )

Reset the assert handler to default function which shows a message box when an assert happens.

This can be useful for the applications compiled in release build (with `NDEBUG` defined) for which the asserts are by default disabled: if you wish to enable them even in this case you need to call this function.

Since

2.9.1

Include file:

```
#include <wx/debug.h>
```

void wxTrap ( )

Generate a debugger exception meaning that the control is passed to the debugger if one is attached to the process.

Otherwise the program just terminates abnormally.

If `wxDEBUG_LEVEL` is 0 (which is not the default) this function does nothing.

Include file:

```
#include <wx/debug.h>
```

20.16 Device Contexts

20.16.1 Detailed Description

Device contexts are surfaces that may be drawn on, and provide an abstraction that allows parameterisation of your drawing code by passing different device contexts.

Related Overviews: Device Contexts

Classes

- struct wxFontMetrics

  Simple collection of various font metrics.
- class wxSVGBitmapHandler

  Abstract base class for handling bitmaps inside a wxSVGFileDC.
- class wxSVGBitmapEmbedHandler

  Handler embedding bitmaps as base64-encoded PNGs into the SVG.
- class wxSVGBitmapFileHandler

  Handler saving a bitmap to an external file and linking to it from the SVG.
- class wxDC

  A wxDC is a "device context" onto which graphics and text can be drawn.
- class wxBufferedDC

  This class provides a simple way to avoid flicker: when drawing on it, everything is in fact first drawn on an in-memory buffer (a wxBitmap) and then copied to the screen, using the associated wxDC, only once, when this object is destroyed.
- class wxAutoBufferedPaintDC

  This wxDC derivative can be used inside of an `EVT_PAINT()` event handler to achieve double-buffered drawing.
- class wxBufferedPaintDC

  This is a subclass of wxBufferedDC which can be used inside of an `EVT_PAINT()` event handler to achieve double-buffered drawing.
- class wxPaintDC

  A wxPaintDC must be constructed if an application wishes to paint on the client area of a window from within an EVT_PAINT() event handler.
- class wxClientDC

  A wxClientDC must be constructed if an application wishes to paint on the client area of a window from outside an EVT_PAINT() handler.
- class wxWindowDC

  A wxWindowDC must be constructed if an application wishes to paint on the whole area of a window (client and decorations).
- class wxGCDC

  wxGCDC is a device context that draws on a wxGraphicsContext.
- class wxMemoryDC

  A memory device context provides a means to draw graphics onto a bitmap.
- class wxMirrorDC

  wxMirrorDC is a simple wrapper class which is always associated with a real wxDC object and either forwards all of its operations to it without changes (no mirroring takes place) or exchanges x and y coordinates which makes it possible to reuse the same code to draw a figure and its mirror – i.e.
- class wxPostScriptDC

  This defines the wxWidgets Encapsulated PostScript device context, which can write PostScript files on any platform.
- class wxScreenDC

  A wxScreenDC can be used to paint on the screen.
- class wxSVGFileDC

A wxSVGFileDC is a device context onto which graphics and text can be drawn, and the output produced as a vector file, in SVG format.

- class wxGraphicsContext

  A wxGraphicsContext instance is the object that is drawn upon.

- class wxMetafileDC

  *This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal **wxDC**.*

20.17 Dialogs

20.17.1 Detailed Description

Below are a number of convenience functions for getting input from the user or displaying messages.

Note that in these functions the last three parameters are optional. However, it is recommended to pass a parent frame parameter, or (in MS Windows or Motif) the wrong window frame may be brought to the front when the dialog box is popped up.

Functions

- void wxAboutBox (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  This function shows the standard about dialog containing the information specified in info.
- void wxGenericAboutBox (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  This function does the same thing as wxAboutBox() except that it always uses the generic wxWidgets version of the dialog instead of the native one.
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, const wxArrayString &a↩ Choices, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)

  Same as wxGetSingleChoice() but returns the index representing the selected string.
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, int n, const wxString &choices[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, const wxArrayString &choices, int initialSelection, wxWindow *parent=NULL)
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, int n, const wxString *choices, int initialSelection, wxWindow *parent=NULL)
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, const wxArrayString &aChoices, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)

  Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox.
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, int n, const wxString &choices[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, const wxArrayString &choices, int initialSelection, wxWindow *parent=NULL)
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, int n, const wxString *choices, int initialSelection, wxWindow *parent=NULL)
- wxString wxGetSingleChoiceData (const wxString &message, const wxString &caption, const wxArrayString &aChoices, const wxString &client_data[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wx↩ DefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initial↩ Selection=0)

  Same as wxGetSingleChoice but takes an array of client data pointers corresponding to the strings, and returns one of these pointers or NULL if Cancel was pressed.
- wxString wxGetSingleChoiceData (const wxString &message, const wxString &caption, int n, const wx↩ String &choices[], const wxString &client_data[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wx↩ DefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initial↩ Selection=0)
- void * wxGetSingleChoiceData (const wxString &message, const wxString &caption, const wxArrayString &choices, void **client_data, int initialSelection, wxWindow *parent=NULL)
- void * wxGetSingleChoiceData (const wxString &message, const wxString &caption, int n, const wxString *choices, void **client_data, int initialSelection, wxWindow *parent=NULL)
- int wxGetSelectedChoices (wxArrayInt &selections, const wxString &message, const wxString &caption, const wxArrayString &aChoices, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT)

Pops up a dialog box containing a message, OK/Cancel buttons and a multiple-selection listbox.

- int wxGetSelectedChoices (wxArrayInt &selections, const wxString &message, const wxString &caption, int n, const wxString &choices[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT)
- wxColour wxGetColourFromUser (wxWindow *parent, const wxColour &colInit, const wxString &caption=wxEmptyString, wxColourData *data=NULL)

  Shows the colour selection dialog and returns the colour selected by user or invalid colour (use wxColour::IsOk() to test whether a colour is valid) if the dialog was cancelled.
- wxString wxDirSelector (const wxString &message=wxDirSelectorPromptStr, const wxString &default_path=wxEmptyString, long style=0, const wxPoint &pos=wxDefaultPosition, wxWindow *parent=NULL)

  Pops up a directory selector dialog.
- wxString wxFileSelector (const wxString &message, const wxString &default_path=wxEmptyString, const wxString &default_filename=wxEmptyString, const wxString &default_extension=wxEmptyString, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, int flags=0, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord)

  Pops up a file selector box.
- wxString wxFileSelectorEx (const wxString &message=wxFileSelectorPromptStr, const wxString &default_path=wxEmptyString, const wxString &default_filename=wxEmptyString, int *indexDefaultExtension=NULL, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, int flags=0, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord)

  An extended version of wxFileSelector.
- wxString wxLoadFileSelector (const wxString &what, const wxString &extension, const wxString &default_name=wxEmptyString, wxWindow *parent=NULL)

  Ask for filename to load.
- wxString wxSaveFileSelector (const wxString &what, const wxString &extension, const wxString &default_name=wxEmptyString, wxWindow *parent=NULL)

  Ask for filename to save.
- wxFont wxGetFontFromUser (wxWindow *parent, const wxFont &fontInit, const wxString &caption=wxEmptyString)

  Shows the font selection dialog and returns the font selected by user or invalid font (use wxFont::IsOk() to test whether a font is valid) if the dialog was cancelled.
- int wxMessageBox (const wxString &message, const wxString &caption=wxMessageBoxCaptionStr, int style=wxOK|wxCENTRE, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord)

  Show a general purpose message dialog.
- long wxGetNumberFromUser (const wxString &message, const wxString &prompt, const wxString &caption, long value, long min=0, long max=100, wxWindow *parent=NULL, const wxPoint &pos=wxDefaultPosition)

  Shows a dialog asking the user for numeric input.
- wxString wxGetTextFromUser (const wxString &message, const wxString &caption=wxGetTextFromUserPromptStr, const wxString &default_value=wxEmptyString, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true)

  Pop up a dialog box with title set to caption, `message`, and a `default_value`.
- wxString wxGetPasswordFromUser (const wxString &message, const wxString &caption=wxGetPasswordFromUserPromptStr, const wxString &default_value=wxEmptyString, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true)

  Similar to wxGetTextFromUser() but the text entered in the dialog is not shown on screen but replaced with stars.
- wxTipProvider * wxCreateFileTipProvider (const wxString &filename, size_t currentTip)

  This function creates a wxTipProvider which may be used with wxShowTip().
- bool wxShowTip (wxWindow *parent, wxTipProvider *tipProvider, bool showAtStartup=true)

  This function shows a "startup tip" to the user.
- void wxBeginBusyCursor (const wxCursor *cursor=wxHOURGLASS_CURSOR)

  Changes the cursor to the given cursor for all windows in the application.
- void wxEndBusyCursor ()

  Changes the cursor back to the original cursor, for all windows in the application.
- bool wxIsBusy ()

Returns true if between two wxBeginBusyCursor() and wxEndBusyCursor() calls.

- void wxBell ()

  Ring the system bell.

- void wxInfoMessageBox (wxWindow *parent)

  Shows a message box with the information about the wxWidgets build used, including its version, most important build parameters and the version of the underlying GUI toolkit.

20.17.2 Function Documentation

void wxAboutBox (const wxAboutDialogInfo & *info*, wxWindow * *parent* = NULL)

This function shows the standard about dialog containing the information specified in *info*.

If the current platform has a native about dialog which is capable of showing all the fields in *info*, the native dialog is used, otherwise the function falls back to the generic wxWidgets version of the dialog, i.e. does the same thing as wxGenericAboutBox.

Here is an example of how this function may be used:

```
1 void MyFrame::ShowSimpleAboutDialog(wxCommandEvent& WXUNUSED(event))
2 {
3     wxAboutDialogInfo info;
4     info.SetName(_("My Program"));
5     info.SetVersion(_("1.2.3 Beta"));
6     info.SetDescription(_("This program does something great."));
7     info.SetCopyright(wxT("(C) 2007 Me <my@email.addre.ss>"));
8
9     wxAboutBox(info);
10 }
```

Please see the Dialogs Sample for more examples of using this function and wxAboutDialogInfo for the description of the information which can be shown in the about dialog.

Include file:

```
#include <wx/aboutdlg.h>
```

void wxBeginBusyCursor (const wxCursor * *cursor* = wxHOURGLASS_CURSOR)

Changes the cursor to the given cursor for all windows in the application.

Use wxEndBusyCursor() to revert the cursor back to its previous state. These two calls can be nested, and a counter ensures that only the outer calls take effect.

See also

wxIsBusy(), wxBusyCursor

Include file:

```
#include <wx/utils.h>
```

void wxBell ( )

Ring the system bell.

Note

This function is categorized as a GUI one and so is not thread-safe.

Include file:

```
#include <wx/utils.h>
```

Library: wxCore

wxTipProvider* wxCreateFileTipProvider (const wxString & *filename*, size_t *currentTip*)

This function creates a wxTipProvider which may be used with wxShowTip().

Parameters

filename	The name of the file containing the tips, one per line.
currentTip	The index of the first tip to show. Normally this index is remembered between the 2 program runs.

See also

wxTipProvider Overview

Include file:

```
#include <wx/tipdlg.h>
```

wxString wxDirSelector (const wxString & *message* = wxDirSelectorPromptStr, const wxString & *default_path* = wxEmptyString, long *style* = 0, const wxPoint & *pos* = wxDefaultPosition, wxWindow * *parent* = NULL)

Pops up a directory selector dialog.

The arguments have the same meaning as those of wxDirDialog::wxDirDialog(). The message is displayed at the top, and the default_path, if specified, is set as the initial selection.

The application must check for an empty return value (if the user pressed Cancel). For example:

```
1 const wxString& dir = wxDirSelector("Choose a folder");
2 if ( !dir.empty() )
3 {
4     ...
5 }
```

Include file:

```
#include <wx/dirdlg.h>
```

void wxEndBusyCursor ( )

Changes the cursor back to the original cursor, for all windows in the application.

Use with wxBeginBusyCursor().

See also

wxIsBusy(), wxBusyCursor

Include file:

```
#include <wx/utils.h>
```

wxString wxFileSelector (const wxString & *message*, const wxString & *default_path* = wxEmptyString, const wxString & *default_filename* = wxEmptyString, const wxString & *default_extension* = wxEmptyString, const wxString & *wildcard* = wxFileSelectorDefaultWildcardStr, int *flags* = 0, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord)

Pops up a file selector box.

In Windows, this is the common file selector dialog. In X, this is a file selector box with the same functionality. The path and filename are distinct elements of a full file pathname. If path is empty, the current directory will be used. If filename is empty, no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename. Flags may be a combination of wxFD_OPEN, wxFD_SAVE, wxFD_OVERWRITE_PROMPT or wxFD_FILE_MUST_EXIST.

Note

wxFD_MULTIPLE can only be used with wxFileDialog and not here since this function only returns a single file name.

Both the Unix and Windows versions implement a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed.

The wildcard may be a specification for multiple types of file with a description for each, such as:

```
1 "BMP files (*.bmp)|*.bmp|GIF files (*.gif)|*.gif"
```

The application must check for an empty return value (the user pressed Cancel). For example:

```
1 wxString filename = wxFileSelector("Choose a file to open");
2 if ( !filename.empty() )
3 {
4     // work with the file
5     ...
6 }
7 //else: cancelled by user
```

Include file:

```
#include <wx/filedlg.h>
```

wxString wxFileSelectorEx (const wxString & *message* = wxFileSelectorPromptStr, const wxString & *default_path* = wxEmptyString, const wxString & *default_filename* = wxEmptyString, int * *indexDefaultExtension* = NULL, const wxString & *wildcard* = wxFileSelectorDefaultWildcardStr, int *flags* = 0, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord)

An extended version of wxFileSelector.

void wxGenericAboutBox (const wxAboutDialogInfo & *info*, wxWindow * *parent* = NULL)

This function does the same thing as wxAboutBox() except that it always uses the generic wxWidgets version of the dialog instead of the native one.

This is mainly useful if you need to customize the dialog by e.g. adding custom controls to it (customizing the native dialog is not currently supported).

See the Dialogs Sample for an example of about dialog customization.

See also

wxAboutDialogInfo

Include file:

```
#include <wx/aboutdlg.h>
```

wxColour wxGetColourFromUser (wxWindow * *parent*, const wxColour & *colInit*, const wxString & *caption* = wxEmptyString, wxColourData * *data* = NULL)

Shows the colour selection dialog and returns the colour selected by user or invalid colour (use wxColour::IsOk() to test whether a colour is valid) if the dialog was cancelled.

Parameters

parent	The parent window for the colour selection dialog.
colInit	If given, this will be the colour initially selected in the dialog.
caption	If given, this will be used for the dialog caption.
data	Optional object storing additional colour dialog settings, such as custom colours. If none is provided the same settings as the last time are used.

Include file:

```
#include <wx/colordlg.h>
```

wxFont wxGetFontFromUser (wxWindow * *parent*, const wxFont & *fontInit*, const wxString & *caption* = wxEmptyString)

Shows the font selection dialog and returns the font selected by user or invalid font (use wxFont::IsOk() to test whether a font is valid) if the dialog was cancelled.

Parameters

parent	The parent window for the font selection dialog.
fontInit	If given, this will be the font initially selected in the dialog.
caption	If given, this will be used for the dialog caption.

Include file:

```
#include <wx/fontdlg.h>
```

long wxGetNumberFromUser (const wxString & *message*, const wxString & *prompt*, const wxString & *caption*, long *value*, long *min* = 0, long *max* = 100, wxWindow * *parent* = NULL, const wxPoint & *pos* = wxDefaultPosition)

Shows a dialog asking the user for numeric input.

The dialogs title is set to `caption`, it contains a (possibly) multiline `message` above the single line `prompt` and the zone for entering the number.

The number entered must be in the range `min` to `max` (both of which should be positive) and `value` is the initial value of it. If the user enters an invalid value, it is forced to fall into the specified range. If the user cancels the dialog, the function returns -1.

Dialog is centered on its `parent` unless an explicit position is given in `pos`.

Include file:

```
#include <wx/numdlg.h>
```

wxString wxGetPasswordFromUser (const wxString & *message*, const wxString & *caption* = wxGetPasswordFromUserPromptStr, const wxString & *default_value* = wxEmptyString, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true)

Similar to wxGetTextFromUser() but the text entered in the dialog is not shown on screen but replaced with stars.

This is intended to be used for entering passwords as the function name implies.

Include file:

```
#include <wx/textdlg.h>
```

int wxGetSelectedChoices (wxArrayInt & *selections*, const wxString & *message*, const wxString & *caption*, const wxArrayString & *aChoices*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT)

Pops up a dialog box containing a message, OK/Cancel buttons and a multiple-selection listbox.

The user may choose an arbitrary (including 0) number of items in the listbox whose indices will be returned in `selections` array. The initial contents of this array will be used to select the items when the dialog is shown. If the user cancels the dialog, the function returns -1 and `selections` array is left unchanged.

You may pass the list of strings to choose from either using `choices` which is an array of *n* strings for the listbox or by using a single `aChoices` parameter of type wxArrayString.

If `centre` is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

Include file:

```
#include <wx/choicdlg.h>
```

wxPerl Note: Use an array reference for the *choices* parameter. In wxPerl there is no *selections* parameter; the function returns an array containing the user selections.

int wxGetSelectedChoices (wxArrayInt & *selections*, const wxString & *message*, const wxString & *caption*, int *n*, const wxString & *choices[]*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT)

wxString wxGetSingleChoice (const wxString & *message*, const wxString & *caption*, const wxArrayString & *aChoices*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT, int *initialSelection* = 0)

Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox.

The user may choose an item and press OK to return a string or Cancel to return the empty string. Use wxGetSingleChoiceIndex() if empty string is a valid choice and if you want to be able to detect pressing Cancel reliably.

You may pass the list of strings to choose from either using `choices` which is an array of *n* strings for the listbox or by using a single `aChoices` parameter of type wxArrayString.

If `centre` is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

Include file:

```
#include <wx/choicdlg.h>
```

wxPerl Note: Use an array reference for the *choices* parameter.

wxString wxGetSingleChoice (const wxString & *message*, const wxString & *caption*, int *n*, const wxString & *choices[]*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT, int *initialSelection* = 0)

wxString wxGetSingleChoice (const wxString & *message*, const wxString & *caption*, const wxArrayString & *choices*, int *initialSelection*, wxWindow * *parent* = NULL)

wxString wxGetSingleChoice (const wxString & *message*, const wxString & *caption*, int *n*, const wxString * *choices*, int *initialSelection*, wxWindow * *parent* = NULL)

wxString wxGetSingleChoiceData (const wxString & *message*, const wxString & *caption*, const wxArrayString & *aChoices*, const wxString & *client_data[]*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT, int *initialSelection* = 0)

Same as wxGetSingleChoice but takes an array of client data pointers corresponding to the strings, and returns one of these pointers or NULL if Cancel was pressed.

The `client_data` array must have the same number of elements as `choices` or `aChoices`!

Include file:

```
#include <wx/choicdlg.h>
```

wxPerl Note: Use an array reference for the *aChoices* and *client_data* parameters.

wxString wxGetSingleChoiceData (const wxString & *message*, const wxString & *caption*, int *n*, const wxString & *choices[]*, const wxString & *client_data[]*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT, int *initialSelection* = 0)

void* wxGetSingleChoiceData (const wxString & *message*, const wxString & *caption*, const wxArrayString & *choices*, void ** *client_data*, int *initialSelection*, wxWindow * *parent* = NULL)

void* wxGetSingleChoiceData (const wxString & *message*, const wxString & *caption*, int *n*, const wxString * *choices*, void ** *client_data*, int *initialSelection*, wxWindow * *parent* = NULL)

int wxGetSingleChoiceIndex (const wxString & *message*, const wxString & *caption*, const wxArrayString & *aChoices*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT, int *initialSelection* = 0)

Same as wxGetSingleChoice() but returns the index representing the selected string.

If the user pressed cancel, -1 is returned.

Include file:

```
#include <wx/choicdlg.h>
```

wxPerl Note: Use an array reference for the *aChoices* parameter.

int wxGetSingleChoiceIndex (const wxString & *message*, const wxString & *caption*, int *n*, const wxString & *choices[]*, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true, int *width* = wxCHOICE_WIDTH, int *height* = wxCHOICE_HEIGHT, int *initialSelection* = 0)

int wxGetSingleChoiceIndex (const wxString & *message*, const wxString & *caption*, const wxArrayString & *choices*, int *initialSelection*, wxWindow * *parent* = NULL)

int wxGetSingleChoiceIndex (const wxString & *message*, const wxString & *caption*, int *n*, const wxString * *choices*, int *initialSelection*, wxWindow * *parent* = NULL)

wxString wxGetTextFromUser (const wxString & *message*, const wxString & *caption* = wxGetTextFromUser↩ PromptStr, const wxString & *default_value* = wxEmptyString, wxWindow * *parent* = NULL, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord, bool *centre* = true)

Pop up a dialog box with title set to *caption*, `message`, and a `default_value`.

The user may type in text and press OK to return this text, or press Cancel to return the empty string.

If `centre` is true, the message text (which may include new line characters) is centred; if false, the message is left-justified.

This function is a wrapper around wxTextEntryDialog and while it is usually more convenient to use, using the dialog directly is more flexible, e.g. it allows you to specify the `wxTE_MULTILINE` to allow the user enter multiple lines of text while this function is limited to single line entry only.

Include file:

```
#include <wx/textdlg.h>
```

void wxInfoMessageBox (wxWindow * *parent*)

Shows a message box with the information about the wxWidgets build used, including its version, most important build parameters and the version of the underlying GUI toolkit.

This is mainly used for diagnostic purposes and can be invoked by Ctrl-Alt-middle clicking on any wxWindow which doesn't otherwise handle this event.

Since

2.9.0

See also

wxGetLibraryVersionInfo()

Include file:

```
#include <wx/utils.h>
```

bool wxIsBusy ( )

Returns true if between two wxBeginBusyCursor() and wxEndBusyCursor() calls.

See also

wxBusyCursor.

Include file:

```
#include <wx/utils.h>
```

wxString wxLoadFileSelector (const wxString & *what*, const wxString & *extension*, const wxString & *default_name* = wxEmptyString, wxWindow * *parent* = `NULL`)

Ask for filename to load.

int wxMessageBox (const wxString & *message*, const wxString & *caption* = wxMessageBoxCaptionStr, int *style* = wxOK|wxCENTRE, wxWindow * *parent* = `NULL`, int *x* = wxDefaultCoord, int *y* = wxDefaultCoord)

Show a general purpose message dialog.

This is a convenient function which is usually used instead of using wxMessageDialog directly. Notice however that some of the features, such as extended text and custom labels for the message box buttons, are not provided by this function but only by wxMessageDialog.

The return value is one of: `wxYES`, `wxNO`, `wxCANCEL`, `wxOK` or `wxHELP` (notice that this return value is **different** from the return value of wxMessageDialog::ShowModal()).

For example:

```
1 int answer = wxMessageBox("Quit program?", "Confirm",
2                           wxYES_NO | wxCANCEL, main_frame);
3 if (answer == wxYES)
4     main_frame->Close();
```

message may contain newline characters, in which case the message will be split into separate lines, to cater for large messages.

Parameters

message	Message to show in the dialog.
caption	The dialog title.
parent	Parent window.
style	Combination of style flags described in wxMessageDialog documentation.
x	Horizontal dialog position (ignored under MSW). Use wxDefaultCoord for *x* and *y* to let the system position the window.
y	Vertical dialog position (ignored under MSW).

Include file:

```
#include <wx/msgdlg.h>
```

wxString wxSaveFileSelector (const wxString & *what*, const wxString & *extension*, const wxString & *default_name* = wxEmptyString, wxWindow * *parent* = NULL)

Ask for filename to save.

bool wxShowTip (wxWindow * *parent*, wxTipProvider * *tipProvider*, bool *showAtStartup* = true)

This function shows a "startup tip" to the user.

The return value is the state of the "Show tips at startup" checkbox.

Parameters

parent	The parent window for the modal dialog.
tipProvider	An object which is used to get the text of the tips. It may be created with the wxCreateFileTipProvider() function.
showAtStartup	Should be true if startup tips are shown, false otherwise. This is used as the initial value for "Show tips at startup" checkbox which is shown in the tips dialog.

See also

wxTipProvider Overview

Include file:

```
#include <wx/tipdlg.h>
```

20.18 Document/View Framework

20.18.1 Detailed Description

wxWidgets supports a document/view framework which provides housekeeping for a document-centric application.

Related Overviews: Document/View Framework

Classes

- class wxCommand

  wxCommand is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other means provided by the application to change the data or view.
- class wxCommandProcessor

  wxCommandProcessor is a class that maintains a history of wxCommands, with undo/redo functionality built-in.
- class wxDocMDIParentFrame

  The wxDocMDIParentFrame class provides a default top-level frame for applications using the document/view framework.
- class wxDocMDIChildFrame

  The wxDocMDIChildFrame class provides a default frame for displaying documents on separate windows.
- class wxDocTemplate

  The wxDocTemplate class is used to model the relationship between a document class and a view class.
- class wxDocManager

  The wxDocManager class is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocument and wxDocTemplate classes.
- class wxView

  The view class can be used to model the viewing and editing component of an application's file-based data.
- class wxDocChildFrame

  The wxDocChildFrame class provides a default frame for displaying documents on separate windows.
- class wxDocParentFrame

  The wxDocParentFrame class provides a default top-level frame for applications using the document/view framework.
- class wxDocument

  The document class can be used to model an application's file-based data.
- class wxFileHistory

  The wxFileHistory encapsulates a user interface convenience, the list of most recently visited files as shown on a menu (usually the File menu).

20.19 Environment

20.19.1 Detailed Description

These functions allow access to get or change the values of environment variables in a portable way.

They are currently implemented under Win32 and POSIX-like systems (Unix).

Remember that these functions add/change/delete environment variables of the current process only. Child processes copy the environment variables of the parent but do not share them (a wxSetEnv() in the parent process won't change the value returned by wxGetEnv() in the child process and viceversa).

For more evoluted interprocess communication tecniques see Interprocess Communication.

Typedefs

- typedef wxStringToStringHashMap wxEnvVariableHashMap

  A map type containing environment variables names and values.

Functions

- wxChar * wxGetenv (const wxString &var)

  This is a macro defined as `getenv()` or its wide char version in Unicode mode.
- bool wxGetEnv (const wxString &var, wxString *value)

  Returns the current value of the environment variable var in value.
- bool wxSetEnv (const wxString &var, const wxString &value)

  Sets the value of the environment variable var (adding it if necessary) to value.
- bool wxUnsetEnv (const wxString &var)

  Removes the variable var from the environment.
- bool wxGetEnvMap (wxEnvVariableHashMap *map)

  Fill a map with the complete content of current environment.

20.19.2 Typedef Documentation

typedef wxStringToStringHashMap wxEnvVariableHashMap

A map type containing environment variables names and values.

This type is used with wxGetEnvMap() function and wxExecuteEnv structure optionally passed to wxExecute().

Since

2.9.2

Include file:

```
#include <wx/utils.h>
```

20.19.3 Function Documentation

wxChar* wxGetenv (const wxString & *var*)

This is a macro defined as `getenv()` or its wide char version in Unicode mode.

Note that under Win32 it may not return correct value for the variables set with wxSetEnv(), use wxGetEnv() function instead.

Include file:

```
#include <wx/utils.h>
```

bool wxGetEnv (const wxString & *var,* wxString * *value*)

Returns the current value of the environment variable *var* in *value*.

value may be NULL if you just want to know if the variable exists and are not interested in its value.

Returns true if the variable exists, false otherwise.

Include file:

```
#include <wx/utils.h>
```

bool wxGetEnvMap (wxEnvVariableHashMap * *map*)

Fill a map with the complete content of current environment.

The map will contain the environment variable names as keys and their values as values.

Parameters

map	The environment map to fill, must be non-NULL.

Returns

true if environment was successfully retrieved or false otherwise.

Include file:

```
#include <wx/utils.h>
```

Since

2.9.2

bool wxSetEnv (const wxString & *var,* const wxString & *value*)

Sets the value of the environment variable *var* (adding it if necessary) to *value*.

Notice that under Windows platforms the program may have two different environment blocks: the first one is that of a Windows process and is always present, but the CRT may maintain its own independent copy of the environment. wxSetEnv() will always update the first copy, which means that wxGetEnv(), which uses it directly, will always return the expected value after this call. But wxSetEnv() only updates the second copy for some compilers/CRT implementations (currently only MSVC and MinGW which uses the same MSVC CRT) and so using wxGetenv() (notice the difference in case) may not return the updated value.

Parameters

var	The environment variable to be set, must not contain `'='` character.
value	New value of the variable.

Returns

true on success or false if changing the value failed.

See also

wxUnsetEnv()

Include file:

```
#include <wx/utils.h>
```

bool wxUnsetEnv (const wxString & *var*)

Removes the variable *var* from the environment.

wxGetEnv() will return NULL after the call to this function.

Returns true on success.

Include file:

```
#include <wx/utils.h>
```

20.20 Events

20.20.1 Detailed Description

An event object contains information about a specific event.

Event handlers (usually member functions) have a single, event argument.

Related Overviews: Events and Event Handling

Related macros/global-functions group: Events

Classes

- class wxEventFilter

  A global event filter for pre-processing all the events generated in the program.
- class wxKeyboardState

  Provides methods for testing the state of the keyboard modifier keys.
- class wxAuiNotebookEvent

  This class is used by the events generated by wxAuiNotebook.
- class wxAuiManagerEvent

  Event used to indicate various actions taken with wxAuiManager.
- class wxBookCtrlEvent

  This class represents the events generated by book controls (wxNotebook, wxListbook, wxChoicebook, wxTreebook, wxAuiNotebook).
- class wxCalendarEvent

  The wxCalendarEvent class is used together with wxCalendarCtrl.
- class wxColourPickerEvent

  This event class is used for the events generated by wxColourPickerCtrl.
- class wxCollapsiblePaneEvent

  This event class is used for the events generated by wxCollapsiblePane.
- class wxDataViewEvent

  This is the event class for the wxDataViewCtrl notifications.
- class wxDateEvent

  This event class holds information about a date change and is used together with wxDatePickerCtrl.
- class wxDialUpEvent

  This is the event class for the dialup events sent by wxDialUpManager.
- class wxEvent

  An event is a structure holding information about an event passed to a callback or member function.
- class wxEventBlocker

  This class is a special event handler which allows to discard any event (or a set of event types) directed to a specific window.
- class wxEvtHandler

  A class that can handle events from the windowing system.
- class wxKeyEvent

  This event class contains information about key press and release events.
- class wxJoystickEvent

  This event class contains information about joystick events, particularly events received by windows.
- class wxScrollWinEvent

  A scroll event holds information about events sent from scrolling windows.
- class wxSysColourChangedEvent

  This class is used for system colour change events, which are generated when the user changes the colour settings using the control panel.

- class wxCommandEvent

  This event class contains information about command events, which originate from a variety of simple controls.
- class wxWindowCreateEvent

  This event is sent just after the actual window associated with a wxWindow object has been created.
- class wxPaintEvent

  A paint event is sent when a window's contents needs to be repainted.
- class wxMaximizeEvent

  An event being sent when a top level window is maximized.
- class wxUpdateUIEvent

  This class is used for pseudo-events which are called by wxWidgets to give an application the chance to update various user interface elements.
- class wxClipboardTextEvent

  This class represents the events generated by a control (typically a wxTextCtrl but other windows can generate these events as well) when its content gets copied or cut to, or pasted from the clipboard.
- class wxMouseEvent

  This event class contains information about the events generated by the mouse: they include mouse buttons press and release events and mouse move events.
- class wxDropFilesEvent

  This class is used for drop files events, that is, when files have been dropped onto the window.
- class wxActivateEvent

  An activate event is sent when a window or application is being activated or deactivated.
- class wxContextMenuEvent

  This class is used for context menu events, sent to give the application a chance to show a context (popup) menu for a wxWindow.
- class wxEraseEvent

  An erase event is sent when a window's background needs to be repainted.
- class wxFocusEvent

  A focus event is sent when a window's focus changes.
- class wxChildFocusEvent

  A child focus event is sent to a (parent-)window when one of its child windows gains focus, so that the window could restore the focus back to its corresponding child if it loses it now and regains later.
- class wxMouseCaptureLostEvent

  A mouse capture lost event is sent to a window that had obtained mouse capture, which was subsequently lost due to an "external" event (for example, when a dialog box is shown or if another application captures the mouse).
- class wxNotifyEvent

  This class is not used by the event handlers by itself, but is a base class for other event classes (such as wxBook↩ CtrlEvent).
- class wxThreadEvent

  This class adds some simple functionality to wxEvent to facilitate inter-thread communication.
- class wxHelpEvent

  A help event is sent when the user has requested context-sensitive help.
- class wxScrollEvent

  A scroll event holds information about events sent from stand-alone scrollbars (see wxScrollBar) and sliders (see wxSlider).
- class wxIdleEvent

  This class is used for idle events, which are generated when the system becomes idle.
- class wxInitDialogEvent

  A wxInitDialogEvent is sent as a dialog or panel is being initialised.
- class wxWindowDestroyEvent

  This event is sent as early as possible during the window destruction process.
- class wxNavigationKeyEvent

  This event class contains information about navigation events, generated by navigation keys such as tab and page down.

- class wxMouseCaptureChangedEvent

  An mouse capture changed event is sent to a window that loses its mouse capture.
- class wxCloseEvent

  This event class contains information about window and session close events.
- class wxMenuEvent

  This class is used for a variety of menu-related events.
- class wxShowEvent

  An event being sent when the window is shown or hidden.
- class wxIconizeEvent

  An event being sent when the frame is iconized (minimized) or restored.
- class wxMoveEvent

  A move event holds information about wxTopLevelWindow move change events.
- class wxSizeEvent

  A size event holds information about size change events of wxWindow.
- class wxSetCursorEvent

  A wxSetCursorEvent is generated from wxWindow when the mouse cursor is about to be set as a result of mouse motion.
- class wxFindDialogEvent

  wxFindReplaceDialog events.
- class wxFileCtrlEvent

  A file control event holds information about events associated with wxFileCtrl objects.
- class wxFileDirPickerEvent

  This event class is used for the events generated by wxFilePickerCtrl and by wxDirPickerCtrl.
- class wxFontPickerEvent

  This event class is used for the events generated by wxFontPickerCtrl.
- class wxFileSystemWatcherEvent

  A class of events sent when a file system event occurs.
- class wxGridEvent

  This event class contains information about various grid events.
- class wxGridSizeEvent

  This event class contains information about a row/column resize event.
- class wxGridRangeSelectEvent
- class wxGridEditorCreatedEvent
- class wxHeaderCtrlEvent

  Event class representing the events generated by wxHeaderCtrl.
- class wxHyperlinkEvent

  This event class is used for the events generated by wxHyperlinkCtrl.
- class wxQueryLayoutInfoEvent

  This event is sent when wxLayoutAlgorithm wishes to get the size, orientation and alignment of a window.
- class wxCalculateLayoutEvent

  This event is sent by wxLayoutAlgorithm to calculate the amount of the remaining client area that the window should occupy.
- class wxListEvent

  A list event holds information about events associated with wxListCtrl objects.
- class wxMediaEvent

  Event wxMediaCtrl uses.
- class wxMouseEventsManager

  Helper for handling mouse input events in windows containing multiple items.
- class wxMouseState

  Represents the mouse state.
- class wxActiveXEvent

An event class for handling ActiveX events passed from wxActiveXContainer.

- class wxPowerEvent

  The power events are generated when the system power state changes, e.g.
- class wxProcessEvent

  A process event is sent to the wxEvtHandler specified to wxProcess when a process is terminated.
- class wxRibbonBarEvent

  Event used to indicate various actions relating to a wxRibbonBar.
- class wxRibbonButtonBarEvent

  Event used to indicate various actions relating to a button on a wxRibbonButtonBar.
- class wxRibbonGalleryEvent
- class wxRibbonPanelEvent

  Event used to indicate various actions relating to a wxRibbonPanel.
- class wxRichTextEvent

  This is the event class for wxRichTextCtrl notifications.
- class wxSashEvent

  A sash event is sent when the sash of a wxSashWindow has been dragged by the user.
- class wxSpinEvent

  This event class is used for the events generated by wxSpinButton and wxSpinCtrl.
- class wxSpinDoubleEvent

  This event class is used for the events generated by wxSpinCtrlDouble.
- class wxSplitterEvent

  This class represents the events generated by a splitter control.
- class wxStyledTextEvent

  The type of events sent from wxStyledTextCtrl.
- class wxTaskBarIconEvent

  The event class used by wxTaskBarIcon.
- class wxTimerEvent

  wxTimerEvent object is passed to the event handler of timer events (see wxTimer::SetOwner).
- class wxTreeEvent

  A tree event holds information about events associated with wxTreeCtrl objects.
- class wxWebViewEvent

  A navigation event holds information about events associated with wxWebView objects.
- class wxWizardEvent

  wxWizardEvent class represents an event generated by the wxWizard: this event is first sent to the page itself and, if not processed there, goes up the window hierarchy as usual.

Enumerations

- enum {
  wxEventFilter::Event_Skip = -1,
  wxEventFilter::Event_Ignore = 0,
  wxEventFilter::Event_Processed = 1 }

  Possible return values for FilterEvent().

20.20.2 Enumeration Type Documentation

anonymous enum

Possible return values for FilterEvent().

Enumerator

Event_Skip Process event as usual.

Event_Ignore Don't process the event normally at all.

Event_Processed Event was already handled, don't process it normally.

20.21 Events

20.21.1 Detailed Description

Below are a number of functions/macros used with wxWidgets event-handling system.

Related class group: Events

Macros

- #define wxDEFINE_EVENT(name, cls) const wxEventTypeTag< cls > name(wxNewEventType())

  Define a new event type associated with the specified event class.
- #define wxDECLARE_EVENT(name, cls) wxDECLARE_EXPORTED_EVENT(wxEMPTY_PARAMETER_↩ VALUE, name, cls)

  Declares a custom event type.
- #define wxDECLARE_EXPORTED_EVENT(expdecl, name, cls) extern const expdecl wxEventTypeTag< cls > name;

  Variant of wxDECLARE_EVENT() used for event types defined inside a shared library.
- #define wxEVENT_HANDLER_CAST(functype, func) (&func)

  Helper macro for definition of custom event table macros.
- #define wx__DECLARE_EVT1(evt, id, fn) wx__DECLARE_EVT2(evt, id, wxID_ANY, fn)

  This macro is used to define event table macros for handling custom events.
- #define wx__DECLARE_EVT2(evt, id1, id2, fn) DECLARE_EVENT_TABLE_ENTRY(evt, id1, id2, fn, NULL),

  Generalized version of the wx__DECLARE_EVT1() macro taking a range of IDs instead of a single one.
- #define wx__DECLARE_EVT0(evt, fn) wx__DECLARE_EVT1(evt, wxID_ANY, fn)

  Simplified version of the wx__DECLARE_EVT1() macro, to be used when the event type must be handled regardless of the ID associated with the specific event instances.
- #define wxDECLARE_EVENT_TABLE()

  Use this macro inside a class declaration to declare a static event table for that class.
- #define wxBEGIN_EVENT_TABLE(theClass, baseClass)

  Use this macro in a source file to start listing static event handlers for a specific class.
- #define wxEND_EVENT_TABLE()

  Use this macro in a source file to end listing static event handlers for a specific class.

Typedefs

- typedef int wxEventType

  A value uniquely identifying the type of the event.

Functions

- wxEventType wxNewEventType ()

  Generates a new unique event type.
- void wxPostEvent (wxEvtHandler *dest, const wxEvent &event)

  In a GUI application, this function posts event to the specified dest object using wxEvtHandler::AddPendingEvent().
- void wxQueueEvent (wxEvtHandler *dest, wxEvent *event)

  Queue an event for processing on the given object.

Variables

- wxEventType wxEVT_NULL

  A special event type usually used to indicate that some wxEvent has yet no type assigned.
- wxEventType wxEVT_ANY
- wxEventType wxEVT_BUTTON
- wxEventType wxEVT_CHECKBOX
- wxEventType wxEVT_CHOICE
- wxEventType wxEVT_LISTBOX
- wxEventType wxEVT_LISTBOX_DCLICK
- wxEventType wxEVT_CHECKLISTBOX
- wxEventType wxEVT_MENU
- wxEventType wxEVT_SLIDER
- wxEventType wxEVT_RADIOBOX
- wxEventType wxEVT_RADIOBUTTON
- wxEventType wxEVT_SCROLLBAR
- wxEventType wxEVT_VLBOX
- wxEventType wxEVT_COMBOBOX
- wxEventType wxEVT_TOOL_RCLICKED
- wxEventType wxEVT_TOOL_DROPDOWN
- wxEventType wxEVT_TOOL_ENTER
- wxEventType wxEVT_COMBOBOX_DROPDOWN
- wxEventType wxEVT_COMBOBOX_CLOSEUP
- wxEventType wxEVT_THREAD
- wxEventType wxEVT_LEFT_DOWN
- wxEventType wxEVT_LEFT_UP
- wxEventType wxEVT_MIDDLE_DOWN
- wxEventType wxEVT_MIDDLE_UP
- wxEventType wxEVT_RIGHT_DOWN
- wxEventType wxEVT_RIGHT_UP
- wxEventType wxEVT_MOTION
- wxEventType wxEVT_ENTER_WINDOW
- wxEventType wxEVT_LEAVE_WINDOW
- wxEventType wxEVT_LEFT_DCLICK
- wxEventType wxEVT_MIDDLE_DCLICK
- wxEventType wxEVT_RIGHT_DCLICK
- wxEventType wxEVT_SET_FOCUS
- wxEventType wxEVT_KILL_FOCUS
- wxEventType wxEVT_CHILD_FOCUS
- wxEventType wxEVT_MOUSEWHEEL
- wxEventType wxEVT_AUX1_DOWN
- wxEventType wxEVT_AUX1_UP
- wxEventType wxEVT_AUX1_DCLICK
- wxEventType wxEVT_AUX2_DOWN
- wxEventType wxEVT_AUX2_UP
- wxEventType wxEVT_AUX2_DCLICK
- wxEventType wxEVT_CHAR
- wxEventType wxEVT_CHAR_HOOK
- wxEventType wxEVT_NAVIGATION_KEY
- wxEventType wxEVT_KEY_DOWN
- wxEventType wxEVT_KEY_UP
- wxEventType wxEVT_HOTKEY
- wxEventType wxEVT_SET_CURSOR
- wxEventType wxEVT_SCROLL_TOP
- wxEventType wxEVT_SCROLL_BOTTOM

- wxEventType wxEVT_SCROLL_LINEUP
- wxEventType wxEVT_SCROLL_LINEDOWN
- wxEventType wxEVT_SCROLL_PAGEUP
- wxEventType wxEVT_SCROLL_PAGEDOWN
- wxEventType wxEVT_SCROLL_THUMBTRACK
- wxEventType wxEVT_SCROLL_THUMBRELEASE
- wxEventType wxEVT_SCROLL_CHANGED
- wxEventType wxEVT_SPIN_UP
- wxEventType wxEVT_SPIN_DOWN
- wxEventType wxEVT_SPIN
- wxEventType wxEVT_SCROLLWIN_TOP
- wxEventType wxEVT_SCROLLWIN_BOTTOM
- wxEventType wxEVT_SCROLLWIN_LINEUP
- wxEventType wxEVT_SCROLLWIN_LINEDOWN
- wxEventType wxEVT_SCROLLWIN_PAGEUP
- wxEventType wxEVT_SCROLLWIN_PAGEDOWN
- wxEventType wxEVT_SCROLLWIN_THUMBTRACK
- wxEventType wxEVT_SCROLLWIN_THUMBRELEASE
- wxEventType wxEVT_SIZE
- wxEventType wxEVT_MOVE
- wxEventType wxEVT_CLOSE_WINDOW
- wxEventType wxEVT_END_SESSION
- wxEventType wxEVT_QUERY_END_SESSION
- wxEventType wxEVT_ACTIVATE_APP
- wxEventType wxEVT_ACTIVATE
- wxEventType wxEVT_CREATE
- wxEventType wxEVT_DESTROY
- wxEventType wxEVT_SHOW
- wxEventType wxEVT_ICONIZE
- wxEventType wxEVT_MAXIMIZE
- wxEventType wxEVT_MOUSE_CAPTURE_CHANGED
- wxEventType wxEVT_MOUSE_CAPTURE_LOST
- wxEventType wxEVT_PAINT
- wxEventType wxEVT_ERASE_BACKGROUND
- wxEventType wxEVT_NC_PAINT
- wxEventType wxEVT_MENU_OPEN
- wxEventType wxEVT_MENU_CLOSE
- wxEventType wxEVT_MENU_HIGHLIGHT
- wxEventType wxEVT_CONTEXT_MENU
- wxEventType wxEVT_SYS_COLOUR_CHANGED
- wxEventType wxEVT_DISPLAY_CHANGED
- wxEventType wxEVT_QUERY_NEW_PALETTE
- wxEventType wxEVT_PALETTE_CHANGED
- wxEventType wxEVT_JOY_BUTTON_DOWN
- wxEventType wxEVT_JOY_BUTTON_UP
- wxEventType wxEVT_JOY_MOVE
- wxEventType wxEVT_JOY_ZMOVE
- wxEventType wxEVT_DROP_FILES
- wxEventType wxEVT_INIT_DIALOG
- wxEventType wxEVT_IDLE
- wxEventType wxEVT_UPDATE_UI
- wxEventType wxEVT_SIZING
- wxEventType wxEVT_MOVING
- wxEventType wxEVT_MOVE_START
- wxEventType wxEVT_MOVE_END

- wxEventType wxEVT_HIBERNATE
- wxEventType wxEVT_TEXT_COPY
- wxEventType wxEVT_TEXT_CUT
- wxEventType wxEVT_TEXT_PASTE
- wxEventType wxEVT_COMMAND_LEFT_CLICK
- wxEventType wxEVT_COMMAND_LEFT_DCLICK
- wxEventType wxEVT_COMMAND_RIGHT_CLICK
- wxEventType wxEVT_COMMAND_RIGHT_DCLICK
- wxEventType wxEVT_COMMAND_SET_FOCUS
- wxEventType wxEVT_COMMAND_KILL_FOCUS
- wxEventType wxEVT_COMMAND_ENTER
- wxEventType wxEVT_HELP
- wxEventType wxEVT_DETAILED_HELP
- wxEventType wxEVT_TOOL
- wxEventType wxEVT_WINDOW_MODAL_DIALOG_CLOSED

20.21.2 Macro Definition Documentation

#define wx__DECLARE_EVT0(*evt, fn*) wx__DECLARE_EVT1(evt, wxID_ANY, fn)

Simplified version of the wx__DECLARE_EVT1() macro, to be used when the event type must be handled regardless of the ID associated with the specific event instances.

#define wx__DECLARE_EVT1(*evt, id, fn*) wx__DECLARE_EVT2(evt, id, wxID_ANY, fn)

This macro is used to define event table macros for handling custom events.

Example of use:

```
1 class MyEvent : public wxEvent { ... };
2
3 // note that this is not necessary unless using old compilers: for the
4 // reasonably new ones just use &func instead of MyEventHandler(func)
5 typedef void (wxEvtHandler::*MyEventFunction)(MyEvent&);
6 #define MyEventHandler(func) wxEVENT_HANDLER_CAST(MyEventFunction, func)
7
8 wxDEFINE_EVENT(MY_EVENT_TYPE, MyEvent);
9
10 #define EVT_MY(id, func) \
11     wx__DECLARE_EVT1(MY_EVENT_TYPE, id, MyEventHandler(func))
12
13 ...
14
15 wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
16     EVT_MY(wxID_ANY, MyFrame::OnMyEvent)
17 wxEND_EVENT_TABLE()
```

Parameters

evt	The event type to handle.
id	The identifier of events to handle.
fn	The event handler method.

#define wx__DECLARE_EVT2(*evt, id1, id2, fn*) DECLARE_EVENT_TABLE_ENTRY(evt, id1, id2, fn, NULL),

Generalized version of the wx__DECLARE_EVT1() macro taking a range of IDs instead of a single one.

Argument *id1* is the first identifier of the range, *id2* is the second identifier of the range.

#define wxBEGIN_EVENT_TABLE(*theClass, baseClass*)

Use this macro in a source file to start listing *static* event handlers for a specific class.

Use wxEND_EVENT_TABLE() to terminate the event-declaration block.

See also

Event Handling with Event Tables

#define wxDECLARE_EVENT(*name, cls*) wxDECLARE_EXPORTED_EVENT(wxEMPTY_PARAMETER_VALUE, name, cls)

Declares a custom event type.

This macro declares a variable called *name* which must be defined elsewhere using wxDEFINE_EVENT().

The class *cls* must be the wxEvent-derived class associated with the events of this type and its full declaration must be visible from the point of use of this macro.

For example:

```
1 wxDECLARE_EVENT(MY_COMMAND_EVENT, wxCommandEvent);
2
3 class MyCustomEvent : public wxEvent { ... };
4 wxDECLARE_EVENT(MY_CUSTOM_EVENT, MyCustomEvent);
```

#define wxDECLARE_EVENT_TABLE( )

Use this macro inside a class declaration to declare a *static* event table for that class.

In the implementation file you'll need to use the wxBEGIN_EVENT_TABLE() and the wxEND_EVENT_TABLE() macros, plus some additional `EVT_xxx` macro to capture events.

Note that this macro requires a final semicolon.

See also

Event Handling with Event Tables

#define wxDECLARE_EXPORTED_EVENT(*expdecl, name, cls*) extern const expdecl wxEventTypeTag< cls > name;

Variant of wxDECLARE_EVENT() used for event types defined inside a shared library.

This is mostly used by wxWidgets internally, e.g.

```
1 wxDECLARE_EXPORTED_EVENT(WXDLLIMPEXP_CORE, wxEVT_BUTTON, wxCommandEvent)
```

#define wxDEFINE_EVENT(*name, cls*) const wxEventTypeTag< cls > name(wxNewEventType())

Define a new event type associated with the specified event class.

This macro defines a new unique event type *name* associated with the event class *cls*.

For example:

```
1 wxDEFINE_EVENT(MY_COMMAND_EVENT, wxCommandEvent);
2
3 class MyCustomEvent : public wxEvent { ... };
4 wxDEFINE_EVENT(MY_CUSTOM_EVENT, MyCustomEvent);
```

See also

wxDECLARE_EVENT(), Custom Event Summary

#define wxEND_EVENT_TABLE( )

Use this macro in a source file to end listing *static* event handlers for a specific class.

Use wxBEGIN_EVENT_TABLE() to start the event-declaration block.

See also

Event Handling with Event Tables

#define wxEVENT_HANDLER_CAST(*functype, func*) (&func)

Helper macro for definition of custom event table macros.

This macro must only be used if wxEVENTS_COMPATIBILITY_2_8 is 1, otherwise it is better and more clear to just use the address of the function directly as this is all this macro does in this case. However it needs to explicitly cast *func* to *functype*, which is the type of wxEvtHandler member function taking the custom event argument when wxEVENTS_COMPATIBILITY_2_8 is 0.

See wx__DECLARE_EVT0 for an example of use.

See also

Defining Your Own Event Class

20.21.3 Typedef Documentation

typedef int wxEventType

A value uniquely identifying the type of the event.

The values of this type should only be created using wxNewEventType().

See the macro DEFINE_EVENT_TYPE() for more info.

See also

Events and Event Handling

20.21.4 Function Documentation

wxEventType wxNewEventType ( )

Generates a new unique event type.

Usually this function is only used by wxDEFINE_EVENT() and not called directly.

void wxPostEvent (wxEvtHandler * *dest,* const wxEvent & *event*)

In a GUI application, this function posts *event* to the specified *dest* object using wxEvtHandler::AddPendingEvent().

Otherwise, it dispatches *event* immediately using wxEvtHandler::ProcessEvent(). See the respective documentation for details (and caveats). Because of limitation of wxEvtHandler::AddPendingEvent() this function is not thread-safe for event objects having wxString fields, use wxQueueEvent() instead.

Include file:

```
#include <wx/event.h>
```

void wxQueueEvent (wxEvtHandler * *dest,* wxEvent * *event*)

Queue an event for processing on the given object.

This is a wrapper around wxEvtHandler::QueueEvent(), see its documentation for more details.

Include file:

```
#include <wx/event.h>
```

Parameters

dest	The object to queue the event on, can't be NULL.
event	The heap-allocated and non-NULL event to queue, the function takes ownership of it.

20.21.5 Variable Documentation

wxEventType wxEVT_ACTIVATE

wxEventType wxEVT_ACTIVATE_APP

wxEventType wxEVT_ANY

wxEventType wxEVT_AUX1_DCLICK

wxEventType wxEVT_AUX1_DOWN

wxEventType wxEVT_AUX1_UP

wxEventType wxEVT_AUX2_DCLICK

wxEventType wxEVT_AUX2_DOWN

wxEventType wxEVT_AUX2_UP

wxEventType wxEVT_BUTTON

wxEventType wxEVT_CHAR

wxEventType wxEVT_CHAR_HOOK

wxEventType wxEVT_CHECKBOX

wxEventType wxEVT_CHECKLISTBOX

wxEventType wxEVT_CHILD_FOCUS

wxEventType wxEVT_CHOICE

wxEventType wxEVT_CLOSE_WINDOW

wxEventType wxEVT_COMBOBOX

wxEventType wxEVT_COMBOBOX_CLOSEUP

wxEventType wxEVT_COMBOBOX_DROPDOWN

wxEventType wxEVT_COMMAND_ENTER

wxEventType wxEVT_COMMAND_KILL_FOCUS

wxEventType wxEVT_COMMAND_LEFT_CLICK

wxEventType wxEVT_COMMAND_LEFT_DCLICK

wxEventType wxEVT_COMMAND_RIGHT_CLICK

wxEventType wxEVT_COMMAND_RIGHT_DCLICK

wxEventType wxEVT_COMMAND_SET_FOCUS

wxEventType wxEVT_CONTEXT_MENU

wxEventType wxEVT_CREATE

wxEventType wxEVT_DESTROY

wxEventType wxEVT_DETAILED_HELP

wxEventType wxEVT_DISPLAY_CHANGED

wxEventType wxEVT_DROP_FILES

wxEventType wxEVT_END_SESSION

wxEventType wxEVT_ENTER_WINDOW

wxEventType wxEVT_ERASE_BACKGROUND

wxEventType wxEVT_HELP

wxEventType wxEVT_HIBERNATE

wxEventType wxEVT_HOTKEY

wxEventType wxEVT_ICONIZE

wxEventType wxEVT_IDLE

wxEventType wxEVT_INIT_DIALOG

wxEventType wxEVT_JOY_BUTTON_DOWN

wxEventType wxEVT_JOY_BUTTON_UP

wxEventType wxEVT_JOY_MOVE

wxEventType wxEVT_JOY_ZMOVE

wxEventType wxEVT_KEY_DOWN

wxEventType wxEVT_KEY_UP

wxEventType wxEVT_KILL_FOCUS

wxEventType wxEVT_LEAVE_WINDOW

wxEventType wxEVT_LEFT_DCLICK

wxEventType wxEVT_LEFT_DOWN

wxEventType wxEVT_LEFT_UP

wxEventType wxEVT_LISTBOX

wxEventType wxEVT_LISTBOX_DCLICK

wxEventType wxEVT_MAXIMIZE

wxEventType wxEVT_MENU

wxEventType wxEVT_MENU_CLOSE

wxEventType wxEVT_MENU_HIGHLIGHT

wxEventType wxEVT_MENU_OPEN

wxEventType wxEVT_MIDDLE_DCLICK

wxEventType wxEVT_MIDDLE_DOWN

wxEventType wxEVT_MIDDLE_UP

wxEventType wxEVT_MOTION

wxEventType wxEVT_MOUSE_CAPTURE_CHANGED

wxEventType wxEVT_MOUSE_CAPTURE_LOST

wxEventType wxEVT_MOUSEWHEEL

wxEventType wxEVT_MOVE

wxEventType wxEVT_MOVE_END

wxEventType wxEVT_MOVE_START

wxEventType wxEVT_MOVING

wxEventType wxEVT_NAVIGATION_KEY

wxEventType wxEVT_NC_PAINT

wxEventType wxEVT_NULL

A special event type usually used to indicate that some wxEvent has yet no type assigned.

wxEventType wxEVT_PAINT

wxEventType wxEVT_PALETTE_CHANGED

wxEventType wxEVT_QUERY_END_SESSION

wxEventType wxEVT_QUERY_NEW_PALETTE

wxEventType wxEVT_RADIOBOX

wxEventType wxEVT_RADIOBUTTON

wxEventType wxEVT_RIGHT_DCLICK

wxEventType wxEVT_RIGHT_DOWN

wxEventType wxEVT_RIGHT_UP

wxEventType wxEVT_SCROLL_BOTTOM

wxEventType wxEVT_SCROLL_CHANGED

wxEventType wxEVT_SCROLL_LINEDOWN

wxEventType wxEVT_SCROLL_LINEUP

wxEventType wxEVT_SCROLL_PAGEDOWN

wxEventType wxEVT_SCROLL_PAGEUP

wxEventType wxEVT_SCROLL_THUMBRELEASE

wxEventType wxEVT_SCROLL_THUMBTRACK

wxEventType wxEVT_SCROLL_TOP

wxEventType wxEVT_SCROLLBAR

wxEventType wxEVT_SCROLLWIN_BOTTOM

wxEventType wxEVT_SCROLLWIN_LINEDOWN

wxEventType wxEVT_SCROLLWIN_LINEUP

wxEventType wxEVT_SCROLLWIN_PAGEDOWN

wxEventType wxEVT_SCROLLWIN_PAGEUP

wxEventType wxEVT_SCROLLWIN_THUMBRELEASE

wxEventType wxEVT_SCROLLWIN_THUMBTRACK

wxEventType wxEVT_SCROLLWIN_TOP

wxEventType wxEVT_SET_CURSOR

wxEventType wxEVT_SET_FOCUS

wxEventType wxEVT_SHOW

wxEventType wxEVT_SIZE

wxEventType wxEVT_SIZING

wxEventType wxEVT_SLIDER

wxEventType wxEVT_SPIN

wxEventType wxEVT_SPIN_DOWN

wxEventType wxEVT_SPIN_UP

wxEventType wxEVT_SYS_COLOUR_CHANGED

wxEventType wxEVT_TEXT_COPY

wxEventType wxEVT_TEXT_CUT

wxEventType wxEVT_TEXT_PASTE

wxEventType wxEVT_THREAD

wxEventType wxEVT_TOOL

wxEventType wxEVT_TOOL_DROPDOWN

wxEventType wxEVT_TOOL_ENTER

wxEventType wxEVT_TOOL_RCLICKED

wxEventType wxEVT_UPDATE_UI

wxEventType wxEVT_VLBOX

wxEventType wxEVT_WINDOW_MODAL_DIALOG_CLOSED

20.22 File Handling

20.22.1 Detailed Description

wxWidgets has several small classes to work with disk files and directories.

Related Overviews: File Classes and Functions

Related macros/global-functions group: Files and Directories

Classes

- class wxDirTraverser

  wxDirTraverser is an abstract interface which must be implemented by objects passed to wxDir::Traverse() function.
- class wxDir

  wxDir is a portable equivalent of Unix open/read/closedir functions which allow enumerating of the files in a directory.
- class wxFFile

  wxFFile implements buffered file I/O.
- class wxTempFile

  wxTempFile provides a relatively safe way to replace the contents of the existing file.
- class wxFile

  A wxFile performs raw file I/O.
- class wxPathList

  The path list is a convenient way of storing a number of directories, and when presented with a filename without a directory, searching for an existing file in those directories.
- class wxFileName

  wxFileName encapsulates a file name.
- class wxFSFile

  This class represents a single file opened by wxFileSystem.
- class wxFileSystemWatcher

  The wxFileSystemWatcher class allows to receive notifications of file system changes.
- class wxStandardPaths

  wxStandardPaths returns the standard locations in the file system and should be used by applications to find their data files in a portable way.
- class wxTextFile

  The wxTextFile is a simple class which allows to work with text files on line by line basis.

20.23 Files and Directories

20.23.1 Detailed Description

These functions provide a platform-independent API for common file and directory functionality.

Related class group: File Handling

Macros

- #define wxCHANGE_UMASK(mask)

  Under Unix this macro changes the current process umask to the given value, unless it is equal to -1 in which case nothing is done, and restores it to the original value on scope exit.

Typedefs

- typedef off_t wxFileOffset

  The type used to store and provide byte offsets or byte sizes for files or streams.

Enumerations

- enum wxPosixPermissions {
  wxS_IRUSR = 00400,
  wxS_IWUSR = 00200,
  wxS_IXUSR = 00100,
  wxS_IRGRP = 00040,
  wxS_IWGRP = 00020,
  wxS_IXGRP = 00010,
  wxS_IROTH = 00004,
  wxS_IWOTH = 00002,
  wxS_IXOTH = 00001,
  wxPOSIX_USER_READ = wxS_IRUSR,
  wxPOSIX_USER_WRITE = wxS_IWUSR,
  wxPOSIX_USER_EXECUTE = wxS_IXUSR,
  wxPOSIX_GROUP_READ = wxS_IRGRP,
  wxPOSIX_GROUP_WRITE = wxS_IWGRP,
  wxPOSIX_GROUP_EXECUTE = wxS_IXGRP,
  wxPOSIX_OTHERS_READ = wxS_IROTH,
  wxPOSIX_OTHERS_WRITE = wxS_IWOTH,
  wxPOSIX_OTHERS_EXECUTE = wxS_IXOTH,
  wxS_DEFAULT,
  wxS_DIR_DEFAULT }

  File permission bit names.
- enum wxSeekMode {
  wxFromStart,
  wxFromCurrent,
  wxFromEnd }

  Parameter indicating how file offset should be interpreted.
- enum wxFileKind {
  wxFILE_KIND_UNKNOWN,
  wxFILE_KIND_DISK,
  wxFILE_KIND_TERMINAL,
  wxFILE_KIND_PIPE }

  File kind enumerations returned from wxGetFileKind().

Functions

- bool wxTransferFileToStream (const wxString &filename, ostream &stream)

  Copies the given file to stream.
- bool wxTransferStreamToFile (istream &stream, const wxString &filename)

  Copies the given stream to the file filename.
- bool wxGetDiskSpace (const wxString &path, wxLongLong total=NULL, wxLongLong free=NULL)

  This function returns the total number of bytes and number of free bytes on the disk containing the directory path (it should exist).
- wxString wxGetOSDirectory ()

  Returns the Windows directory under Windows; other platforms return an empty string.
- int wxParseCommonDialogsFilter (const wxString &wildCard, wxArrayString &descriptions, wxArrayString &filters)

  Parses the wildCard, returning the number of filters.
- void wxDos2UnixFilename (wxChar *s)

  Converts a DOS to a Unix filename by replacing backslashes with forward slashes.
- void wxUnix2DosFilename (wxChar *s)

  Converts a Unix to a DOS filename by replacing forward slashes with backslashes.
- bool wxDirExists (const wxString &dirname)

  Returns true if dirname exists and is a directory.
- void wxSplitPath (const wxString &fullname, wxString *path, wxString *name, wxString *ext)
- time_t wxFileModificationTime (const wxString &filename)

  Returns time of last modification of given file.
- bool wxRenameFile (const wxString &file1, const wxString &file2, bool overwrite=true)

  Renames file1 to file2, returning true if successful.
- bool wxCopyFile (const wxString &file1, const wxString &file2, bool overwrite=true)

  Copies file1 to file2, returning true if successful.
- bool wxFileExists (const wxString &filename)

  Returns true if the file exists and is a plain file.
- bool wxMatchWild (const wxString &pattern, const wxString &text, bool dot_special)

  Returns true if the pattern matches the text; if dot_special is true, filenames beginning with a dot are not matched with wildcard characters.
- wxString wxGetWorkingDirectory (char *buf=NULL, int sz=1000)
- wxString wxPathOnly (const wxString &path)

  Returns the directory part of the filename.
- bool wxIsWild (const wxString &pattern)

  Returns true if the pattern contains wildcards.
- bool wxIsAbsolutePath (const wxString &filename)

  Returns true if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.
- wxString wxGetCwd ()

  Returns a string containing the current (or working) directory.
- bool wxSetWorkingDirectory (const wxString &dir)

  Sets the current working directory, returning true if the operation succeeded.
- bool wxConcatFiles (const wxString &file1, const wxString &file2, const wxString &file3)

  Concatenates file1 and file2 to file3, returning true if successful.
- bool wxRemoveFile (const wxString &file)

  Removes file, returning true if successful.
- bool wxMkdir (const wxString &dir, int perm=wxS_DIR_DEFAULT)

  Makes the directory dir, returning true if successful.
- bool wxRmdir (const wxString &dir, int flags=0)

  Removes the directory dir, returning true if successful.
- wxString wxFindNextFile ()

Returns the next file that matches the path passed to wxFindFirstFile().

- wxString wxFindFirstFile (const wxString &spec, int flags=0)

  This function does directory searching; returns the first file that matches the path spec, or the empty string.
- wxFileKind wxGetFileKind (int fd)

  Returns the type of an open file.
- wxFileKind wxGetFileKind (FILE *fp)
- wxString wxFileNameFromPath (const wxString &path)
- char * wxFileNameFromPath (char *path)
- char * wxGetTempFileName (const wxString &prefix, char *buf=NULL)
- bool wxGetTempFileName (const wxString &prefix, wxString &buf)

Variables

- const int wxInvalidOffset = -1

  A special return value of many wxWidgets classes to indicate that an invalid offset was given.

20.23.2 Macro Definition Documentation

#define wxCHANGE_UMASK(*mask*)

Under Unix this macro changes the current process umask to the given value, unless it is equal to -1 in which case nothing is done, and restores it to the original value on scope exit.

It works by declaring a variable which sets umask to *mask* in its constructor and restores it in its destructor. Under other platforms this macro expands to nothing.

Include file:

```
#include <wx/filefn.h>
```

20.23.3 Typedef Documentation

typedef off_t wxFileOffset

The type used to store and provide byte offsets or byte sizes for files or streams.

This type is usually just a synonym for `off_t` but can be defined as `wxLongLong_t` if `wxHAS_HUGE_FILES` is defined but `off_t` is only 32 bits.

20.23.4 Enumeration Type Documentation

enum wxFileKind

File kind enumerations returned from wxGetFileKind().

Also used by wxFFile::GetKind() and wxFile::GetKind().

Include file:

```
#include <wx/filefn.h>
```

Enumerator

wxFILE_KIND_UNKNOWN Unknown file kind, or unable to determine.

wxFILE_KIND_DISK A file supporting seeking to arbitrary offsets.

wxFILE_KIND_TERMINAL A tty.

wxFILE_KIND_PIPE A pipe.

enum wxPosixPermissions

File permission bit names.

We define these constants in wxWidgets because S_IREAD &c are not standard. However, we do assume that the values correspond to the Unix umask bits.

Enumerator

wxS_IRUSR Standard POSIX names for these permission flags with "wx" prefix.

wxS_IWUSR Standard POSIX names for these permission flags with "wx" prefix.

wxS_IXUSR Standard POSIX names for these permission flags with "wx" prefix.

wxS_IRGRP Standard POSIX names for these permission flags with "wx" prefix.

wxS_IWGRP Standard POSIX names for these permission flags with "wx" prefix.

wxS_IXGRP Standard POSIX names for these permission flags with "wx" prefix.

wxS_IROTH Standard POSIX names for these permission flags with "wx" prefix.

wxS_IWOTH Standard POSIX names for these permission flags with "wx" prefix.

wxS_IXOTH Standard POSIX names for these permission flags with "wx" prefix.

wxPOSIX_USER_READ Longer but more readable synonyms for the constants above.

wxPOSIX_USER_WRITE Longer but more readable synonyms for the constants above.

wxPOSIX_USER_EXECUTE Longer but more readable synonyms for the constants above.

wxPOSIX_GROUP_READ Longer but more readable synonyms for the constants above.

wxPOSIX_GROUP_WRITE Longer but more readable synonyms for the constants above.

wxPOSIX_GROUP_EXECUTE Longer but more readable synonyms for the constants above.

wxPOSIX_OTHERS_READ Longer but more readable synonyms for the constants above.

wxPOSIX_OTHERS_WRITE Longer but more readable synonyms for the constants above.

wxPOSIX_OTHERS_EXECUTE Longer but more readable synonyms for the constants above.

wxS_DEFAULT Default mode for the new files: allow reading/writing them to everybody but the effective file mode will be set after ANDing this value with umask and so won't include wxS_IW{GRP,OTH} for the default 022 umask value.

wxS_DIR_DEFAULT Default mode for the new directories (see wxFileName::Mkdir): allow reading/writing/executing them to everybody, but just like wxS_DEFAULT the effective directory mode will be set after ANDing this value with umask.

enum wxSeekMode

Parameter indicating how file offset should be interpreted.

This is used by wxFFile::Seek() and wxFile::Seek().

Include file:

```
#include <wx/filefn.h>
```

Enumerator

wxFromStart Seek from the file beginning.

wxFromCurrent Seek from the current position.

wxFromEnd Seek from end of the file.

20.23.5 Function Documentation

bool wxConcatFiles (const wxString & *file1,* const wxString & *file2,* const wxString & *file3*)

Concatenates *file1* and *file2* to *file3*, returning true if successful.

Include file:

```
#include <wx/filefn.h>
```

bool wxCopyFile (const wxString & *file1,* const wxString & *file2,* bool *overwrite* = `true`)

Copies *file1* to *file2*, returning true if successful.

If *overwrite* parameter is true (default), the destination file is overwritten if it exists, but if *overwrite* is false, the functions fails in this case.

This function supports resources forks under Mac OS.

Include file:

```
#include <wx/filefn.h>
```

bool wxDirExists (const wxString & *dirname*)

Returns true if *dirname* exists and is a directory.

Include file:

```
#include <wx/filefn.h>
```

void wxDos2UnixFilename (wxChar * *s*)

Converts a DOS to a Unix filename by replacing backslashes with forward slashes.

Deprecated Construct a wxFileName with wxPATH_DOS and then use wxFileName::GetFullPath(wxPATH_UNIX) instead.

Include file:

```
#include <wx/filefn.h>
```

bool wxFileExists (const wxString & *filename*)

Returns true if the file exists and is a plain file.

Include file:

```
#include <wx/filefn.h>
```

time_t wxFileModificationTime (const wxString & *filename*)

Returns time of last modification of given file.

The function returns `(time_t)-1` if an error occurred (e.g. file not found).

Include file:

```
#include <wx/filefn.h>
```

wxString wxFileNameFromPath (const wxString & *path*)

Deprecated This function is obsolete, please use wxFileName::SplitPath() instead.

Returns the filename for a full path. The second form returns a pointer to temporary storage that should not be deallocated.

Include file:

```
#include <wx/filefn.h>
```

char* wxFileNameFromPath (char * *path*)

wxString wxFindFirstFile (const wxString & *spec*, int *flags* = 0)

This function does directory searching; returns the first file that matches the path *spec*, or the empty string.

Use wxFindNextFile() to get the next matching file. Neither will report the current directory "." or the parent directory "..".

Warning

> As of 2.5.2, these functions are not thread-safe! (they use static variables). You probably want to use wxDir::GetFirst() or wxDirTraverser instead.

spec may contain wildcards.

flags may be wxDIR for restricting the query to directories, wxFILE for files or zero for either.

For example:

```
1 wxString f = wxFindFirstFile("/home/project/*.*");
2 while ( !f.empty() )
3 {
4     ...
5     f = wxFindNextFile();
6 }
```

Include file:

```
#include <wx/filefn.h>
```

wxString wxFindNextFile ( )

Returns the next file that matches the path passed to wxFindFirstFile().

See wxFindFirstFile() for an example.

Include file:

```
#include <wx/filefn.h>
```

wxString wxGetCwd ( )

Returns a string containing the current (or working) directory.

Include file:

```
#include <wx/filefn.h>
```

bool wxGetDiskSpace (const wxString & *path*, wxLongLong *total* = NULL, wxLongLong *free* = NULL)

This function returns the total number of bytes and number of free bytes on the disk containing the directory *path* (it should exist).

Both *total* and *free* parameters may be NULL if the corresponding information is not needed.

Since

2.3.2

Note

The generic Unix implementation depends on the system having the `statfs()` or `statvfs()` function.

Returns

true on success, false if an error occurred (for example, the directory doesn't exist).

Include file:

```
#include <wx/filefn.h>
```

wxFileKind wxGetFileKind (int *fd*)

Returns the type of an open file.

Possible return values are enumerations of wxFileKind.

Include file:

```
#include <wx/filefn.h>
```

wxFileKind wxGetFileKind (FILE * *fp*)

wxString wxGetOSDirectory ( )

Returns the Windows directory under Windows; other platforms return an empty string.

Include file:

```
#include <wx/filefn.h>
```

char* wxGetTempFileName (const wxString & *prefix*, char * *buf* = NULL)

Deprecated This function is obsolete, please use wxFileName::CreateTempFileName() instead.

Include file:

```
#include <wx/filefn.h>
```

bool wxGetTempFileName (const wxString & *prefix,* wxString & *buf*)

wxString wxGetWorkingDirectory (char * *buf* = NULL, int *sz* = 1000)

Deprecated This function is deprecated, use wxGetCwd() instead.

Copies the current working directory into the buffer if supplied, or copies the working directory into new storage (which you must delete yourself) if the buffer is NULL.

sz is the size of the buffer if supplied.

Include file:

```
#include <wx/filefn.h>
```

bool wxIsAbsolutePath (const wxString & *filename*)

Returns true if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.

Include file:

```
#include <wx/filefn.h>
```

bool wxIsWild (const wxString & *pattern*)

Returns true if the pattern contains wildcards.

See also

wxMatchWild()

Include file:

```
#include <wx/filefn.h>
```

bool wxMatchWild (const wxString & *pattern,* const wxString & *text,* bool *dot_special*)

Returns true if the *pattern* matches the *text*; if *dot_special* is true, filenames beginning with a dot are not matched with wildcard characters.

See also

wxIsWild()

Include file:

```
#include <wx/filefn.h>
```

bool wxMkdir (const wxString & *dir,* int *perm* = wxS_DIR_DEFAULT)

Makes the directory *dir*, returning true if successful.

perm is the access mask for the directory for the systems on which it is supported (Unix) and doesn't have any effect on the other ones.

Include file:

```
#include <wx/filefn.h>
```

int wxParseCommonDialogsFilter (const wxString & *wildCard,* wxArrayString & *descriptions,* wxArrayString & *filters*)

Parses the *wildCard*, returning the number of filters.

Returns 0 if none or if there's a problem.

The arrays will contain an equal number of items found before the error. On platforms where native dialogs handle only one filter per entry, entries in arrays are automatically adjusted. *wildCard* is in the form:

```
1 "All files (*)|*|Image Files (*.jpeg *.png)|*.jpg;*.png"
```

Include file:

```
#include <wx/filefn.h>
```

wxString wxPathOnly (const wxString & *path*)

Returns the directory part of the filename.

Include file:

```
#include <wx/filefn.h>
```

bool wxRemoveFile (const wxString & *file*)

Removes *file*, returning true if successful.

Include file:

```
#include <wx/filefn.h>
```

bool wxRenameFile (const wxString & *file1,* const wxString & *file2,* bool *overwrite* = `true`)

Renames *file1* to *file2*, returning true if successful.

If *file2* is a directory, *file1* is moved into it (*overwrite* is ignored in this case). Otherwise, if *file2* is an existing file, it is overwritten if *overwrite* is true (default) and the function fails if *overwrite* is false.

Include file:

```
#include <wx/filefn.h>
```

bool wxRmdir (const wxString & *dir,* int *flags* = `0`)

Removes the directory *dir*, returning true if successful.

Does not work under VMS.

The *flags* parameter is reserved for future use.

Note

There is also a wxRmDir() function which simply wraps the standard POSIX `rmdir()` function and so return an integer error code instead of a boolean value (but otherwise is currently identical to wxRmdir()), don't confuse these two functions.

Include file:

```
#include <wx/filefn.h>
```

bool wxSetWorkingDirectory (const wxString & *dir*)

Sets the current working directory, returning true if the operation succeeded.

Under MS Windows, the current drive is also changed if *dir* contains a drive specification.

Include file:

```
#include <wx/filefn.h>
```

void wxSplitPath (const wxString & *fullname,* wxString * *path,* wxString * *name,* wxString * *ext*)

Deprecated This function is obsolete, please use wxFileName::SplitPath() instead.

This function splits a full file name into components: the path (including possible disk/drive specification under Windows), the base name, and the extension. Any of the output parameters (*path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component.

wxSplitPath() will correctly handle filenames with both DOS and Unix path separators under Windows, however it will not consider backslashes as path separators under Unix (where backslash is a valid character in a filename).

On entry, *fullname* should be non-NULL (it may be empty though).

On return, *path* contains the file path (without the trailing separator), *name* contains the file name and `ext` contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

Include file:

```
#include <wx/filefn.h>
```

bool wxTransferFileToStream (const wxString & *filename,* ostream & *stream*)

Copies the given file to *stream*.

Useful when converting an old application to use streams (within the document/view framework, for example).

Include file:

```
#include <wx/docview.h>
```

bool wxTransferStreamToFile (istream & *stream,* const wxString & *filename*)

Copies the given stream to the file *filename*.

Useful when converting an old application to use streams (within the document/view framework, for example).

Include file:

```
#include <wx/docview.h>
```

void wxUnix2DosFilename (wxChar * *s*)

Converts a Unix to a DOS filename by replacing forward slashes with backslashes.

Deprecated Construct a wxFileName with wxPATH_UNIX and then use wxFileName::GetFullPath(wxPATH_DOS) instead.

Include file:

```
#include <wx/filefn.h>
```

20.23.6 Variable Documentation

const int wxInvalidOffset = -1

A special return value of many wxWidgets classes to indicate that an invalid offset was given.

20.24 Functions and Macros by Category

20.24.1 Detailed Description

This group defines all major function and macro groups.

Modules

- Application Initialization and Termination

  The functions in this section are used on application startup/shutdown and also to control the behaviour of the main event loop of the GUI programs.
- Atomic Operations

  When using multi-threaded applications, it is often required to access or modify memory which is shared between threads.
- Byte Order

  The endian-ness issues (that is the difference between big-endian and little-endian architectures) are important for the portable programs working with the external binary data (for example, data files or data coming from network) which is usually in some fixed, platform-independent format.
- Debugging macros

  Useful macros and functions for error checking and defensive programming.
- Dialogs

  Below are a number of convenience functions for getting input from the user or displaying messages.
- Environment

  These functions allow access to get or change the values of environment variables in a portable way.
- Events

  Below are a number of functions/macros used with wxWidgets event-handling system.
- Files and Directories

  These functions provide a platform-independent API for common file and directory functionality.
- Graphics Device Interface (GDI)

  The following are functions and macros related to GDI (Graphics Device Interface) access.
- Locale-dependent functions

  Below are a number of functions/macros which accept as last parameter a specific wxXLocale instance.
- Logging

  These functions provide a variety of logging functions.
- Math

  The functions in this section are typically related with math operations and floating point numbers.
- Miscellaneous

  Group of miscellaneous functions and macros.
- Network, User and OS

  The functions in this section are used to retrieve information about the current computer and/or user characteristics.
- Process Control

  The functions in this section are used to launch or terminate the other processes.
- Runtime Type Information (RTTI)

  wxWidgets uses its own RTTI ("run-time type identification") system which predates the current standard C++ RTTI and so is kept for backwards compatibility reasons but also because it allows some things which the standard RTTI doesn't directly support (such as creating a class from its name).
- Strings

  Global string functions and macros.
- Threads

  The functions and macros here mainly exist to make it possible to write code which may be compiled in multi thread build (wxUSE_THREADS = 1) as well as in single thread configuration (wxUSE_THREADS = 0).
- Time

The functions in this section deal with getting the current time and sleeping for the specified time interval.

- Versioning

  The following constants are defined in wxWidgets:

- Wrappers of CRT functions

  For documentation of these functions please refer to the documentation of the standard CRT functions (see e.g. http://www.cppreference.com/wiki/c/start*).*

20.25 Graphics Device Interface (GDI)

20.25.1 Detailed Description

The following are classes related to GDI (Graphics Device Interface) access.

They provide an API for drawing on device contexts, windows, and printing.

Related Overviews: Device Contexts, Bitmaps and Icons

Related macros/global-functions group: Graphics Device Interface (GDI)

Classes

- struct wxFontMetrics

  Simple collection of various font metrics.
- class wxGraphicsGradientStop

  Represents a single gradient stop in a collection of gradient stops as represented by wxGraphicsGradientStops.
- class wxGraphicsGradientStops

  Represents a collection of wxGraphicGradientStop values for use with CreateLinearGradientBrush and Create↩ RadialGradientBrush.
- class wxAnimation

  This class encapsulates the concept of a platform-dependent animation.
- class wxBitmapHandler

  This is the base class for implementing bitmap file loading/saving, and bitmap creation from data.
- class wxBitmap

  This class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour or colour with alpha channel support.
- class wxMask

  This class encapsulates a monochrome mask bitmap, where the masked area is black and the unmasked area is white.
- class wxBrush

  A brush is a drawing tool for filling in areas.
- class wxBrushList

  A brush list is a list containing all brushes which have been created.
- class wxColour

  A colour is an object representing a combination of Red, Green, and Blue (RGB) intensity values, and is used to determine drawing colours.
- class wxCursor

  A cursor is a small bitmap usually used for denoting where the mouse pointer is, with a picture that might indicate the interpretation of a mouse click.
- class wxDC

  A wxDC is a "device context" onto which graphics and text can be drawn.
- class wxDCClipper

  wxDCClipper is a helper class for setting a clipping region on a wxDC during its lifetime.
- class wxDCBrushChanger

  wxDCBrushChanger is a small helper class for setting a brush on a wxDC and unsetting it automatically in the destructor, restoring the previous one.
- class wxDCPenChanger

  wxDCPenChanger is a small helper class for setting a pen on a wxDC and unsetting it automatically in the destructor, restoring the previous one.
- class wxDCTextColourChanger

  wxDCTextColourChanger is a small helper class for setting a foreground text colour on a wxDC and unsetting it automatically in the destructor, restoring the previous one.

- class wxDCFontChanger

  wxDCFontChanger is a small helper class for setting a font on a wxDC and unsetting it automatically in the destructor, restoring the previous one.
- class wxFont

  A font is an object which determines the appearance of text.
- class wxFontList

  A font list is a list containing all fonts which have been created.
- class wxFontEnumerator

  wxFontEnumerator enumerates either all available fonts on the system or only the ones with given attributes - either only fixed-width (suited for use in programs such as terminal emulators and the like) or the fonts available in the given encoding).
- class wxNativeFontInfo

  wxNativeFontInfo is platform-specific font representation: this class should be considered as an opaque font description only used by the native functions, the user code can only get the objects of this type from somewhere and pass it somewhere else (possibly save them somewhere using ToString() and restore them using FromString())
- class wxColourDatabase

  wxWidgets maintains a database of standard RGB colours for a predefined set of named colours.
- class wxGDIObject

  This class allows platforms to implement functionality to optimise GDI objects, such as wxPen, wxBrush and wxFont.
- class wxGraphicsPath

  A wxGraphicsPath is a native representation of a geometric path.
- class wxGraphicsObject

  This class is the superclass of native graphics objects like pens etc.
- class wxGraphicsContext

  A wxGraphicsContext instance is the object that is drawn upon.
- class wxGraphicsRenderer

  A wxGraphicsRenderer is the instance corresponding to the rendering engine used.
- class wxGraphicsBrush

  A wxGraphicsBrush is a native representation of a brush.
- class wxGraphicsFont

  A wxGraphicsFont is a native representation of a font.
- class wxGraphicsPen

  A wxGraphicsPen is a native representation of a pen.
- class wxGraphicsMatrix

  A wxGraphicsMatrix is a native representation of an affine matrix.
- class wxIcon

  An icon is a small rectangular bitmap usually used for denoting a minimized application.
- class wxIconBundle

  This class contains multiple copies of an icon in different sizes.
- class wxIconLocation

  wxIconLocation is a tiny class describing the location of an (external, i.e.
- class wxImageHandler

  This is the base class for implementing image file loading/saving, and image creation from data.
- class wxImage

  This class encapsulates a platform-independent image.
- class wxImageList

  A wxImageList contains a list of images, which are stored in an unspecified form.
- class wxMetafile

  *A **wxMetafile** represents the MS Windows metafile object, so metafile operations have no effect in X.*
- class wxPalette

  A palette is a table that maps pixel values to RGB colours.
- class wxPen

A pen is a drawing tool for drawing outlines.

- class wxPenList

  There is only one instance of this class: wxThePenList.

- class wxPixelData< Image, PixelFormat >

  A class template with ready to use implementations for getting direct and efficient access to wxBitmap's internal data and wxImage's internal data through a standard interface.

- class wxRegionIterator

  This class is used to iterate through the rectangles in a region, typically when examining the damaged regions of a window within an OnPaint call.

- class wxRegion

  A wxRegion represents a simple or complex region on a device context or window.

- struct wxSplitterRenderParams

  This is just a simple `struct` used as a return value of wxRendererNative::GetSplitterParams().

- struct wxHeaderButtonParams

  This `struct` can optionally be used with wxRendererNative::DrawHeaderButton() to specify custom values used to draw the text or bitmap label.

- class wxDelegateRendererNative

  wxDelegateRendererNative allows reuse of renderers code by forwarding all the wxRendererNative methods to the given object and thus allowing you to only modify some of its methods – without having to reimplement all of them.

- class wxRendererNative

  First, a brief introduction to wxRendererNative and why it is needed.

- struct wxRendererVersion

  This simple struct represents the wxRendererNative interface version and is only used as the return value of wxRendererNative::GetVersion().

- class wxTextWrapper

  Helps wrap lines of text to given width.

20.26 Graphics Device Interface (GDI)

20.26.1 Detailed Description

The following are functions and macros related to GDI (Graphics Device Interface) access.

Related Overviews: Device Contexts

Related class group: Graphics Device Interface (GDI)

Macros

- #define wxDROP_ICON(name)

  This macro creates either a cursor (MSW) or an icon (elsewhere) with the given name (of type `const char*`).
- #define wxBITMAP(bitmapName)

  This macro loads a bitmap from either application resources (on the platforms for which they exist, i.e. Windows) or from an XPM file.
- #define wxBITMAP_PNG(bitmapName)

  Creates a bitmap from either application resources or embedded image data in PNG format.
- #define wxBITMAP_PNG_FROM_DATA(bitmapName)

  Creates a bitmap from embedded image data in PNG format.
- #define wxICON(iconName)

  This macro loads an icon from either application resources (on the platforms for which they exist, i.e. Windows) or from an XPM file.

Functions

- bool wxIsDragResultOk (wxDragResult res)

  Returns true if res indicates that something was done during a DnD operation, i.e.
- bool wxColourDisplay ()

  Returns true if the display is colour, false otherwise.
- int wxDisplayDepth ()

  Returns the depth of the display (a value of 1 denotes a monochrome display).
- void wxSetCursor (const wxCursor &cursor)

  Globally sets the cursor; only has an effect on Windows, Mac and GTK+.
- void wxClientDisplayRect (int *x, int *y, int *width, int *height)

  Returns the dimensions of the work area on the display.
- wxRect wxGetClientDisplayRect ()

  Returns the dimensions of the work area on the display.
- wxSize wxGetDisplayPPI ()

  Returns the display resolution in pixels per inch.
- void wxDisplaySize (int *width, int *height)

  Returns the display size in pixels.
- wxSize wxGetDisplaySize ()

  Returns the display size in pixels.
- void wxDisplaySizeMM (int *width, int *height)

  Returns the display size in millimeters.
- wxSize wxGetDisplaySizeMM ()

  Returns the display size in millimeters.
- bool wxMakeMetafilePlaceable (const wxString &filename, int minX, int minY, int maxX, int maxY, float scale=1.0)

  Given a filename for an existing, valid metafile (as constructed using wxMetafileDC) makes it into a placeable metafile by prepending a header containing the given bounding box.

20.26.2 Macro Definition Documentation

#define wxBITMAP(*bitmapName*)

This macro loads a bitmap from either application resources (on the platforms for which they exist, i.e. Windows) or from an XPM file.

This can help to avoid using #ifdef when creating bitmaps.

See also

Bitmaps and Icons, wxICON()

Include file:

```
#include <wx/gdicmn.h>
```

#define wxBITMAP_PNG(*bitmapName*)

Creates a bitmap from either application resources or embedded image data in PNG format.

This macro is similar to wxBITMAP() but works with bitmap data in PNG format and not BMP or XPM.

Under Windows the given *bitmapName* must be present in the application resource file with the type `RCDATA` and refer to a PNG image. I.e. you should have a definition similar to the following in your .rc file:

```
1 mybitmap    RCDATA  "mybitmap.png"
```

to be able to use `wxBITMAP_PNG(mybitmap)` in the code.

Under OS X the file with the specified name and "png" extension must be present in the "Resources" subdirectory of the application bundle.

Under the other platforms, this is equivalent to wxBITMAP_PNG_FROM_DATA() and so loads the image data from the array called `bitmapName_png` that must exist. Notice that it *must* be an array and not a pointer as the macro needs to be able to determine its size. Such an array can be produced by a number of conversion programs. A very simple one is included in wxWidgets distribution as `misc/scripts/png2c.py`.

Finally notice that you must register PNG image handler to be able to load bitmaps from PNG data. This can be done either by calling wxInitAllImageHandlers() which also registers all the other image formats or including the necessary header:

```
1 #include <wx/imagpng.h>
```

and calling

```
1 wxImage::AddHandler(new wxPNGHandler);
```

in your application startup code.

See also

wxBITMAP_PNG_FROM_DATA()

Include file:

```
#include <wx/gdicmn.h>
```

Since

2.9.5

#define wxBITMAP_PNG_FROM_DATA(*bitmapName*)

Creates a bitmap from embedded image data in PNG format.

This macro is a thin wrapper around wxBitmap::NewFromPNGData() and takes just the base name of the array containing the image data and computes its size internally. In other words, the array called `bitmapName_png` must exist. Notice that it *must* be an array and not a pointer as the macro needs to be able to determine its size. Such an array can be produced by a number of conversion programs. A very simple one is included in wxWidgets distribution as `misc/scripts/png2c.py`.

You can use wxBITMAP_PNG() to load the PNG bitmaps from resources on the platforms that support this and only fall back to loading them from data under the other ones (i.e. not Windows and not OS X).

Include file:

```
#include <wx/gdicmn.h>
```

Since

2.9.5

#define wxDROP_ICON(*name*)

This macro creates either a cursor (MSW) or an icon (elsewhere) with the given *name* (of type `const char*`).

Under MSW, the cursor is loaded from the resource file and the icon is loaded from XPM file under other platforms.

This macro should be used with wxDropSource::wxDropSource().

Returns

wxCursor on MSW, otherwise returns a wxIcon

Include file:

```
#include <wx/dnd.h>
```

#define wxICON(*iconName*)

This macro loads an icon from either application resources (on the platforms for which they exist, i.e. Windows) or from an XPM file.

This can help to avoid using #ifdef when creating icons.

See also

Bitmaps and Icons, wxBITMAP()

Include file:

```
#include <wx/gdicmn.h>
```

20.26.3 Function Documentation

void wxClientDisplayRect (int * *x*, int * *y*, int * *width*, int * *height*)

Returns the dimensions of the work area on the display.

This is the same as wxGetClientDisplayRect() but allows to retrieve the individual components instead of the entire rectangle.

Any of the output pointers can be NULL if the corresponding value is not needed by the caller.

See also

wxDisplay

Include file:

```
#include <wx/gdicmn.h>
```

bool wxColourDisplay ( )

Returns true if the display is colour, false otherwise.
Include file:

```
#include <wx/gdicmn.h>
```

int wxDisplayDepth ( )

Returns the depth of the display (a value of 1 denotes a monochrome display).
Include file:

```
#include <wx/gdicmn.h>
```

void wxDisplaySize (int * *width*, int * *height*)

Returns the display size in pixels.

Either of output pointers can be NULL if the caller is not interested in the corresponding value.

See also

wxGetDisplaySize(), wxDisplay

Include file:

```
#include <wx/gdicmn.h>
```

void wxDisplaySizeMM (int * *width*, int * *height*)

Returns the display size in millimeters.

Either of output pointers can be NULL if the caller is not interested in the corresponding value.

See also

wxGetDisplaySizeMM(), wxDisplay

Include file:

```
#include <wx/gdicmn.h>
```

wxRect wxGetClientDisplayRect ( )

Returns the dimensions of the work area on the display.

On Windows this means the area not covered by the taskbar, etc. Other platforms are currently defaulting to the whole display until a way is found to provide this info for all window managers, etc.

See also

wxDisplay

Include file:

```
#include <wx/gdicmn.h>
```

wxSize wxGetDisplayPPI ( )

Returns the display resolution in pixels per inch.

The `x` component of the returned wxSize object contains the horizontal resolution and the `y` one – the vertical resolution.

Include file:

```
#include <wx/gdicmn.h>
```

See also

wxDisplay

Since

2.9.0

wxSize wxGetDisplaySize ( )

Returns the display size in pixels.

See also

wxDisplay

Include file:

```
#include <wx/gdicmn.h>
```

wxSize wxGetDisplaySizeMM ( )

Returns the display size in millimeters.

See also

wxDisplay

Include file:

```
#include <wx/gdicmn.h>
```

bool wxIsDragResultOk (wxDragResult *res*)

Returns true if res indicates that something was done during a DnD operation, i.e.

is neither error nor none nor cancel.

bool wxMakeMetafilePlaceable (const wxString & *filename,* int *minX,* int *minY,* int *maxX,* int *maxY,* float *scale* = 1.0)

Given a filename for an existing, valid metafile (as constructed using wxMetafileDC) makes it into a placeable metafile by prepending a header containing the given bounding box.

The bounding box may be obtained from a device context after drawing into it, using the functions wxDC::MinX(), wxDC::MinY(), wxDC::MaxX() and wxDC::MaxY().

In addition to adding the placeable metafile header, this function adds the equivalent of the following code to the start of the metafile data:

```
1 SetMapMode(dc, MM_ANISOTROPIC);
2 SetWindowOrg(dc, minX, minY);
3 SetWindowExt(dc, maxX - minX, maxY - minY);
```

This simulates the wxMM_TEXT mapping mode, which wxWidgets assumes.

Placeable metafiles may be imported by many Windows applications, and can be used in RTF (Rich Text Format) files.

scale allows the specification of scale for the metafile.

This function is only available under Windows.

Include file:

```
#include <wx/metafile.h>
```

void wxSetCursor (const wxCursor & *cursor*)

Globally sets the cursor; only has an effect on Windows, Mac and GTK+.

You should call this function with wxNullCursor to restore the system cursor.

See also

wxCursor, wxWindow::SetCursor()

Include file:

```
#include <wx/gdicmn.h>
```

20.27 Grid Related Classes

20.27.1 Detailed Description

Classes related to the wxGrid generic widget.

Classes

- class wxGridCellRenderer

  This class is responsible for actually drawing the cell in the grid.
- class wxGridCellAutoWrapStringRenderer

  This class may be used to format string data in a cell.
- class wxGridCellBoolRenderer

  This class may be used to format boolean data in a cell.
- class wxGridCellDateTimeRenderer

  This class may be used to format a date/time data in a cell.
- class wxGridCellEnumRenderer

  This class may be used to render in a cell a number as a textual equivalent.
- class wxGridCellFloatRenderer

  This class may be used to format floating point data in a cell.
- class wxGridCellNumberRenderer

  This class may be used to format integer data in a cell.
- class wxGridCellStringRenderer

  This class may be used to format string data in a cell; it is the default for string cells.
- class wxGridCellEditor

  This class is responsible for providing and manipulating the in-place edit controls for the grid.
- class wxGridCellAutoWrapStringEditor

  Grid cell editor for wrappable string/text data.
- class wxGridCellBoolEditor

  Grid cell editor for boolean data.
- class wxGridCellChoiceEditor

  Grid cell editor for string data providing the user a choice from a list of strings.
- class wxGridCellEnumEditor

  Grid cell editor which displays an enum number as a textual equivalent (eg.
- class wxGridCellTextEditor

  Grid cell editor for string/text data.
- class wxGridCellFloatEditor

  The editor for floating point numbers data.
- class wxGridCellNumberEditor

  Grid cell editor for numeric integer data.
- class wxGridCellAttr

  This class can be used to alter the cells' appearance in the grid by changing their attributes from the defaults.
- class wxGridTableBase

  The almost abstract base class for grid tables.
- class wxGridTableMessage

  A simple class used to pass messages from the table to the grid.
- class wxGridSizesInfo

  wxGridSizesInfo stores information about sizes of all wxGrid rows or columns.
- class wxGrid

  wxGrid and its related classes are used for displaying and editing tabular data.

- class wxGridUpdateLocker

  This small class can be used to prevent wxGrid from redrawing during its lifetime by calling wxGrid::BeginBatch() in its constructor and wxGrid::EndBatch() in its destructor.
- class wxGridEvent

  This event class contains information about various grid events.
- class wxGridSizeEvent

  This event class contains information about a row/column resize event.
- class wxGridRangeSelectEvent
- class wxGridEditorCreatedEvent

20.28 HTML

20.28.1 Detailed Description

wxWidgets provides a set of classes to display text in HTML format.

These classes include a help system based on the HTML widget.

Classes

- class wxHtmlHelpController

  This help controller provides an easy way of displaying HTML help in your application (see HTML Sample, test example).
- class wxHtmlModalHelp

  This class uses wxHtmlHelpController to display help in a modal dialog.
- class wxHtmlHelpData

  This class is used by wxHtmlHelpController and wxHtmlHelpFrame to access HTML help items.
- class wxHtmlHelpDialog

  This class is used by wxHtmlHelpController to display help.
- class wxHtmlHelpFrame

  This class is used by wxHtmlHelpController to display help.
- class wxHtmlHelpWindow

  This class is used by wxHtmlHelpController to display help within a frame or dialog, but you can use it yourself to create an embedded HTML help window.
- class wxHtmlRenderingStyle

  wxHtmlSelection is data holder with information about text selection.
- class wxHtmlRenderingState

  Selection state is passed to wxHtmlCell::Draw so that it can render itself differently e.g.
- class wxHtmlRenderingInfo

  This class contains information given to cells when drawing them.
- class wxHtmlCell

  Internal data structure.
- class wxHtmlContainerCell

  The wxHtmlContainerCell class is an implementation of a cell that may contain more cells in it.
- class wxHtmlLinkInfo

  This class stores all necessary information about hypertext links (as represented by <A> tag in HTML documents).
- class wxHtmlColourCell

  This cell changes the colour of either the background or the foreground.
- class wxHtmlWidgetCell

  wxHtmlWidgetCell is a class that provides a connection between HTML cells and widgets (an object derived from wxWindow).
- class wxHtmlWordCell

  This html cell represents a single word or text fragment in the document stream.
- class wxHtmlWordWithTabsCell

  wxHtmlWordCell is a specialization for storing text fragments with embedded tab characters.
- class wxHtmlFontCell

  This cell represents a font change in the document stream.
- class wxHtmlFilter

  This class is the parent class of input filters for wxHtmlWindow.
- class wxHtmlTagHandler
- class wxHtmlParser

*Classes derived from this handle the **generic** parsing of HTML documents: it scans the document and divide it into blocks of tags (where one block consists of beginning and ending tag and of text between these two tags).*

- class wxHtmlTag

  This class represents a single HTML tag.

- class wxHtmlWindow

  wxHtmlWindow is probably the only class you will directly use unless you want to do something special (like adding new tag handlers or MIME filters).

- class wxHtmlLinkEvent

  This event class is used for the events generated by wxHtmlWindow.

- class wxHtmlCellEvent

  This event class is used for the events generated by wxHtmlWindow.

- class wxHtmlDCRenderer

  This class can render HTML document into a specified area of a DC.

- class wxHtmlEasyPrinting

  This class provides very simple interface to printing architecture.

- class wxHtmlPrintout

  This class serves as printout class for HTML documents.

- class wxHtmlTagsModule

  This class provides easy way of filling wxHtmlWinParser's table of tag handlers.

- class wxHtmlWinTagHandler

  This is basically wxHtmlTagHandler except that it is extended with protected member m_WParser pointing to the wxHtmlWinParser object (value of this member is identical to wxHtmlParser's m_Parser).

- class wxHtmlWinParser

  This class is derived from wxHtmlParser and its main goal is to parse HTML input so that it can be displayed in wxHtmlWindow.

20.29 Help

20.29.1 Detailed Description

Classes for loading and displaying help manuals or help informations in general.

Classes

- class wxHelpProvider

  wxHelpProvider is an abstract class used by a program implementing context-sensitive help to show the help text for the given window.
- class wxHelpControllerHelpProvider

  wxHelpControllerHelpProvider is an implementation of wxHelpProvider which supports both context identifiers and plain text help strings.
- class wxContextHelp

  This class changes the cursor to a query and puts the application into a 'context-sensitive help mode'.
- class wxContextHelpButton

  Instances of this class may be used to add a question mark button that when pressed, puts the application into context-help mode.
- class wxSimpleHelpProvider

  wxSimpleHelpProvider is an implementation of wxHelpProvider which supports only plain text help strings, and shows the string associated with the control (if any) in a tooltip.
- class wxExtHelpController

  This class implements help via an external browser.
- class wxHelpControllerBase

  This is the abstract base class a family of classes by which applications may invoke a help viewer to provide on-line help.
- class wxHelpController

  This is an alias for one of a family of help controller classes which is most appropriate for the current platform.
- class wxHtmlHelpController

  This help controller provides an easy way of displaying HTML help in your application (see HTML Sample, test example).
- class wxHtmlModalHelp

  This class uses wxHtmlHelpController to display help in a modal dialog.
- class wxHtmlHelpData

  This class is used by wxHtmlHelpController and wxHtmlHelpFrame to access HTML help items.
- class wxHtmlHelpDialog

  This class is used by wxHtmlHelpController to display help.
- class wxHtmlHelpFrame

  This class is used by wxHtmlHelpController to display help.
- class wxHtmlHelpWindow

  This class is used by wxHtmlHelpController to display help within a frame or dialog, but you can use it yourself to create an embedded HTML help window.
- class wxToolTip

  This class holds information about a tooltip associated with a window (see wxWindow::SetToolTip()).

20.30 Interprocess Communication

20.30.1 Detailed Description

wxWidgets provides simple interprocess communications facilities based on Windows DDE, but they are available on most platforms using TCP.

Related Overviews: Interprocess Communication

Classes

- class wxDDEConnection

  A wxDDEConnection object represents the connection between a client and a server.
- class wxDDEClient

  A wxDDEClient object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation.
- class wxDDEServer

  A wxDDEServer object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation.
- class wxConnection

  A wxConnection object represents the connection between a client and a server.
- class wxClient

  A wxClient object represents the client part of a client-server DDE-like (Dynamic Data Exchange) conversation.
- class wxServer

  A wxServer object represents the server part of a client-server DDE-like (Dynamic Data Exchange) conversation.
- class wxConnectionBase
- class wxActiveXContainer

  wxActiveXContainer is a host for an ActiveX control on Windows (and as such is a platform-specific class).

20.31 Locale-dependent functions

20.31.1 Detailed Description

Below are a number of functions/macros which accept as last parameter a specific wxXLocale instance.

For the documentation of function `wxFunc_l()`, please see the documentation of the standard `Func()` function (see e.g. `http://www.cppreference.com/wiki/c/string/start`) and keep in mind that the wxWidgets function takes as last parameter the locale which should be internally used for locale-dependent operations.

Last, note that when the `wxHAS_XLOCALE_SUPPORT` symbol is not defined, then wxWidgets will provide implementations of these functions itself and that they are not granted to be thread-safe (and they will work only with the C locale; see Availability).

Functions

- int wxIsalnum_l (wchar_t c, const wxXLocale &loc)
- int wxIsalpha_l (wchar_t c, const wxXLocale &loc)
- int wxIscntrl_l (wchar_t c, const wxXLocale &loc)
- int wxIsdigit_l (wchar_t c, const wxXLocale &loc)
- int wxIsgraph_l (wchar_t c, const wxXLocale &loc)
- int wxIslower_l (wchar_t c, const wxXLocale &loc)
- int wxIsprint_l (wchar_t c, const wxXLocale &loc)
- int wxIspunct_l (wchar_t c, const wxXLocale &loc)
- int wxIsspace_l (wchar_t c, const wxXLocale &loc)
- int wxIsupper_l (wchar_t c, const wxXLocale &loc)
- int wxIsxdigit_l (wchar_t c, const wxXLocale &loc)
- wchar_t wxTolower_l (wchar_t c, const wxXLocale &loc)
- wchar_t wxToupper_l (wchar_t c, const wxXLocale &loc)
- double wxStrtod_l (const wchar_t *c, wchar_t **endptr, const wxXLocale &loc)
- long wxStrtol_l (const wchar_t *c, wchar_t **endptr, int base, const wxXLocale &loc)
- unsigned long wxStrtoul_l (const wchar_t *c, wchar_t **endptr, int base, const wxXLocale &loc)

20.31.2 Function Documentation

int wxIsalnum_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsalpha_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIscntrl_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsdigit_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsgraph_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIslower_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsprint_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIspunct_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsspace_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsupper_l (wchar_t *c,* const wxXLocale & *loc*)

int wxIsxdigit_l (wchar_t *c,* const **wxXLocale** & *loc*)

double wxStrtod_l (const wchar_t * *c,* wchar_t ** *endptr,* const **wxXLocale** & *loc*)

long wxStrtol_l (const wchar_t * *c,* wchar_t ** *endptr,* int *base,* const **wxXLocale** & *loc*)

unsigned long wxStrtoul_l (const wchar_t * *c,* wchar_t ** *endptr,* int *base,* const **wxXLocale** & *loc*)

wchar_t wxTolower_l (wchar_t *c,* const **wxXLocale** & *loc*)

wchar_t wxToupper_l (wchar_t *c,* const **wxXLocale** & *loc*)

20.32 Logging

20.32.1 Detailed Description

wxWidgets provides several classes and functions for message logging.

Related Overviews: Logging Overview

Related macros/global-functions group: Logging

Classes

- class wxMessageOutput

  Simple class allowing to write strings to various output channels.
- class wxMessageOutputStderr

  Output messages to stderr or another STDIO file stream.
- class wxMessageOutputBest

  Output messages in the best possible way.
- class wxMessageOutputDebug

  Output messages to the system debug output channel.
- class wxMessageOutputMessageBox

  Output messages by showing them in a message box.
- class wxLogFormatter

  wxLogFormatter class is used to format the log messages.
- class wxLog

  wxLog class defines the interface for the log targets used by wxWidgets logging functions as explained in the Logging Overview.
- class wxLogChain

  This simple class allows you to chain log sinks, that is to install a new sink but keep passing log messages to the old one instead of replacing it completely as wxLog::SetActiveTarget does.
- class wxLogInterposer

  A special version of wxLogChain which uses itself as the new log target.
- class wxLogInterposerTemp

  A special version of wxLogChain which uses itself as the new log target.
- class wxLogStream

  This class can be used to redirect the log messages to a C++ stream.
- class wxLogStderr

  This class can be used to redirect the log messages to a C file stream (not to be confused with C++ streams).
- class wxLogBuffer

  wxLogBuffer is a very simple implementation of log sink which simply collects all the logged messages in a string (except the debug messages which are output in the usual way immediately as we're presumably not interested in collecting them for later).
- class wxLogNull

  This class allows you to temporarily suspend logging.
- class wxLogWindow

  This class represents a background log window: to be precise, it collects all log messages in the log frame which it manages but also passes them on to the log target which was active at the moment of its creation.
- class wxLogGui

  This is the default log target for the GUI wxWidgets applications.
- class wxLogTextCtrl

  Using these target all the log messages can be redirected to a text control.
- class wxStreamToTextRedirector

  This class can be used to (temporarily) redirect all output sent to a C++ ostream object to a wxTextCtrl instead.

20.33 Logging

20.33.1 Detailed Description

These functions provide a variety of logging functions.

The functions use (implicitly) the currently active log target, so their descriptions here may not apply if the log target is not the standard one (installed by wxWidgets in the beginning of the program).

Related Overviews: Logging Overview

Related class group: Logging

Macros

- #define WXTRACE(format,...)
- #define WXTRACELEVEL(level, format,...)

Functions

- void wxSafeShowMessage (const wxString &title, const wxString &text)

  This function shows a message to the user in a safe way and should be safe to call even before the application has been initialized or if it is currently in some other strange state (for example, about to crash).
- unsigned long wxSysErrorCode ()

  Returns the error code from the last system call.
- const wxChar * wxSysErrorMsg (unsigned long errCode=0)

  Returns the error message corresponding to the given system error code.
- void wxLogGeneric (wxLogLevel level, const char *formatString,...)

  Logs a message with the given wxLogLevel.
- void wxVLogGeneric (wxLogLevel level, const char *formatString, va_list argPtr)
- void wxLogMessage (const char *formatString,...)

  For all normal, informational messages.
- void wxVLogMessage (const char *formatString, va_list argPtr)
- void wxLogVerbose (const char *formatString,...)

  For verbose output.
- void wxVLogVerbose (const char *formatString, va_list argPtr)
- void wxLogWarning (const char *formatString,...)

  For warnings - they are also normally shown to the user, but don't interrupt the program work.
- void wxVLogWarning (const char *formatString, va_list argPtr)
- void wxLogFatalError (const char *formatString,...)

  Like wxLogError(), but also terminates the program with the exit code 3.
- void wxVLogFatalError (const char *formatString, va_list argPtr)
- void wxLogError (const char *formatString,...)

  The functions to use for error messages, i.e.
- void wxVLogError (const char *formatString, va_list argPtr)
- void wxLogTrace (const char *mask, const char *formatString,...)

  Log a message at `wxLOG_Trace` log level (see wxLogLevelValues enum).
- void wxVLogTrace (const char *mask, const char *formatString, va_list argPtr)
- void wxLogTrace (wxTraceMask mask, const char *formatString,...)

  Like wxLogDebug(), trace functions only do something in debug builds and expand to nothing in the release one.
- void wxVLogTrace (wxTraceMask mask, const char *formatString, va_list argPtr)
- void wxLogDebug (const char *formatString,...)

  The right functions for debug output.

- void wxVLogDebug (const char *formatString, va_list argPtr)
- void wxLogStatus (wxFrame *frame, const char *formatString,...)

  Messages logged by this function will appear in the statusbar of the frame or of the top level application window by default (i.e.
- void wxVLogStatus (wxFrame *frame, const char *formatString, va_list argPtr)
- void wxLogStatus (const char *formatString,...)
- void wxVLogStatus (const char *formatString, va_list argPtr)
- void wxLogSysError (const char *formatString,...)

  Mostly used by wxWidgets itself, but might be handy for logging errors after system call (API function) failure.
- void wxVLogSysError (const char *formatString, va_list argPtr)
- void wxTrace (const wxString &format,...)
- void wxTraceLevel (int level, const wxString &format,...)

20.33.2 Macro Definition Documentation

#define WXTRACE(*format, ...*)

Deprecated Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Calls wxTrace() with printf-style variable argument syntax. Output is directed to the current output stream (see wxDebugContext).

Include file:

```
#include <wx/memory.h>
```

#define WXTRACELEVEL(*level, format, ...*)

Deprecated Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Calls wxTraceLevel with printf-style variable argument syntax. Output is directed to the current output stream (see wxDebugContext). The first argument should be the level at which this information is appropriate. It will only be output if the level returned by wxDebugContext::GetLevel is equal to or greater than this value.

Include file:

```
#include <wx/memory.h>
```

20.33.3 Function Documentation

void wxLogDebug (const char * *formatString, ...*)

The right functions for debug output.

They only do something in debug mode (when the preprocessor symbol **WXDEBUG** is defined) and expand to nothing in release mode (otherwise).

Include file:

```
#include <wx/log.h>
```

void wxLogError (const char * *formatString,* ...)

The functions to use for error messages, i.e.

the messages that must be shown to the user. The default processing is to pop up a message box to inform the user about it.

Include file:

```
#include <wx/log.h>
```

void wxLogFatalError (const char * *formatString,* ...)

Like wxLogError(), but also terminates the program with the exit code 3.

Using *abort()* standard function also terminates the program with this exit code.

Include file:

```
#include <wx/log.h>
```

void wxLogGeneric (wxLogLevel *level,* const char * *formatString,* ...)

Logs a message with the given wxLogLevel.

E.g. using `wxLOG_Message` as first argument, this function behaves like wxLogMessage().

Include file:

```
#include <wx/log.h>
```

void wxLogMessage (const char * *formatString,* ...)

For all normal, informational messages.

They also appear in a message box by default (but it can be changed).

Include file:

```
#include <wx/log.h>
```

void wxLogStatus (wxFrame * *frame,* const char * *formatString,* ...)

Messages logged by this function will appear in the statusbar of the *frame* or of the top level application window by default (i.e.

when using the second version of the functions).

If the target frame doesn't have a statusbar, the message will be lost.

Include file:

```
#include <wx/log.h>
```

void wxLogStatus (const char * *formatString,* ...)

void wxLogSysError (const char * *formatString,* ...)

Mostly used by wxWidgets itself, but might be handy for logging errors after system call (API function) failure.

It logs the specified message text as well as the last system error code (*errno* or *GetLastError()* depending on the platform) and the corresponding error message. The second form of this function takes the error code explicitly as the first argument.

See also

wxSysErrorCode(), wxSysErrorMsg()

Include file:

```
#include <wx/log.h>
```

void wxLogTrace (const char * *mask*, const char * *formatString*, ...)

Log a message at `wxLOG_Trace` log level (see wxLogLevelValues enum).

Notice that the use of trace masks is not recommended any more as setting the log components (please see Log Messages Selection) provides a way to do the same thing for log messages of any level, and not just the tracing ones.

Like wxLogDebug(), trace functions only do something in debug builds and expand to nothing in the release one. The reason for making it a separate function is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages.

Trace messages can be separated into different categories; these functions in facts only log the message if the given *mask* is currently enabled in wxLog. This lets you selectively trace only some operations and not others by enabling the desired trace masks with wxLog::AddTraceMask() or by setting the `WXTRACE` environment variable.

The predefined string trace masks used by wxWidgets are:

wxTRACE_MemAlloc	Trace memory allocation (new/delete)
wxTRACE_Messages	Trace window messages/X callbacks
wxTRACE_ResAlloc	Trace GDI resource allocation
wxTRACE_RefCount	Trace various ref counting operations
wxTRACE_OleCalls	Trace OLE method calls (Win32 only)

Include file:

```
#include <wx/log.h>
```

void wxLogTrace (wxTraceMask *mask*, const char * *formatString*, ...)

Like wxLogDebug(), trace functions only do something in debug builds and expand to nothing in the release one.

The reason for making it a separate function is that usually there are a lot of trace messages, so it might make sense to separate them from other debug messages.

Deprecated This version of wxLogTrace() only logs the message if all the bits corresponding to the *mask* are set in the wxLog trace mask which can be set by calling wxLog::SetTraceMask(). This version is less flexible than wxLogTrace(const char*,const char*,...) because it doesn't allow defining the user trace masks easily. This is why it is deprecated in favour of using string trace masks.

The following bitmasks are defined for wxTraceMask:

wxTraceMemAlloc	Trace memory allocation (new/delete)
wxTraceMessages	Trace window messages/X callbacks

wxTraceResAlloc	Trace GDI resource allocation
wxTraceRefCount	Trace various ref counting operations
wxTraceOleCalls	Trace OLE method calls (Win32 only)

Include file:

```
#include <wx/log.h>
```

void wxLogVerbose (const char * *formatString,* ...)

For verbose output.

Normally, it is suppressed, but might be activated if the user wishes to know more details about the program progress (another, but possibly confusing name for the same function could be `wxLogInfo`).

Include file:

```
#include <wx/log.h>
```

void wxLogWarning (const char * *formatString,* ...)

For warnings - they are also normally shown to the user, but don't interrupt the program work.

Include file:

```
#include <wx/log.h>
```

void wxSafeShowMessage (const wxString & *title,* const wxString & *text*)

This function shows a message to the user in a safe way and should be safe to call even before the application has been initialized or if it is currently in some other strange state (for example, about to crash).

Under Windows this function shows a message box using a native dialog instead of wxMessageBox() (which might be unsafe to call), elsewhere it simply prints the message to the standard output using the title as prefix.

Parameters

title	The title of the message box shown to the user or the prefix of the message string.
text	The text to show to the user.

See also

wxLogFatalError()

Include file:

```
#include <wx/log.h>
```

unsigned long wxSysErrorCode ( )

Returns the error code from the last system call.

This function uses `errno` on Unix platforms and `GetLastError` under Win32.

See also

wxSysErrorMsg(), wxLogSysError()

Include file:

```
#include <wx/log.h>
```

const wxChar* wxSysErrorMsg (unsigned long *errCode* = 0)

Returns the error message corresponding to the given system error code.

If *errCode* is 0 (default), the last error code (as returned by wxSysErrorCode()) is used.

See also

wxSysErrorCode(), wxLogSysError()

Include file:

```
#include <wx/log.h>
```

void wxTrace (const wxString & *format*, ...)

Deprecated Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Takes printf-style variable argument syntax. Output is directed to the current output stream (see wxDebugContext).

Include file:

```
#include <wx/memory.h>
```

void wxTraceLevel (int *level*, const wxString & *format*, ...)

Deprecated Use one of the wxLogTrace() functions or one of the wxVLogTrace() functions instead.

Takes *printf()* style variable argument syntax. Output is directed to the current output stream (see wxDebugContext). The first argument should be the level at which this information is appropriate. It will only be output if the level returned by wxDebugContext::GetLevel() is equal to or greater than this value.

Include file:

```
#include <wx/memory.h>
```

void wxVLogDebug (const char * *formatString*, va_list *argPtr*)

void wxVLogError (const char * *formatString*, va_list *argPtr*)

void wxVLogFatalError (const char * *formatString*, va_list *argPtr*)

void wxVLogGeneric (wxLogLevel *level*, const char * *formatString*, va_list *argPtr*)

void wxVLogMessage (const char * *formatString*, va_list *argPtr*)

void wxVLogStatus (wxFrame * *frame*, const char * *formatString*, va_list *argPtr*)

void wxVLogStatus (const char * *formatString*, va_list *argPtr*)

void wxVLogSysError (const char * *formatString*, va_list *argPtr*)

void wxVLogTrace (const char * *mask*, const char * *formatString*, va_list *argPtr*)

void wxVLogTrace (wxTraceMask *mask*, const char * *formatString*, va_list *argPtr*)

void wxVLogVerbose (const char * *formatString,* va_list *argPtr*)

void wxVLogWarning (const char * *formatString,* va_list *argPtr*)

20.34 Managed Windows

20.34.1 Detailed Description

There are several types of window that are directly controlled by the window manager (such as MS Windows, or the Motif Window Manager).

Frames and dialogs are similar in wxWidgets, but only dialogs may be modal.

Related Overviews: Common Dialogs

Related macros/global-functions group: Dialogs

Classes

- class wxFrame

  A frame is a window whose size and position can (usually) be changed by the user.
- class wxMDIClientWindow

  An MDI client window is a child of wxMDIParentFrame, and manages zero or more wxMDIChildFrame objects.
- class wxMDIParentFrame

  An MDI (Multiple Document Interface) parent frame is a window which can contain MDI child frames in its client area which emulates the full desktop.
- class wxMDIChildFrame

  An MDI child frame is a frame that can only exist inside a wxMDIClientWindow, which is itself a child of wxMDI↩ ParentFrame.
- class wxMiniFrame

  A miniframe is a frame with a small title bar.
- class wxPopupWindow

  A special kind of top level window used for popup menus, combobox popups and such.
- class wxPopupTransientWindow

  A wxPopupWindow which disappears automatically when the user clicks mouse outside it or if it loses focus in any other way.
- class wxPropertySheetDialog

  This class represents a property sheet dialog: a tabbed dialog for showing settings.
- class wxSplashScreen

  wxSplashScreen shows a window with a thin border, displaying a bitmap describing your application.
- class wxTipWindow

  Shows simple text in a popup tip window on creation.
- class wxTopLevelWindow

  wxTopLevelWindow is a common base class for wxDialog and wxFrame.

20.35 Math

20.35.1 Detailed Description

The functions in this section are typically related with math operations and floating point numbers.

Functions

- int wxFinite (double x)

  Returns a non-zero value if x is neither infinite nor NaN (not a number), returns 0 otherwise.
- unsigned int wxGCD (unsigned int u, unsigned int v)

  Returns the greatest common divisor of the two given numbers.
- bool wxIsNaN (double x)

  Returns a non-zero value if x is NaN (not a number), returns 0 otherwise.
- wxFloat64 wxConvertFromIeeeExtended (const wxInt8 ∗bytes)

  Converts the given array of 10 bytes (corresponding to 80 bits) to a float number according to the IEEE floating point standard format (aka IEEE standard 754).
- void wxConvertToIeeeExtended (wxFloat64 num, wxInt8 ∗bytes)

  Converts the given floating number num in a sequence of 10 bytes which are stored in the given array bytes (which must be large enough) according to the IEEE floating point standard format (aka IEEE standard 754).
- double wxDegToRad (double deg)

  Convert degrees to radians.
- double wxRadToDeg (double rad)

  Convert radians to degrees.
- int wxRound (double x)

  Small wrapper around round().
- bool wxIsSameDouble (double x, double y)

  Returns true if both double values are identical.
- bool wxIsNullDouble (double x)

  Return true of x is exactly zero.

20.35.2 Function Documentation

wxFloat64 wxConvertFromIeeeExtended (const wxInt8 ∗ *bytes*)

Converts the given array of 10 bytes (corresponding to 80 bits) to a float number according to the IEEE floating point standard format (aka IEEE standard 754).

See also

wxConvertToIeeeExtended() to perform the opposite operation

void wxConvertToIeeeExtended (wxFloat64 *num*, wxInt8 ∗ *bytes*)

Converts the given floating number *num* in a sequence of 10 bytes which are stored in the given array *bytes* (which must be large enough) according to the IEEE floating point standard format (aka IEEE standard 754).

See also

wxConvertFromIeeeExtended() to perform the opposite operation

double wxDegToRad (double *deg*)

Convert degrees to radians.

This function simply returns its argument multiplied by `M_PI/180` but is more readable than writing this expression directly.

See also

wxRadToDeg()

Since

3.1.0

int wxFinite (double *x*)

Returns a non-zero value if *x* is neither infinite nor NaN (not a number), returns 0 otherwise.

Include file:

```
#include <wx/math.h>
```

unsigned int wxGCD (unsigned int *u*, unsigned int *v*)

Returns the greatest common divisor of the two given numbers.

Since

3.1.0

Include file:

```
#include <wx/math.h>
```

bool wxIsNaN (double *x*)

Returns a non-zero value if x is NaN (not a number), returns 0 otherwise.

Include file:

```
#include <wx/math.h>
```

bool wxIsNullDouble (double *x*)

Return true of *x* is exactly zero.

This is only reliable if it has been assigned 0.

bool wxIsSameDouble (double *x*, double *y*)

Returns true if both double values are identical.

This is only reliable if both values have been assigned the same value.

double wxRadToDeg (double *rad*)

Convert radians to degrees.

This function simply returns its argument multiplied by `180/M_PI` but is more readable than writing this expression directly.

See also

wxDegToRad()

Since

3.1.0

int wxRound (double *x*)

Small wrapper around round().

20.36 Menus

20.36.1 Detailed Description

Group of classes for handling menu bars and items.

Classes

- class wxMenuBar

  A menu bar is a series of menus accessible from the top of a frame.
- class wxMenu

  A menu is a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu).
- class wxMenuItem

  A menu item represents an item in a menu.

20.37 Miscellaneous

20.37.1 Detailed Description

Group of miscellaneous classes.

Related macros/global-functions group: Miscellaneous

Classes

- class wxPowerResource

  Helper functions for acquiring and releasing the given power resource.
- class wxPowerResourceBlocker

  Helper RAII class ensuring that power resources are released.
- class wxAccessible

  The wxAccessible class allows wxWidgets applications, and wxWidgets itself, to return extended information about user interface elements to client applications such as screen readers.
- class wxAffineMatrix2D

  A 3x2 matrix representing an affine 2D transformation.
- class wxMatrix2D

  A simple container for 2x2 matrix.
- class wxAffineMatrix2DBase

  A 2x3 matrix representing an affine 2D transformation.
- class wxAppProgressIndicator

  A helper class that can be used to update the progress bar in the taskbar button.
- class wxArtProvider

  wxArtProvider class is used to customize the look of wxWidgets application.
- class wxCaret

  A caret is a blinking cursor showing the position where the typed text will appear.
- class wxJoystick

  wxJoystick allows an application to control one or more joysticks.
- class wxNotificationMessage

  This class allows to show the user a message non intrusively.
- class wxQuantize

  Performs quantization, or colour reduction, on a wxImage.
- class wxRecursionGuardFlag

  This is a completely opaque class which exists only to be used with wxRecursionGuard, please see the example in that class' documentation.
- class wxRecursionGuard

  wxRecursionGuard is a very simple class which can be used to prevent reentrancy problems in a function.
- class wxScopeGuard

  Scope guard is an object which allows executing an action on scope exit.
- class wxStopWatch

  The wxStopWatch class allow you to measure time intervals.
- class wxTaskBarIcon

  This class represents a taskbar icon.
- class wxThumbBarButton

  A thumbnail toolbar button is a control that displayed in the thumbnail image of a window in a taskbar button flyout.
- class wxTaskBarButton

  A taskbar button that associated with the window under Windows 7 or later.
- class wxTaskBarJumpListItem

A wxTaskBarJumpListItem represents an item in a jump list category.

- class wxTaskBarJumpListCategory

  This class represents a category of jump list in the taskbar button.

- class wxTaskBarJumpList

  This class is an transparent wrapper around Windows Jump Lists.

- class wxTimer

  The wxTimer class allows you to execute code at specified intervals.

- class wxTipProvider

  This is the class used together with wxShowTip() function.

- class wxWindowDisabler

  This class disables all windows of the application (may be with the exception of one of them) in its constructor and enables them back in its destructor.

- class wxBusyCursor

  This class makes it easy to tell your user that the program is temporarily busy.

- class wxFSVolume

  wxFSVolume represents a volume (also known as 'drive') in a file system under wxMSW.

- class wxWindowUpdateLocker

  This tiny class prevents redrawing of a wxWindow during its lifetime by using wxWindow::Freeze() and wxWindow::Thaw() methods.

20.38 Miscellaneous

20.38.1 Detailed Description

Group of miscellaneous functions and macros.

Related class group: Miscellaneous

Macros

- #define wxCONCAT(x1, x2)

  This macro returns the concatenation of the arguments passed.
- #define wxCONCAT3(x1, x2, x3)
- #define wxCONCAT4(x1, x2, x3, x4)
- #define wxCONCAT5(x1, x2, x3, x4, x5)
- #define wxSTRINGIZE(x)

  Returns the string representation of the given symbol which can be either a literal or a macro (hence the advantage of using this macro instead of the standard preprocessor # operator which doesn't work with macros).
- #define wxSTRINGIZE_T(x)

  Returns the string representation of the given symbol as either an ASCII or Unicode string, depending on the current build.
- #define __WXFUNCTION__

  This macro expands to the name of the current function if the compiler supports any of ***`__FUNCTION__`****,* ***`__func__`*** *or equivalent variables or macros or to NULL if none of them is available.*
- #define wxDECLARE_NO_ASSIGN_CLASS(classname)

  This macro can be used in a class declaration to disable the generation of default assignment operator.
- #define wxDECLARE_NO_COPY_CLASS(classname)

  This macro can be used in a class declaration to disable the generation of default copy ctor and assignment operator.
- #define wxDECLARE_NO_COPY_TEMPLATE_CLASS(classname, arg)

  Analog of wxDECLARE_NO_COPY_CLASS() for template classes.
- #define wxDECLARE_NO_COPY_TEMPLATE_CLASS_2(classname, arg1, arg2)

  Analog of wxDECLARE_NO_COPY_TEMPLATE_CLASS() for templates with 2 parameters.
- #define wxDEPRECATED(function)

  Generate deprecation warning with the given message when a function is used.
- #define wxDEPRECATED_BUT_USED_INTERNALLY(function)

  This is a special version of wxDEPRECATED() macro which only does something when the deprecated function is used from the code outside wxWidgets itself but doesn't generate warnings when it is used from wxWidgets.
- #define wxDEPRECATED_INLINE(func, body)

  This macro is similar to wxDEPRECATED() but can be used to not only declare the function function as deprecated but to also provide its (inline) implementation body.
- #define wxDEPRECATED_ACCESSOR(func, what)

  A helper macro allowing to easily define a simple deprecated accessor.
- #define wxDEPRECATED_BUT_USED_INTERNALLY_INLINE(func, body)

  Combination of wxDEPRECATED_BUT_USED_INTERNALLY() and wxDEPRECATED_INLINE().
- #define wxEXPLICIT

  `wxEXPLICIT` is a macro which expands to the C++ `explicit` keyword if the compiler supports it or nothing otherwise.
- #define wxOVERRIDE

  `wxOVERRIDE` expands to the C++11 `override` keyword if it's supported by the compiler or nothing otherwise.
- #define wxSUPPRESS_GCC_PRIVATE_DTOR_WARNING(name)

  GNU C++ compiler gives a warning for any class whose destructor is private unless it has a friend.
- #define wxDYNLIB_FUNCTION(type, name, dynlib)

When loading a function from a DLL you always have to cast the returned `void *` pointer to the correct type and, even more annoyingly, you have to repeat this type twice if you want to declare and define a function pointer all in one line.

- #define wxLongLongFmtSpec

  This macro is defined to contain the `printf()` format specifier using which 64 bit integer numbers (i.e.

- #define wxON_BLOCK_EXIT(function,...)

  Ensure that the global function with a few (up to some implementation-defined limit) is executed on scope exit, whether due to a normal function return or because an exception has been thrown.

- #define wxON_BLOCK_EXIT0(function)
- #define wxON_BLOCK_EXIT1(function, p1)
- #define wxON_BLOCK_EXIT2(function, p1, p2)
- #define wxON_BLOCK_EXIT3(function, p1, p2, p3)
- #define wxON_BLOCK_EXIT_OBJ(object, method,...)

  This family of macros is similar to wxON_BLOCK_EXIT(), but calls a method of the given object instead of a free function.

- #define wxON_BLOCK_EXIT_OBJ0(object, method)
- #define wxON_BLOCK_EXIT_OBJ1(object, method, p1)
- #define wxON_BLOCK_EXIT_OBJ2(object, method, p1, p2)
- #define wxON_BLOCK_EXIT_OBJ3(object, method, p1, p2, p3)
- #define wxON_BLOCK_EXIT_THIS(method,...)

  This family of macros is similar to wxON_BLOCK_OBJ(), but calls a method of `this` object instead of a method of the specified object.

- #define wxON_BLOCK_EXIT_THIS0(method)
- #define wxON_BLOCK_EXIT_THIS1(method, p1)
- #define wxON_BLOCK_EXIT_THIS2(method, p1, p2)
- #define wxON_BLOCK_EXIT_THIS3(method, p1, p2, p3)
- #define wxON_BLOCK_EXIT_SET(var, value)

  This macro sets a variable to the specified value on scope exit.

- #define wxON_BLOCK_EXIT_NULL(ptr)

  This macro sets the pointer passed to it as argument to NULL on scope exit.

Typedefs

- typedef int(* wxSortCallback)(const void *pItem1, const void *pItem2, const void *user_data)

  Compare function type for use with wxQsort()

Enumerations

- enum wxBase64DecodeMode {
  wxBase64DecodeMode_Strict,
  wxBase64DecodeMode_SkipWS,
  wxBase64DecodeMode_Relaxed }

  Elements of this enum specify the possible behaviours of wxBase64Decode when an invalid character is encountered.

- enum {
  wxStrip_Mnemonics = 1,
  wxStrip_Accel = 2,
  wxStrip_All = wxStrip_Mnemonics | wxStrip_Accel }

  flags for wxStripMenuCodes

Functions

- size_t wxBase64Encode (char *dst, size_t dstLen, const void *src, size_t srcLen)

  This function encodes the given data using base64.
- wxString wxBase64Encode (const void *src, size_t srcLen)

  This function encodes the given data using base64 and returns the output as a wxString.
- wxString wxBase64Encode (const wxMemoryBuffer &buf)

  This function encodes the given data using base64 and returns the output as a wxString.
- size_t wxBase64DecodedSize (size_t srcLen)

  Returns the size of the buffer necessary to contain the data encoded in a base64 string of length srcLen.
- size_t wxBase64EncodedSize (size_t len)

  Returns the length of the string with base64 representation of a buffer of specified size len.
- size_t wxBase64Decode (void *dst, size_t dstLen, const char *src, size_t srcLen=wxNO_LEN, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  This function decodes a Base64-encoded string.
- size_t wxBase64Decode (void *dst, size_t dstLen, const wxString &str, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  Decode a Base64-encoded wxString.
- wxMemoryBuffer wxBase64Decode (const char *src, size_t srcLen=wxNO_LEN, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  Decode a Base64-encoded string and return decoded contents in a buffer.
- wxMemoryBuffer wxBase64Decode (const wxString &src, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  Decode a Base64-encoded wxString and return decoded contents in a buffer.
- bool wxFromString (const wxString &string, wxColour *colour)

  Converts string to a wxColour best represented by the given string.
- wxString wxToString (const wxColour &colour)

  Converts the given wxColour into a string.
- void wxDDECleanUp ()

  Called when wxWidgets exits, to clean up the DDE system.
- void wxDDEInitialize ()

  Initializes the DDE system.
- template<typename T >
  wxDELETE (T *&ptr)

  A function which deletes and nulls the pointer.
- template<typename T >
  wxDELETEA (T *&array)

  A function which deletes and nulls the pointer.
- template<typename T >
  wxSwap (T &first, T &second)

  Swaps the contents of two variables.
- void wxVaCopy (va_list argptrDst, va_list argptrSrc)

  This macro is the same as the standard C99 `va_copy` for the compilers which support it or its replacement for those that don't.
- bool wxFromString (const wxString &string, wxFont *font)

  Converts string to a wxFont best represented by the given string.
- wxString wxToString (const wxFont &font)

  Converts the given wxFont into a string.
- wxLongLong_t wxLL (number)

  This macro is defined for the platforms with a native 64 bit integer type and allow the use of 64 bit compile time constants:
- wxLongLong_t wxULL (number)

This macro is defined for the platforms with a native 64 bit integer type and allow the use of 64 bit compile time constants:

- template<typename F , typename P1 , ... , typename PN >
  wxScopeGuard wxMakeGuard (F func, P1 p1,..., PN pN)

  Returns a scope guard object which will call the specified function with the given parameters on scope exit.
- wxString wxGetStockLabel (wxWindowID id, long flags=wxSTOCK_WITH_MNEMONIC)

  Returns label that should be used for given id element.
- wxBatteryState wxGetBatteryState ()

  Returns battery state as one of `wxBATTERY_NORMAL_STATE`, `wxBATTERY_LOW_STATE`, `wxBATTERY_CRITICAL_STATE`, `wxBATTERY_SHUTDOWN_STATE` *or* `wxBATTERY_UNKNOWN_STATE`.
- wxPowerType wxGetPowerType ()

  Returns the type of power source as one of `wxPOWER_SOCKET`, `wxPOWER_BATTERY` *or* `wxPOWER_UNKNOWN`.
- wxString wxGetDisplayName ()

  Under X only, returns the current display name.
- bool wxGetKeyState (wxKeyCode key)

  For normal keys, returns true if the specified key is currently down.
- wxPoint wxGetMousePosition ()

  Returns the mouse position in screen coordinates.
- wxMouseState wxGetMouseState ()

  Returns the current state of the mouse.
- void wxEnableTopLevelWindows (bool enable=true)

  This function enables or disables all top level windows.
- wxWindow * wxFindWindowAtPoint (const wxPoint &pt)

  Find the deepest window at the given mouse position in screen coordinates, returning the window if found, or NULL if not.
- wxWindow * wxFindWindowByLabel (const wxString &label, wxWindow *parent=NULL)
- wxWindow * wxFindWindowByName (const wxString &name, wxWindow *parent=NULL)
- int wxFindMenuItemId (wxFrame *frame, const wxString &menuString, const wxString &itemString)

  Find a menu item identifier associated with the given frame's menu bar.
- int wxNewId ()
- void wxRegisterId (int id)

  Ensures that Ids subsequently generated by wxNewId() do not clash with the given id.
- bool wxLaunchDefaultApplication (const wxString &document, int flags=0)

  Opens the document in the application associated with the files of this type.
- bool wxLaunchDefaultBrowser (const wxString &url, int flags=0)

  Opens the url in user's default browser.
- bool wxLoadUserResource (const void **outData, size_t *outLen, const wxString &resourceName, const wxChar *resourceType="TEXT", WXHINSTANCE module=0)

  Loads an object from Windows resource file.
- char * wxLoadUserResource (const wxString &resourceName, const wxChar *resourceType="TEXT", int *pLen=NULL, WXHINSTANCE module=0)

  Loads a user-defined Windows resource as a string.
- void wxPostDelete (wxObject *object)
- void wxQsort (void *pbase, size_t total_elems, size_t size, wxSortCallback cmp, const void *user_data)

  Function implementing quick sort algorithm.
- void wxSetDisplayName (const wxString &displayName)

  Under X only, sets the current display name.
- wxString wxStripMenuCodes (const wxString &str, int flags=wxStrip_All)

  Strips any menu codes from str and returns the result.
- wxWindow * wxFindWindowAtPointer (wxPoint &pt)

  Find the deepest window at the mouse pointer position, returning the window and current pointer position in screen coordinates.

- wxWindow * wxGetActiveWindow ()

  Gets the currently active window (implemented for MSW and GTK only currently, always returns NULL in the other ports).
- wxWindow * wxGetTopLevelParent (wxWindow *window)

  Returns the first top level parent of the given window, or in other words, the frame or dialog containing it, or NULL.

20.38.2 Macro Definition Documentation

#define __WXFUNCTION__

This macro expands to the name of the current function if the compiler supports any of **`FUNCTION`**, **`func`** or equivalent variables or macros or to NULL if none of them is available.

Include file:

```
#include <wx/cpp.h>
```

#define wxCONCAT(*x1*, *x2*)

This macro returns the concatenation of the arguments passed.

Unlike when using the preprocessor operator, the arguments undergo macro expansion before being concatenated.

Include file:

```
#include <wx/cpp.h>
```

#define wxCONCAT3(*x1*, *x2*, *x3*)

#define wxCONCAT4(*x1*, *x2*, *x3*, *x4*)

#define wxCONCAT5(*x1*, *x2*, *x3*, *x4*, *x5*)

#define wxDECLARE_NO_ASSIGN_CLASS(*classname*)

This macro can be used in a class declaration to disable the generation of default assignment operator.

Some classes have a well-defined copy constructor but cannot have an assignment operator, typically because they can't be modified once created. In such case, this macro can be used to disable the automatic assignment operator generation.

See also

wxDECLARE_NO_COPY_CLASS()

#define wxDECLARE_NO_COPY_CLASS(*classname*)

This macro can be used in a class declaration to disable the generation of default copy ctor and assignment operator.

Some classes don't have a well-defined copying semantics. In this case the standard C++ convention is to not allow copying them. One way of achieving it is to use this macro which simply defines a private copy constructor and assignment operator.

Beware that simply not defining copy constructor and assignment operator is *not* enough as the compiler would provide its own automatically-generated versions of them – hence the usefulness of this macro.

Example of use:

```
1 class FooWidget
2 {
3 public:
4     FooWidget();
5     ...
6
7 private:
8     // widgets can't be copied
9     wxDECLARE_NO_COPY_CLASS(FooWidget);
10 };
```

Notice that a semicolon must be used after this macro and that it changes the access specifier to private internally so it is better to use it at the end of the class declaration.

See also

wxDECLARE_NO_ASSIGN_CLASS(), wxDECLARE_NO_COPY_TEMPLATE_CLASS()

#define wxDECLARE_NO_COPY_TEMPLATE_CLASS(*classname, arg*)

Analog of wxDECLARE_NO_COPY_CLASS() for template classes.

This macro can be used for template classes (with a single template parameter) for the same purpose as wxDECLARE_NO_COPY_CLASS() is used with the non-template classes.

Parameters

classname	The name of the template class.
arg	The name of the template parameter.

See also

wxDECLARE_NO_COPY_TEMPLATE_CLASS_2

#define wxDECLARE_NO_COPY_TEMPLATE_CLASS_2(*classname, arg1, arg2*)

Analog of wxDECLARE_NO_COPY_TEMPLATE_CLASS() for templates with 2 parameters.

This macro can be used for template classes with two template parameters for the same purpose as wxDECLARE_NO_COPY_CLASS() is used with the non-template classes.

Parameters

classname	The name of the template class.
arg1	The name of the first template parameter.
arg2	The name of the second template parameter.

See also

wxDECLARE_NO_COPY_TEMPLATE_CLASS

#define wxDEPRECATED(*function*)

Generate deprecation warning with the given message when a function is used.

This macro can be used to generate a warning indicating that a function is deprecated (i.e. scheduled for removal in the future) and explaining why is it so and/or what should it be replaced with. It applies to the declaration following it, for example:

```
1 wxDEPRECATED_MSG("use safer overload returning wxString instead")
2 void wxGetSomething(char* buf, size_t len);
3
4 wxString wxGetSomething();
```

For compilers other than clang, g++ 4.5 or later and MSVC 8 (MSVS 2005) or later, the message is ignored and a generic deprecation warning is given if possible, i.e. if the compiler is g++ (any supported version) or MSVC 7 (MSVS 2003) or later.

Since

3.0

Include file:

```
#include <wx/defs.h>
```

This macro can be used around a function declaration to generate warnings indicating that this function is deprecated (i.e. obsolete and planned to be removed in the future) when it is used.

Notice that this macro itself is deprecated in favour of wxDEPRECATED_MSG()!

Only Visual C++ 7 and higher and g++ compilers currently support this functionality.

Example of use:

```
1 // old function, use wxString version instead
2 wxDEPRECATED( void wxGetSomething(char *buf, size_t len) );
3
4 // ...
5 wxString wxGetSomething();
```

Include file:

```
#include <wx/defs.h>
```

#define wxDEPRECATED_ACCESSOR(*func, what*)

A helper macro allowing to easily define a simple deprecated accessor.

Compared to wxDEPRECATED_INLINE() it saves a `return` statement and, especially, a strangely looking semicolon inside a macro.

Example of use

```
1 class wxFoo
2 {
3 public:
4     int GetValue() const { return m_value; }
5
6     // this one is deprecated because it was erroneously non-const
7     wxDEPRECATED_ACCESSOR( int GetValue(), m_value )
8
9 private:
10     int m_value;
11 };
```

#define wxDEPRECATED_BUT_USED_INTERNALLY(*function*)

This is a special version of wxDEPRECATED() macro which only does something when the deprecated function is used from the code outside wxWidgets itself but doesn't generate warnings when it is used from wxWidgets.

It is used with the virtual functions which are called by the library itself – even if such function is deprecated the library still has to call it to ensure that the existing code overriding it continues to work, but the use of this macro ensures that a deprecation warning will be generated if this function is used from the user code or, in case of Visual C++, even when it is simply overridden.

Include file:

```
#include <wx/defs.h>
```

#define wxDEPRECATED_BUT_USED_INTERNALLY_INLINE(*func, body*)

Combination of wxDEPRECATED_BUT_USED_INTERNALLY() and wxDEPRECATED_INLINE().

This macro should be used for deprecated functions called by the library itself (usually for backwards compatibility reasons) and which are defined inline.

Include file:

```
#include <wx/defs.h>
```

#define wxDEPRECATED_INLINE(*func, body*)

This macro is similar to wxDEPRECATED() but can be used to not only declare the function *function* as deprecated but to also provide its (inline) implementation *body*.

It can be used as following:

```
1 class wxFoo
2 {
3 public:
4     // OldMethod() is deprecated, use NewMethod() instead
5     void NewMethod();
6     wxDEPRECATED_INLINE( void OldMethod(), NewMethod(); )
7 };
```

Include file:

```
#include <wx/defs.h>
```

#define wxDYNLIB_FUNCTION(*type, name, dynlib*)

When loading a function from a DLL you always have to cast the returned `void *` pointer to the correct type and, even more annoyingly, you have to repeat this type twice if you want to declare and define a function pointer all in one line.

This macro makes this slightly less painful by allowing you to specify the type only once, as the first parameter, and creating a variable of this type named after the function but with `pfn` prefix and initialized with the function *name* from the wxDynamicLibrary *dynlib*.

Parameters

type	The type of the function.
name	The name of the function to load, not a string (without quotes, it is quoted automatically by the macro).
dynlib	The library to load the function from.

Include file:

```
#include <wx/dynlib.h>
```

#define wxEXPLICIT

`wxEXPLICIT` is a macro which expands to the C++ `explicit` keyword if the compiler supports it or nothing otherwise.

Thus, it can be used even in the code which might have to be compiled with an old compiler without support for this language feature but still take advantage of it when it is available.

Include file:

```
#include <wx/defs.h>
```

#define wxLongLongFmtSpec

This macro is defined to contain the `printf()` format specifier using which 64 bit integer numbers (i.e. those of type `wxLongLong_t`) can be printed. Example of using it:

```
1 #ifdef wxLongLong_t
2     wxLongLong_t ll = wxLL(0x1234567890abcdef);
3     printf("Long long = %" wxLongLongFmtSpec "x\n", ll);
4 #endif
```

See also

wxLL()

Include file:

```
#include <wx/longlong.h>
```

#define wxON_BLOCK_EXIT(*function, ...*)

Ensure that the global *function* with a few (up to some implementation-defined limit) is executed on scope exit, whether due to a normal function return or because an exception has been thrown.

A typical example of its usage:

```
1 void *buf = malloc(size);
2 wxON_BLOCK_EXIT1(free, buf);
```

Please see the original article by Andrei Alexandrescu and Petru Marginean published in December 2000 issue of C/C++ Users Journal for more details.

See also

wxON_BLOCK_EXIT_OBJ0()

Include file:

```
#include <wx/scopeguard.h>
```

#define wxON_BLOCK_EXIT0(*function*)

#define wxON_BLOCK_EXIT1(*function, p1*)

#define wxON_BLOCK_EXIT2(*function, p1, p2*)

#define wxON_BLOCK_EXIT3(*function, p1, p2, p3*)

#define wxON_BLOCK_EXIT_NULL(*ptr*)

This macro sets the pointer passed to it as argument to NULL on scope exit.

It must be used instead of wxON_BLOCK_EXIT_SET() when the value being set is `NULL`.

Include file:

```
#include <wx/scopeguard.h>
```

#define wxON_BLOCK_EXIT_OBJ(*object, method, ...*)

This family of macros is similar to wxON_BLOCK_EXIT(), but calls a method of the given object instead of a free function.

Include file:

```
#include <wx/scopeguard.h>
```

#define wxON_BLOCK_EXIT_OBJ0(*object, method*)

#define wxON_BLOCK_EXIT_OBJ1(*object, method, p1*)

#define wxON_BLOCK_EXIT_OBJ2(*object, method, p1, p2*)

#define wxON_BLOCK_EXIT_OBJ3(*object, method, p1, p2, p3*)

#define wxON_BLOCK_EXIT_SET(*var, value*)

This macro sets a variable to the specified value on scope exit.

Example of usage:

```
1 void foo()
2 {
3     bool isDoingSomething = true;
4     {
5         wxON_BLOCK_EXIT_SET(isDoingSomething, false);
6         ... do something ...
7     }
8     ... isDoingSomething is false now ...
9 }
```

Notice that *value* is copied, i.e. stored by value, so it can be a temporary object returned by a function call, for example.

See also

wxON_BLOCK_EXIT_OBJ0(), wxON_BLOCK_EXIT_NULL()

Include file:

```
#include <wx/scopeguard.h>
```

#define wxON_BLOCK_EXIT_THIS(*method, ...*)

This family of macros is similar to wxON_BLOCK_OBJ(), but calls a method of `this` object instead of a method of the specified object.

Include file:

```
#include <wx/scopeguard.h>
```

#define wxON_BLOCK_EXIT_THIS0(*method*)

#define wxON_BLOCK_EXIT_THIS1(*method, p1*)

#define wxON_BLOCK_EXIT_THIS2(*method, p1, p2*)

#define wxON_BLOCK_EXIT_THIS3(*method, p1, p2, p3*)

#define wxOVERRIDE

`wxOVERRIDE` expands to the C++11 `override` keyword if it's supported by the compiler or nothing otherwise.

This macro is useful for writing code which may be compiled by both C++11 and non-C++11 compilers and still allow the use of `override` for the former.

Example of using this macro:

```
1 class MyApp : public wxApp {
2 public:
3     virtual bool OnInit() wxOVERRIDE;
4
5     // This would result in an error from a C++11 compiler as the
6     // method doesn't actually override the base class OnExit() due to
7     // a typo in its name.
8     //virtual int OnEzit() wxOVERRIDE;
9 };
```

Include file:

```
#include <wx/defs.h>
```

Since

3.1.0

#define wxSTRINGIZE(*x*)

Returns the string representation of the given symbol which can be either a literal or a macro (hence the advantage of using this macro instead of the standard preprocessor # operator which doesn't work with macros).

Notice that this macro always produces a `char` string, use wxSTRINGIZE_T() to build a wide string Unicode build.

See also

wxCONCAT()

Include file:

```
#include <wx/cpp.h>
```

#define wxSTRINGIZE_T(*x*)

Returns the string representation of the given symbol as either an ASCII or Unicode string, depending on the current build.

This is the Unicode-friendly equivalent of wxSTRINGIZE().

Include file:

```
#include <wx/cpp.h>
```

#define wxSUPPRESS_GCC_PRIVATE_DTOR_WARNING(*name*)

GNU C++ compiler gives a warning for any class whose destructor is private unless it has a friend.

This warning may sometimes be useful but it doesn't make sense for reference counted class which always delete themselves (hence destructor should be private) but don't necessarily have any friends, so this macro is provided

to disable the warning in such case. The *name* parameter should be the name of the class but is only used to construct a unique friend class name internally.

Example of using the macro:

```
1 class RefCounted
2 {
3 public:
4     RefCounted() { m_nRef = 1; }
5     void IncRef() { m_nRef++ ; }
6     void DecRef() { if ( !--m_nRef ) delete this; }
7
8 private:
9     ~RefCounted() { }
10
11     wxSUPPRESS_GCC_PRIVATE_DTOR(RefCounted)
12 };
```

Notice that there should be no semicolon after this macro.

Include file:

```
#include <wx/defs.h>
```

20.38.3 Typedef Documentation

typedef int(∗ wxSortCallback)(const void ∗pItem1, const void ∗pItem2, const void ∗user_data)

Compare function type for use with wxQsort()

Include file:

```
#include <wx/utils.h>
```

20.38.4 Enumeration Type Documentation

anonymous enum

flags for wxStripMenuCodes

Enumerator

wxStrip_Mnemonics

wxStrip_Accel

wxStrip_All

enum wxBase64DecodeMode

Elements of this enum specify the possible behaviours of wxBase64Decode when an invalid character is encountered.

Enumerator

wxBase64DecodeMode_Strict Normal behaviour: stop at any invalid characters.

wxBase64DecodeMode_SkipWS Skip whitespace characters.

wxBase64DecodeMode_Relaxed The most lenient behaviour: simply ignore all invalid characters.

20.38.5 Function Documentation

size_t wxBase64Decode (void * *dst*, size_t *dstLen*, const char * *src*, size_t *srcLen* = wxNO_LEN, wxBase64DecodeMode *mode* = wxBase64DecodeMode_Strict, size_t * *posErr* = NULL)

This function decodes a Base64-encoded string.

This overload is a raw decoding function and decodes the data into the provided buffer *dst* of the given size *dstLen*. An error is returned if the buffer is not large enough – that is not at least wxBase64DecodedSize(srcLen) bytes. Notice that the buffer will *not* be NULL-terminated.

This overload returns the number of bytes written to the buffer or the necessary buffer size if *dst* was NULL or wxCONV_FAILED on error, e.g. if the output buffer is too small or invalid characters were encountered in the input string.

Parameters

dst	Pointer to output buffer, may be NULL to just compute the necessary buffer size.
dstLen	The size of the output buffer, ignored if dst is NULL.
src	The input string, must not be NULL. For the version using wxString, the input string should contain only ASCII characters.
srcLen	The length of the input string or special value wxNO_LEN if the string is NULL-terminated and the length should be computed by this function itself.
mode	This parameter specifies the function behaviour when invalid characters are encountered in input. By default, any such character stops the decoding with error. If the mode is wxBase64DecodeMode_SkipWS, then the white space characters are silently skipped instead. And if it is wxBase64DecodeMode_Relaxed, then all invalid characters are skipped.
posErr	If this pointer is non-NULL and an error occurs during decoding, it is filled with the index of the invalid character.

Include file:

```
#include <wx/base64.h>
```

size_t wxBase64Decode (void * *dst*, size_t *dstLen*, const wxString & *str*, wxBase64DecodeMode *mode* = wxBase64DecodeMode_Strict, size_t * *posErr* = NULL)

Decode a Base64-encoded wxString.

See the wxBase64Decode(void*,size_t,const char*,size_t,wxBase64DecodeMode,size_t*) overload for more information about the parameters of this function, the only difference between it and this one is that a wxString is used instead of a char* pointer and its length.

Since

2.9.1

Include file:

```
#include <wx/base64.h>
```

wxMemoryBuffer wxBase64Decode (const char * *src*, size_t *srcLen* = wxNO_LEN, wxBase64DecodeMode *mode* = wxBase64DecodeMode_Strict, size_t * *posErr* = NULL)

Decode a Base64-encoded string and return decoded contents in a buffer.

See the wxBase64Decode(void*,size_t,const char*,size_t,wxBase64DecodeMode,size_t*) overload for more information about the parameters of this function. The difference of this overload is that it allocates a buffer of necessary size on its own and returns it, freeing you from the need to do it manually. Because of this, it is simpler to use and is recommended for normal use.

Include file:

```
#include <wx/base64.h>
```

wxMemoryBuffer wxBase64Decode (const wxString & *src*, wxBase64DecodeMode *mode* = wxBase64DecodeMode_Strict, size_t * *posErr* = NULL)

Decode a Base64-encoded wxString and return decoded contents in a buffer.

See the wxBase64Decode(void*,size_t,const char*,size_t,wxBase64DecodeMode,size_t*) overload for more information about the parameters of this function.

This overload takes as input a wxString and returns the internally-allocated memory as a wxMemoryBuffer, containing the Base64-decoded data.

Include file:

```
#include <wx/base64.h>
```

size_t wxBase64DecodedSize (size_t *srcLen*)

Returns the size of the buffer necessary to contain the data encoded in a base64 string of length *srcLen*.

This can be useful for allocating a buffer to be passed to wxBase64Decode().

Include file:

```
#include <wx/base64.h>
```

size_t wxBase64Encode (char * *dst*, size_t *dstLen*, const void * *src*, size_t *srcLen*)

This function encodes the given data using base64.

To allocate the buffer of the correct size, use wxBase64EncodedSize() or call this function with *dst* set to NULL – it will then return the necessary buffer size.

This raw encoding function overload writes the output string into the provided buffer; the other overloads return it as a wxString.

Parameters

dst	The output buffer, may be NULL to retrieve the needed buffer size.
dstLen	The output buffer size, ignored if dst is NULL.
src	The input buffer, must not be NULL.
srcLen	The length of the input data.

Returns

wxCONV_FAILED if the output buffer is too small.

Include file:

```
#include <wx/base64.h>
```

wxString wxBase64Encode (const void * *src*, size_t *srcLen*)

This function encodes the given data using base64 and returns the output as a wxString.

There is no error return.

To allocate the buffer of the correct size, use wxBase64EncodedSize() or call this function with *dst* set to NULL – it will then return the necessary buffer size.

Parameters

src	The input buffer, must not be NULL.
srcLen	The length of the input data.

Include file:

```
#include <wx/base64.h>
```

wxString wxBase64Encode (const wxMemoryBuffer & *buf*)

This function encodes the given data using base64 and returns the output as a wxString.

There is no error return.

Include file:

```
#include <wx/base64.h>
```

size_t wxBase64EncodedSize (size_t *len*)

Returns the length of the string with base64 representation of a buffer of specified size *len*.

This can be useful for allocating the buffer passed to wxBase64Encode().

Include file:

```
#include <wx/base64.h>
```

void wxDDECleanUp ( )

Called when wxWidgets exits, to clean up the DDE system.

This no longer needs to be called by the application.

See also

wxDDEInitialize()

Include file:

```
#include <wx/dde.h>
```

void wxDDEInitialize ( )

Initializes the DDE system.

May be called multiple times without harm.

This no longer needs to be called by the application: it will be called by wxWidgets if necessary.

See also

wxDDEServer, wxDDEClient, wxDDEConnection, wxDDECleanUp()

Include file:

```
#include <wx/dde.h>
```

template<typename T > wxDELETE (T *& *ptr*)

A function which deletes and nulls the pointer.

This function uses operator delete to free the pointer and also sets it to NULL. Notice that this does *not* work for arrays, use wxDELETEA() for them.

```
1 MyClass *ptr = new MyClass;
2 ...
3 wxDELETE(ptr);
4 wxASSERT(!ptr);
```

Include file:

```
#include <wx/defs.h>
```

template<typename T > wxDELETEA (T *& *array*)

A function which deletes and nulls the pointer.

This function uses vector operator delete (`delete[]`) to free the array pointer and also sets it to NULL. Notice that this does *not* work for non-array pointers, use wxDELETE() for them.

```
1 MyClass *array = new MyClass[17];
2 ...
3 wxDELETEA(array);
4 wxASSERT(!array);
```

See also

wxDELETE()

Include file:

```
#include <wx/defs.h>
```

void wxEnableTopLevelWindows (bool *enable* = `true`)

This function enables or disables all top level windows.

It is used by wxSafeYield().

Include file:

```
#include <wx/utils.h>
```

int wxFindMenuItemId (wxFrame * *frame*, const wxString & *menuString*, const wxString & *itemString*)

Find a menu item identifier associated with the given frame's menu bar.

Include file:

```
#include <wx/utils.h>
```

wxWindow* wxFindWindowAtPoint (const wxPoint & *pt*)

Find the deepest window at the given mouse position in screen coordinates, returning the window if found, or NULL if not.

This function takes child windows at the given position into account even if they are disabled. The hidden children are however skipped by it.

Include file:

```
#include <wx/utils.h>
```

wxWindow* wxFindWindowAtPointer (wxPoint & *pt*)

Find the deepest window at the mouse pointer position, returning the window and current pointer position in screen coordinates.

Include file:

```
#include <wx/window.h>
```

wxWindow* wxFindWindowByLabel (const wxString & *label,* wxWindow * *parent* = NULL)

Deprecated Replaced by wxWindow::FindWindowByLabel().

Find a window by its label. Depending on the type of window, the label may be a window title or panel item label. If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

Include file:

```
#include <wx/utils.h>
```

wxWindow* wxFindWindowByName (const wxString & *name,* wxWindow * *parent* = NULL)

Deprecated Replaced by wxWindow::FindWindowByName().

Find a window by its name (as given in a window constructor or *Create* function call). If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

If no such named window is found, wxFindWindowByLabel() is called.

Include file:

```
#include <wx/utils.h>
```

bool wxFromString (const wxString & *string,* wxColour * *colour*)

Converts string to a wxColour best represented by the given string.

Returns true on success.

See also

wxToString(const wxColour&)

Include file:

```
#include <wx/colour.h>
```

bool wxFromString (const wxString & *string,* wxFont * *font*)

Converts string to a wxFont best represented by the given string.

Returns true on success.

See also

wxToString(const wxFont&)

Include file:

```
#include <wx/font.h>
```

wxWindow* wxGetActiveWindow ( )

Gets the currently active window (implemented for MSW and GTK only currently, always returns NULL in the other ports).

Include file:

```
#include <wx/window.h>
```

wxBatteryState wxGetBatteryState ( )

Returns battery state as one of `wxBATTERY_NORMAL_STATE`, `wxBATTERY_LOW_STATE`, `wxBATTERY_CRITICAL_STATE`, `wxBATTERY_SHUTDOWN_STATE` or `wxBATTERY_UNKNOWN_STATE`.

`wxBATTERY_UNKNOWN_STATE` is also the default on platforms where this feature is not implemented (currently everywhere but MS Windows).

Include file:

```
#include <wx/utils.h>
```

wxString wxGetDisplayName ( )

Under X only, returns the current display name.

See also

wxSetDisplayName()

Include file:

```
#include <wx/utils.h>
```

bool wxGetKeyState (wxKeyCode *key*)

For normal keys, returns true if the specified key is currently down.

For togglable keys (Caps Lock, Num Lock and Scroll Lock), returns true if the key is toggled such that its LED indicator is lit. There is currently no way to test whether togglable keys are up or down.

Even though there are virtual key codes defined for mouse buttons, they cannot be used with this function currently.

Include file:

```
#include <wx/utils.h>
```

wxPoint wxGetMousePosition ( )

Returns the mouse position in screen coordinates.

Include file:

```
#include <wx/utils.h>
```

wxMouseState wxGetMouseState ( )

Returns the current state of the mouse.

Returns a wxMouseState instance that contains the current position of the mouse pointer in screen coordinates, as well as boolean values indicating the up/down status of the mouse buttons and the modifier keys.

Include file:

```
#include <wx/utils.h>
```

wxPowerType wxGetPowerType ( )

Returns the type of power source as one of `wxPOWER_SOCKET`, `wxPOWER_BATTERY` or `wxPOWER_UNKNOWN`.

`wxPOWER_UNKNOWN` is also the default on platforms where this feature is not implemented (currently everywhere but MS Windows).

Include file:

```
#include <wx/utils.h>
```

wxString wxGetStockLabel (wxWindowID *id,* long *flags* = wxSTOCK_WITH_MNEMONIC)

Returns label that should be used for given *id* element.

Parameters

id	Given id of the wxMenuItem, wxButton, wxToolBar tool, etc.
flags	Combination of the elements of wxStockLabelQueryFlag.

Include file:

```
#include <wx/stockitem.h>
```

wxWindow* wxGetTopLevelParent (wxWindow * *window*)

Returns the first top level parent of the given window, or in other words, the frame or dialog containing it, or NULL.

Notice that if *window* is itself already a TLW, it is returned directly.

Include file:

```
#include <wx/window.h>
```

bool wxLaunchDefaultApplication (const wxString & *document,* int *flags* = 0)

Opens the *document* in the application associated with the files of this type.

The *flags* parameter is currently not used

Returns true if the application was successfully launched.

See also

wxLaunchDefaultBrowser(), wxExecute()

Include file:

```
#include <wx/utils.h>
```

bool wxLaunchDefaultBrowser (const wxString & *url,* int *flags* = 0)

Opens the *url* in user's default browser.

If the *flags* parameter contains `wxBROWSER_NEW_WINDOW` flag, a new window is opened for the URL (currently this is only supported under Windows).

And unless the *flags* parameter contains `wxBROWSER_NOBUSYCURSOR` flag, a busy cursor is shown while the browser is being launched (using wxBusyCursor).

The parameter *url* is interpreted as follows:

- if it has a valid scheme (e.g. `"file:"`, `"http:"` or `"mailto:"`) it is passed to the appropriate browser configured in the user system.
- if it has no valid scheme (e.g. it's a local file path without the `"file:"` prefix), then wxFileExists and wxDirExists are used to test if it's a local file/directory; if it is, then the browser is called with the *url* parameter eventually prefixed by `"file:"`.
- if it has no valid scheme and it's not a local file/directory, then `"http:"` is prepended and the browser is called.

Returns true if the application was successfully launched.

Note

For some configurations of the running user, the application which is launched to open the given URL may be URL-dependent (e.g. a browser may be used for local URLs while another one may be used for remote URLs).

See also

wxLaunchDefaultApplication(), wxExecute()

Include file:

```
#include <wx/utils.h>
```

wxLongLong_t wxLL (number)

This macro is defined for the platforms with a native 64 bit integer type and allow the use of 64 bit compile time constants:

```
1 #ifdef wxLongLong_t
2     wxLongLong_t ll = wxLL(0x1234567890abcdef);
3 #endif
```

See also

wxULL(), wxLongLong

Include file:

```
#include <wx/longlong.h>
```

bool wxLoadUserResource (const void ** *outData,* size_t * *outLen,* const wxString & *resourceName,* const wxChar * *resourceType =* "TEXT", WXHINSTANCE *module =* 0)

Loads an object from Windows resource file.

This function loads the resource with the given name and type from the resources embedded into a Windows application.

The typical use for it is to load some data from the data files embedded into the program itself. For example, you could have the following fragment in your .rc file

```
1 mydata  MYDATA  "myfile.dat"
```

and then use it in the following way:

```
1 const void* data = NULL;
2 size_t size = 0;
3 if ( !wxLoadUserResource(&data, &size, "mydata", "MYDATA") ) {
4     ... handle error ...
5 }
6 else {
7     // Use the data in any way, for example:
8     wxMemoryInputStream is(data, size);
9     ... read the data from stream ...
10 }
```

Parameters

outData	Filled with the pointer to the data on successful return. Notice that this pointer does *not* need to be freed by the caller.
outLen	Filled with the length of the data in bytes.
resourceName	The name of the resource to load.
resourceType	The type of the resource in usual Windows format, i.e. either a real string like "MYDATA" or an integer created by the standard Windows MAKEINTRESOURCE() macro, including any constants for the standard resources types like RT_RCDATA.
module	The HINSTANCE of the module to load the resources from. The current module is used by default.

Returns

true if the data was loaded from resource or false if it couldn't be found (in which case no error is logged) or was found but couldn't be loaded (which is unexpected and does result in an error message).

This function is available under Windows only.

Library: wxBase

Include file:

```
#include <wx/utils.h>
```

Since

2.9.1

char* wxLoadUserResource (const wxString & *resourceName,* const wxChar * *resourceType =* "TEXT", int * *pLen =* NULL, WXHINSTANCE *module =* 0)

Loads a user-defined Windows resource as a string.

This is a wrapper for the general purpose overload wxLoadUserResource(const void∗∗, size_t∗, const wxString&, const wxChar∗, WXHINSTANCE) and can be more convenient for the string data, but does an extra copy compared to the general version.

Parameters

resourceName	The name of the resource to load.
resourceType	The type of the resource in usual Windows format, i.e. either a real string like "MYDATA" or an integer created by the standard Windows `MAKEINTRESOURCE()` macro, including any constants for the standard resources types like `RT_RCDATA`.
pLen	Filled with the length of the returned buffer if it is non-NULL. This parameter should be used if NUL characters can occur in the resource data. It is new since wxWidgets 2.9.1
module	The `HINSTANCE` of the module to load the resources from. The current module is used by default. This parameter is new since wxWidgets 2.9.1.

Returns

A pointer to the data to be `delete[]`d by caller on success or NULL on error.

This function is available under Windows only.

Library: wxBase

Include file:

```
#include <wx/utils.h>
```

template<typename F , typename P1 , ... , typename PN > wxScopeGuard wxMakeGuard (F *func,* P1 *p1,* ..., PN *pN*)

Returns a scope guard object which will call the specified function with the given parameters on scope exit.

This function is overloaded to take several parameters up to some implementation-defined (but relatively low) limit.

The *func* should be a functor taking parameters of the types P1, ..., PN, i.e. the expression `func`(p1, ..., pN) should be valid.

int wxNewId ( )

Deprecated Ids generated by it can conflict with the Ids defined by the user code, use `wxID_ANY` to assign ids which are guaranteed to not conflict with the user-defined ids for the controls and menu items you create instead of using this function.

Generates an integer identifier unique to this run of the program.

Include file:

```
#include <wx/utils.h>
```

void wxPostDelete (wxObject * *object*)

Deprecated Replaced by wxWindow::Close(). See the window deletion overview.

Tells the system to delete the specified object when all other events have been processed. In some environments, it is necessary to use this instead of deleting a frame directly with the delete operator, because some GUIs will still send events to a deleted window.

Include file:

```
#include <wx/utils.h>
```

void wxQsort (void * *pbase,* size_t *total_elems,* size_t *size,* wxSortCallback *cmp,* const void * *user_data*)

Function implementing quick sort algorithm.

This function sorts *total_elems* objects of size *size* located at *pbase*. It uses *cmp* function for comparing them and passes *user_data* pointer to the comparison function each time it's called.

Include file:

```
#include <wx/utils.h>
```

void wxRegisterId (int *id*)

Ensures that Ids subsequently generated by wxNewId() do not clash with the given *id*.

Include file:

```
#include <wx/utils.h>
```

void wxSetDisplayName (const wxString & *displayName*)

Under X only, sets the current display name.

This is the X host and display name such as "colonsay:0.0", and the function indicates which display should be used for creating windows from this point on. Setting the display within an application allows multiple displays to be used.

See also

wxGetDisplayName()

Include file:

```
#include <wx/utils.h>
```

wxString wxStripMenuCodes (const wxString & *str,* int *flags* = wxStrip_All)

Strips any menu codes from *str* and returns the result.

By default, the functions strips both the mnemonics character (`'&'`) which is used to indicate a keyboard shortkey, and the accelerators, which are used only in the menu items and are separated from the main text by the `\t` (TAB) character. By using *flags* of `wxStrip_Mnemonics` or `wxStrip_Accel` to strip only the former or the latter part, respectively.

Notice that in most cases wxMenuItem::GetLabelFromText() or wxControl::GetLabelText() can be used instead.

Include file:

```
#include <wx/utils.h>
```

template<typename T > wxSwap (T & *first,* T & *second*)

Swaps the contents of two variables.

This is similar to std::swap() but can be used even on the platforms where the standard C++ library is not available (if you don't target such platforms, please use std::swap() instead).

The function relies on type T being copy constructible and assignable.

Example of use:

```
1 int x = 3,
2     y = 4;
3 wxSwap(x, y);
4 wxASSERT( x == 4 && y == 3 );
```

wxString wxToString (const wxColour & *colour*)

Converts the given wxColour into a string.

See also

wxFromString(const wxString&, wxColour*)

Include file:

```
#include <wx/colour.h>
```

wxString wxToString (const wxFont & *font*)

Converts the given wxFont into a string.

See also

wxFromString(const wxString&, wxFont*)

Include file:

```
#include <wx/font.h>
```

wxLongLong_t wxULL (number)

This macro is defined for the platforms with a native 64 bit integer type and allow the use of 64 bit compile time constants:

```
1 #ifdef wxLongLong_t
2     unsigned wxLongLong_t ll = wxULL(0x1234567890abcdef);
3 #endif
```

See also

wxLL(), wxLongLong

Include file:

```
#include <wx/longlong.h>
```

void wxVaCopy (va_list *argptrDst*, va_list *argptrSrc*)

This macro is the same as the standard C99 `va_copy` for the compilers which support it or its replacement for those that don't.

It must be used to preserve the value of a `va_list` object if you need to use it after passing it to another function because it can be modified by the latter.

As with `va_start`, each call to `wxVaCopy` must have a matching `va_end`.

Include file:

```
#include <wx/defs.h>
```

20.39 Miscellaneous Windows

20.39.1 Detailed Description

The following are a variety of classes that are derived from wxWindow.

Classes

- class wxBannerWindow

  A simple banner window showing either a bitmap or text.
- class wxCustomBackgroundWindow< W >

  A helper class making it possible to use custom background for any window.
- class wxInfoBar

  An info bar is a transient window shown at top or bottom of its parent window to display non-critical information to the user.
- class wxRichToolTip

  Allows to show a tool tip with more customizations than wxToolTip.
- class wxScrolled< T >

  The wxScrolled class manages scrolling for its client area, transforming the coordinates according to the scrollbar positions, and setting the scroll positions, thumb sizes and ranges according to the area in view.
- class wxSashLayoutWindow

  wxSashLayoutWindow responds to OnCalculateLayout events generated by wxLayoutAlgorithm.
- class wxPanel

  A panel is a window on which controls are placed.
- class wxSashWindow

  wxSashWindow allows any of its edges to have a sash which can be dragged to resize the window.
- class wxSplitterWindow

  This class manages up to two subwindows.
- class wxStatusBar

  A status bar is a narrow window that can be placed along the bottom of a frame to give small amounts of status information.
- class wxToolBar

  A toolbar is a bar of buttons and/or other controls usually placed below the menu bar in a wxFrame.
- class wxVarScrollHelperBase

  This class provides all common base functionality for scroll calculations shared among all variable scrolled window implementations as well as automatic scrollbar functionality, saved scroll positions, controlling target windows to be scrolled, as well as defining all required virtual functions that need to be implemented for any orientation specific work.
- class wxVarVScrollHelper

  This class provides functions wrapping the wxVarScrollHelperBase class, targeted for vertical-specific scrolling.
- class wxVarHScrollHelper

  This class provides functions wrapping the wxVarScrollHelperBase class, targeted for horizontal-specific scrolling.
- class wxVarHVScrollHelper

  This class provides functions wrapping the wxVarHScrollHelper and wxVarVScrollHelper classes, targeted for scrolling a window in both axis.
- class wxVScrolledWindow

  In the name of this class, "V" may stand for "variable" because it can be used for scrolling rows of variable heights; "virtual", because it is not necessary to know the heights of all rows in advance – only those which are shown on the screen need to be measured; or even "vertical", because this class only supports scrolling vertically.
- class wxHScrolledWindow

  In the name of this class, "H" stands for "horizontal" because it can be used for scrolling columns of variable widths.
- class wxHVScrolledWindow

  This window inherits all functionality of both vertical and horizontal, variable scrolled windows.

- class wxWindow

  wxWindow is the base class for all windows and represents any visible object on screen.

- class wxWizardPage

  wxWizardPage is one of the screens in wxWizard: it must know what are the following and preceding pages (which may be NULL for the first/last page).

- class wxWizardPageSimple

  wxWizardPageSimple is the simplest possible wxWizardPage implementation: it just returns the pointers given to its constructor from wxWizardPage::GetNext() and wxWizardPage::GetPrev() functions.

Typedefs

- typedef wxScrolled< wxPanel > wxScrolledWindow

  Scrolled window derived from wxPanel.

- typedef wxScrolled< wxWindow > wxScrolledCanvas

  Alias for wxScrolled<wxWindow>.

20.39.2 Typedef Documentation

typedef wxScrolled<wxWindow> wxScrolledCanvas

Alias for wxScrolled<wxWindow>.

Scrolled window that doesn't have children and so doesn't need or want special handling of TAB traversal.

Since

2.9.0

Library: wxCore

Category: Miscellaneous Windows

See also

wxScrolled, wxScrolledWindow

typedef wxScrolled<wxPanel> wxScrolledWindow

Scrolled window derived from wxPanel.

See wxScrolled for a detailed description.

Note

Note that because this class derives from wxPanel, it shares its behaviour with regard to TAB traversal and focus handling (in particular, it forwards focus to its children). If you don't want this behaviour, use wxScrolled↩ Canvas instead.
wxScrolledWindow is an alias for wxScrolled<wxPanel> since version 2.9.0. In older versions, it was a standalone class.

Library: wxCore

Category: Miscellaneous Windows

See also

wxScrolled, wxScrolledCanvas

20.40 Multimedia

20.40.1 Detailed Description

Classes for showing multimedia contents.

Classes

- class wxMediaCtrl

  wxMediaCtrl is a class for displaying types of media, such as videos, audio files, natively through native codecs.
- class wxSound

  This class represents a short sound (loaded from Windows WAV file), that can be stored in memory and played.

20.41 Network, User and OS

20.41.1 Detailed Description

The functions in this section are used to retrieve information about the current computer and/or user characteristics.

Related class group: Networking, wxPlatformInfo.

Functions

- wxString wxGetEmailAddress ()

  Copies the user's email address into the supplied buffer, by concatenating the values returned by wxGetFullHostName() and wxGetUserId().
- bool wxGetEmailAddress (char *buf, int sz)
- wxMemorySize wxGetFreeMemory ()

  Returns the amount of free memory in bytes under environments which support it, and -1 if not supported or failed to perform measurement.
- wxString wxGetHomeDir ()

  Return the (current) user's home directory.
- wxString wxGetHostName ()

  Copies the current host machine's name into the supplied buffer.
- bool wxGetHostName (char *buf, int sz)
- wxString wxGetFullHostName ()

  Returns the FQDN (fully qualified domain host name) or an empty string on error.
- wxString wxGetUserHome (const wxString &user=wxEmptyString)

  Returns the home directory for the given user.
- wxString wxGetUserId ()

  This function returns the "user id" also known as "login name" under Unix (i.e.
- bool wxGetUserId (char *buf, int sz)
- wxString wxGetUserName ()

  This function returns the full user name (something like "Mr. John Smith").
- bool wxGetUserName (char *buf, int sz)
- wxString wxGetOsDescription ()

  Returns the string containing the description of the current platform in a user-readable form.
- wxOperatingSystemId wxGetOsVersion (int *major=NULL, int *minor=NULL)

  Gets the version and the operating system ID for currently running OS.
- bool wxIsPlatform64Bit ()

  Returns true if the operating system the program is running under is 64 bit.
- bool wxIsPlatformLittleEndian ()

  Returns true if the current platform is little endian (instead of big endian).
- wxLinuxDistributionInfo wxGetLinuxDistributionInfo ()

  Returns a structure containing information about the currently running Linux distribution.

20.41.2 Function Documentation

wxString wxGetEmailAddress ( )

Copies the user's email address into the supplied buffer, by concatenating the values returned by wxGetFullHostName() and wxGetUserId().

Returns

true if successful, false otherwise.

Include file:

```
#include <wx/utils.h>
```

bool wxGetEmailAddress (char * *buf*, int *sz*)

Deprecated Use wxGetEmailAddress() instead.

Parameters

buf	Buffer to store the email address in.
sz	Size of the buffer.

Returns

true if successful, false otherwise.

Include file:

```
#include <wx/utils.h>
```

wxMemorySize wxGetFreeMemory ( )

Returns the amount of free memory in bytes under environments which support it, and -1 if not supported or failed to perform measurement.

Include file:

```
#include <wx/utils.h>
```

wxString wxGetFullHostName ( )

Returns the FQDN (fully qualified domain host name) or an empty string on error.

See also

wxGetHostName()

Include file:

```
#include <wx/utils.h>
```

wxString wxGetHomeDir ( )

Return the (current) user's home directory.

See also

wxGetUserHome(), wxStandardPaths

Include file:

```
#include <wx/utils.h>
```

wxString wxGetHostName ( )

Copies the current host machine's name into the supplied buffer.

Please note that the returned name is *not* fully qualified, i.e. it does not include the domain name.

Under Windows or NT, this function first looks in the environment variable SYSTEM_NAME; if this is not found, the entry **HostName** in the wxWidgets section of the WIN.INI file is tried.

Returns

The hostname if successful or an empty string otherwise.

See also

wxGetFullHostName()

Include file:

```
#include <wx/utils.h>
```

bool wxGetHostName (char * *buf*, int *sz*)

Deprecated Use wxGetHostName() instead.

Parameters

buf	Buffer to store the host name in.
sz	Size of the buffer.

Returns

true if successful, false otherwise.

Include file:

```
#include <wx/utils.h>
```

wxLinuxDistributionInfo wxGetLinuxDistributionInfo ( )

Returns a structure containing information about the currently running Linux distribution.

This function uses the `lsb_release` utility which is part of the `Linux Standard Base Core` specification (see `http://refspecs.linux-foundation.org/lsb.shtml`) since the very first LSB release 1.↩ 0 (released in 2001). The `lsb_release` utility is very common on modern Linux distributions but in case it's not available, then this function will return a wxLinuxDistributionInfo structure containing empty strings.

This function is Linux-specific and is only available when the **`LINUX`** symbol is defined.

wxString wxGetOsDescription ( )

Returns the string containing the description of the current platform in a user-readable form.

For example, this function may return strings like "Windows NT Version 4.0" or "Linux 2.2.2 i386".

See also

wxGetOsVersion()

Include file:

```
#include <wx/utils.h>
```

wxOperatingSystemId wxGetOsVersion (int * *major* = NULL, int * *minor* = NULL)

Gets the version and the operating system ID for currently running OS.

The returned wxOperatingSystemId value can be used for a basic categorization of the OS family; the major and minor version numbers allows to detect a specific system.

For Unix-like systems (wxOS_UNIX) the major and minor version integers will contain the kernel major and minor version numbers (as returned by the 'uname -r' command); e.g. "2" and "6" if the machine is using kernel 2.6.19.

For Mac OS X systems (wxOS_MAC) the major and minor version integers are the natural version numbers associated with the OS; e.g. "10" and "6" if the machine is using Mac OS X Snow Leopard.

For Windows-like systems (wxOS_WINDOWS) the major and minor version integers will contain the following values:

Windows OS name	Major version	Minor version
Windows 7	6	1
Windows Server 2008 R2	6	1
Windows Server 2008	6	0
Windows Vista	6	0
Windows Server 2003 R2	5	2
Windows Server 2003	5	2
Windows XP	5	1
Windows 2000	5	0

See the MSDN for more info about the values above.

See also

wxGetOsDescription(), wxPlatformInfo

Include file:

```
#include <wx/utils.h>
```

wxString wxGetUserHome (const wxString & *user* = wxEmptyString)

Returns the home directory for the given user.

If the *user* is empty (default value), this function behaves like wxGetHomeDir() (i.e. returns the current user home directory).

If the home directory couldn't be determined, an empty string is returned.

Include file:

```
#include <wx/utils.h>
```

wxString wxGetUserId ( )

This function returns the "user id" also known as "login name" under Unix (i.e.

something like "jsmith"). It uniquely identifies the current user (on this system). Under Windows or NT, this function first looks in the environment variables USER and LOGNAME; if neither of these is found, the entry **UserId** in the **wxWidgets** section of the WIN.INI file is tried.

Returns

The login name if successful or an empty string otherwise.

See also

wxGetUserName()

Include file:

```
#include <wx/utils.h>
```

bool wxGetUserId (char * *buf*, int *sz*)

Deprecated Use wxGetUserId() instead.

Parameters

buf	Buffer to store the login name in.
sz	Size of the buffer.

Returns

true if successful, false otherwise.

Include file:

```
#include <wx/utils.h>
```

wxString wxGetUserName ( )

This function returns the full user name (something like "Mr. John Smith").

Under Windows or NT, this function looks for the entry UserName in the wxWidgets section of the WIN.INI file. If PenWindows is running, the entry Current in the section User of the PENWIN.INI file is used.

Returns

The full user name if successful or an empty string otherwise.

See also

wxGetUserId()

Include file:

```
#include <wx/utils.h>
```

bool wxGetUserName (char * *buf*, int *sz*)

Deprecated Use wxGetUserName() instead.

Parameters

buf	Buffer to store the full user name in.

sz	Size of the buffer.

Returns

true if successful, false otherwise.

Include file:

```
#include <wx/utils.h>
```

bool wxIsPlatform64Bit ( )

Returns true if the operating system the program is running under is 64 bit.

The check is performed at run-time and may differ from the value available at compile-time (at compile-time you can just check if `sizeof(void*) == 8`) since the program could be running in emulation mode or in a mixed 32/64 bit system (bi-architecture operating system).

Note

This function is not 100% reliable on some systems given the fact that there isn't always a standard way to do a reliable check on the OS architecture.

Include file:

```
#include <wx/utils.h>
```

bool wxIsPlatformLittleEndian ( )

Returns true if the current platform is little endian (instead of big endian).

The check is performed at run-time.

See also

Byte Order Functions and Macros

Include file:

```
#include <wx/utils.h>
```

20.42 Networking

20.42.1 Detailed Description

wxWidgets provides its own classes for socket based networking.

Related macros/global-functions group: Network, User and OS

Classes

- class wxProtocolLog

  Class allowing to log network operations performed by wxProtocol.
- class wxDialUpManager

  This class encapsulates functions dealing with verifying the connection status of the workstation (connected to the Internet via a direct connection, connected through a modem or not connected at all) and to establish this connection if possible/required (i.e.
- class wxFTP

  wxFTP can be used to establish a connection to an FTP server and perform all the usual operations.
- class wxHTTP

  wxHTTP can be used to establish a connection to an HTTP server.
- class wxProtocol

  Represents a protocol for use with wxURL.
- class wxTCPServer

  A wxTCPServer object represents the server part of a client-server conversation.
- class wxTCPClient

  A wxTCPClient object represents the client part of a client-server conversation.
- class wxTCPConnection

  A wxTCPClient object represents the connection between a client and a server.
- class wxSocketOutputStream

  This class implements an output stream which writes data from a connected socket.
- class wxSocketInputStream

  This class implements an input stream which reads data from a connected socket.
- class wxIPaddress

  wxIPaddress is an abstract base class for all internet protocol address objects.
- class wxIPV4address

  A class for working with IPv4 network addresses.
- class wxSocketServer
- class wxSocketClient
- class wxSockAddress

  You are unlikely to need to use this class: only wxSocketBase uses it.
- class wxSocketEvent

  This event class contains information about socket events.
- class wxSocketBase

  wxSocketBase is the base class for all socket-related objects, and it defines all basic IO functionality.
- class wxDatagramSocket
- class wxURI

  wxURI is used to extract information from a URI (Uniform Resource Identifier).
- class wxURL

  wxURL is a specialization of wxURI for parsing URLs.

20.43 OpenGL

20.43.1 Detailed Description

Classes interfacing wxWidgets with OpenGL (http://opengl.org/).

Classes

- class wxGLContext

  An instance of a wxGLContext represents the state of an OpenGL state machine and the connection between Open↩ GL and the system.
- class wxGLCanvas

  wxGLCanvas is a class for displaying OpenGL graphics.

20.44 Picker Controls

20.44.1 Detailed Description

A picker control is a control whose appearance and behaviour is highly platform-dependent.

Classes

- class wxColourPickerCtrl

  This control allows the user to select a colour.
- class wxDatePickerCtrl

  This control allows the user to select a date.
- class wxFilePickerCtrl

  This control allows the user to select a file.
- class wxDirPickerCtrl

  This control allows the user to select a directory.
- class wxFontPickerCtrl

  This control allows the user to select a font.
- class wxPickerBase

  Base abstract class for all pickers which support an auxiliary text control.
- class wxTimePickerCtrl

  This control allows the user to enter time.

Enumerations

- enum { wxTP_DEFAULT = 0 }

  Styles used with wxTimePickerCtrl.

20.44.2 Enumeration Type Documentation

anonymous enum

Styles used with wxTimePickerCtrl.

Currently no special styles are defined for this object.

Library: wxAdvanced

Category: Picker Controls

Since

2.9.3

Enumerator

wxTP_DEFAULT

20.45 Printing Framework

20.45.1 Detailed Description

A printing and previewing framework is implemented to make it relatively straightforward to provide document printing facilities.

Related Overviews: Printing Framework Overview

Classes

- class wxPageSetupDialogData

  This class holds a variety of information related to wxPageSetupDialog.
- class wxPrintData

  This class holds a variety of information related to printers and printer device contexts.
- class wxPrintDialogData

  This class holds information related to the visual characteristics of wxPrintDialog.
- class wxPrinterDC

  A printer device context is specific to MSW and Mac, and allows access to any printer with a Windows or Macintosh driver.
- class wxHtmlEasyPrinting

  This class provides very simple interface to printing architecture.
- class wxHtmlPrintout

  This class serves as printout class for HTML documents.
- class wxPreviewControlBar

  This is the default implementation of the preview control bar, a panel with buttons and a zoom control.
- class wxPreviewCanvas

  A preview canvas is the default canvas used by the print preview system to display the preview.
- class wxPreviewFrame

  This class provides the default method of managing the print preview interface.
- class wxPrintPreview

  Objects of this class manage the print preview process.
- class wxPrinter

  This class represents the Windows or PostScript printer, and is the vehicle through which printing may be launched by an application.
- class wxPrintout

  This class encapsulates the functionality of printing out an application document.
- class wxPrintDialog

  This class represents the print and print setup common dialogs.
- class wxPageSetupDialog

  This class represents the page setup common dialog.

20.46 Process Control

20.46.1 Detailed Description

The functions in this section are used to launch or terminate the other processes.

Classes

- struct wxExecuteEnv

  This structure can optionally be passed to wxExecute() to specify additional options to use for the child process.

Enumerations

- enum {
  wxEXEC_ASYNC = 0,
  wxEXEC_SYNC = 1,
  wxEXEC_SHOW_CONSOLE = 2,
  wxEXEC_MAKE_GROUP_LEADER = 4,
  wxEXEC_NODISABLE = 8,
  wxEXEC_NOEVENTS = 16,
  wxEXEC_HIDE_CONSOLE = 32,
  wxEXEC_BLOCK = wxEXEC_SYNC | wxEXEC_NOEVENTS }

  Bit flags that can be used with wxExecute().

Functions

- void wxExit ()

  Exits application after calling wxApp::OnExit.
- long wxExecute (const wxString &command, int flags=wxEXEC_ASYNC, wxProcess *callback=NULL, const wxExecuteEnv *env=NULL)

  Executes another program in Unix or Windows.
- long wxExecute (char **argv, int flags=wxEXEC_ASYNC, wxProcess *callback=NULL, const wxExecuteEnv *env=NULL)

  This is an overloaded version of wxExecute(const wxString&,int,wxProcess), please see its documentation for general information.*
- long wxExecute (wchar_t **argv, int flags=wxEXEC_ASYNC, wxProcess *callback=NULL, const wxExecuteEnv *env=NULL)
- long wxExecute (const wxString &command, wxArrayString &output, int flags=0, const wxExecuteEnv *env=NULL)

  This is an overloaded version of wxExecute(const wxString&,int,wxProcess), please see its documentation for general information.*
- long wxExecute (const wxString &command, wxArrayString &output, wxArrayString &errors, int flags=0, const wxExecuteEnv *env=NULL)

  This is an overloaded version of wxExecute(const wxString&,int,wxProcess), please see its documentation for general information.*
- unsigned long wxGetProcessId ()

  Returns the number uniquely identifying the current process in the system.
- int wxKill (long pid, wxSignal sig=wxSIGTERM, wxKillError *rc=NULL, int flags=wxKILL_NOCHILDREN)

  Equivalent to the Unix kill function: send the given signal sig to the process with PID pid.
- bool wxShell (const wxString &command=wxEmptyString)

  Executes a command in an interactive shell window.
- bool wxShutdown (int flags=wxSHUTDOWN_POWEROFF)

  This function shuts down or reboots the computer depending on the value of the flags.

20.46.2 Enumeration Type Documentation

anonymous enum

Bit flags that can be used with wxExecute().

Enumerator

wxEXEC_ASYNC Execute the process asynchronously. Notice that, due to its value, this is the default.

wxEXEC_SYNC Execute the process synchronously.

wxEXEC_SHOW_CONSOLE Always show the child process console under MSW. The child console is hidden by default if the child IO is redirected, this flag allows to change this and show it nevertheless.

This flag is ignored under the other platforms.

wxEXEC_MAKE_GROUP_LEADER Make the new process a group leader. Under Unix, if the process is the group leader then passing wxKILL_CHILDREN to wxKill() kills all children as well as pid.

Under MSW, applies only to console applications and is only supported under NT family (i.e. not under Windows 9x). It corresponds to the native `CREATE_NEW_PROCESS_GROUP` and, in particular, ensures that Ctrl-Break signals will be sent to all children of this process as well to the process itself. Support for this flag under MSW was added in version 2.9.4 of wxWidgets.

wxEXEC_NODISABLE Don't disable the program UI while running the child synchronously. By default synchronous execution disables all program windows to avoid that the user interacts with the program while the child process is running, you can use this flag to prevent this from happening.

This flag can only be used with wxEXEC_SYNC.

wxEXEC_NOEVENTS Don't dispatch events while the child process is executed. By default, the event loop is run while waiting for synchronous execution to complete and this flag can be used to simply block the main process until the child process finishes

This flag can only be used with wxEXEC_SYNC.

wxEXEC_HIDE_CONSOLE Hide child process console under MSW. Under MSW, hide the console of the child process if it has one, even if its IO is not redirected.

This flag is ignored under the other platforms.

wxEXEC_BLOCK Convenient synonym for flags given system()-like behaviour.

20.46.3 Function Documentation

long wxExecute (const wxString & *command*, int *flags* = wxEXEC_ASYNC, wxProcess * *callback* = `NULL`, const wxExecuteEnv * *env* = `NULL`)

Executes another program in Unix or Windows.

In the overloaded versions of this function, if *flags* parameter contains `wxEXEC_ASYNC` flag (the default), flow of control immediately returns. If it contains `wxEXEC_SYNC`, the current application waits until the other program has terminated.

In the case of synchronous execution, the return value is the exit code of the process (which terminates by the moment the function returns) and will be -1 if the process couldn't be started and typically 0 if the process terminated successfully. Also, while waiting for the process to terminate, wxExecute() will call wxYield(). Because of this, by default this function disables all application windows to avoid unexpected reentrancies which could result from the users interaction with the program while the child process is running. If you are sure that it is safe to not disable the program windows, you may pass `wxEXEC_NODISABLE` flag to prevent this automatic disabling from happening.

For asynchronous execution, however, the return value is the process id and zero value indicates that the command could not be executed. As an added complication, the return value of -1 in this case indicates that we didn't launch a new process, but connected to the running one (this can only happen when using DDE under Windows for command execution). In particular, in this case only, the calling code will not get the notification about process termination.

If *callback* isn't NULL and if execution is asynchronous, wxProcess::OnTerminate() will be called when the process finishes. Specifying this parameter also allows you to redirect the standard input and/or output of the process being launched by calling wxProcess::Redirect().

Under Windows, when launching a console process its console is shown by default but hidden if its IO is redirected. Both of these default behaviours may be overridden: if wxEXEC_HIDE_CONSOLE is specified, the console will never be shown. If wxEXEC_SHOW_CONSOLE is used, the console will be shown even if the child process IO is redirected. Neither of these flags affect non-console Windows applications or does anything under the other systems.

Under Unix the flag `wxEXEC_MAKE_GROUP_LEADER` may be used to ensure that the new process is a group leader (this will create a new session if needed). Calling wxKill() passing wxKILL_CHILDREN will kill this process as well as all of its children (except those which have started their own session). Under MSW, this flag can be used with console processes only and corresponds to the native `CREATE_NEW_PROCESS_GROUP` flag.

The `wxEXEC_NOEVENTS` flag prevents processing of any events from taking place while the child process is running. It should be only used for very short-lived processes as otherwise the application windows risk becoming unresponsive from the users point of view. As this flag only makes sense with `wxEXEC_SYNC`, `wxEXEC_BLOCK` equal to the sum of both of these flags is provided as a convenience.

Note

Currently wxExecute() can only be used from the main thread, calling this function from another thread will result in an assert failure in debug build and won't work.

Parameters

command	The command to execute and any parameters to pass to it as a single string, i.e. "emacs file.txt".
flags	Must include either wxEXEC_ASYNC or wxEXEC_SYNC and can also include wxEXEC_SHOW_CONSOLE, wxEXEC_HIDE_CONSOLE, wxEXEC_MAKE_GROUP_LEADER (in either case) or wxEXEC_NODISABLE and wxEXEC_NOEVENTS or wxEXEC_BLOCK, which is equal to their combination, in wxEXEC_SYNC case.
callback	An optional pointer to wxProcess.
env	An optional pointer to additional parameters for the child process, such as its initial working directory and environment variables. This parameter is available in wxWidgets 2.9.2 and later only.

See also

wxShell(), wxProcess, External Program Execution Sample, wxLaunchDefaultApplication(), wxLaunchDefaultBrowser()

Include file:

```
#include <wx/utils.h>
```

wxPerl Note: In wxPerl this function is called `Wx::ExecuteCommand`.

long wxExecute (char ** *argv*, int *flags* = wxEXEC_ASYNC, wxProcess * *callback* = NULL, const wxExecuteEnv * *env* = NULL)

This is an overloaded version of wxExecute(const wxString&,int,wxProcess*), please see its documentation for general information.

This version takes an array of values: a command, any number of arguments, terminated by NULL.

Parameters

argv	The command to execute should be the first element of this array, any additional ones are the command parameters and the array must be terminated with a NULL pointer.

flags	Same as for wxExecute(const wxString&,int,wxProcess*) overload.
callback	An optional pointer to wxProcess.
env	An optional pointer to additional parameters for the child process, such as its initial working directory and environment variables. This parameter is available in wxWidgets 2.9.2 and later only.

See also

wxShell(), wxProcess, External Program Execution Sample, wxLaunchDefaultApplication(), wxLaunchDefaultBrowser()

Include file:

```
#include <wx/utils.h>
```

wxPerl Note: In wxPerl this function is called `Wx::ExecuteArgs`.

long wxExecute (wchar_t ** *argv,* int *flags* = wxEXEC_ASYNC, wxProcess * *callback* = NULL, const wxExecuteEnv * *env* = NULL)

long wxExecute (const wxString & *command,* wxArrayString & *output,* int *flags* = 0, const wxExecuteEnv * *env* = NULL)

This is an overloaded version of wxExecute(const wxString&,int,wxProcess*), please see its documentation for general information.

This version can be used to execute a process (always synchronously, the contents of *flags* is or'd with `wxEXEC_SYNC`) and capture its output in the array *output*.

Parameters

command	The command to execute and any parameters to pass to it as a single string.
output	The string array where the stdout of the executed process is saved.
flags	Combination of flags to which wxEXEC_SYNC is always implicitly added.
env	An optional pointer to additional parameters for the child process, such as its initial working directory and environment variables. This parameter is available in wxWidgets 2.9.2 and later only.

See also

wxShell(), wxProcess, External Program Execution Sample, wxLaunchDefaultApplication(), wxLaunchDefaultBrowser()

Include file:

```
#include <wx/utils.h>
```

wxPerl Note: This function is called `Wx::ExecuteStdout`: it only takes the *command* argument, and returns a 2-element list (`status`, `output`), where `output` in an array reference.

long wxExecute (const wxString & *command,* wxArrayString & *output,* wxArrayString & *errors,* int *flags* = 0, const wxExecuteEnv * *env* = NULL)

This is an overloaded version of wxExecute(const wxString&,int,wxProcess*), please see its documentation for general information.

This version adds the possibility to additionally capture the messages from standard error output in the *errors* array. As with the above overload capturing standard output only, execution is always synchronous.

Parameters

command	The command to execute and any parameters to pass to it as a single string.
output	The string array where the stdout of the executed process is saved.
errors	The string array where the stderr of the executed process is saved.
flags	Combination of flags to which wxEXEC_SYNC is always implicitly added.
env	An optional pointer to additional parameters for the child process, such as its initial working directory and environment variables. This parameter is available in wxWidgets 2.9.2 and later only.

See also

wxShell(), wxProcess, External Program Execution Sample, wxLaunchDefaultApplication(), wxLaunchDefaultBrowser()

Include file:

```
#include <wx/utils.h>
```

wxPerl Note: This function is called `Wx::ExecuteStdoutStderr`: it only takes the *command* argument, and returns a 3-element list (`status`, `output`, `errors`), where `output` and `errors` are array references.

void wxExit ( )

Exits application after calling wxApp::OnExit.

Should only be used in an emergency: normally the top-level frame should be deleted (after deleting all other frames) to terminate the application. See wxCloseEvent and wxApp.

Include file:

```
#include <wx/app.h>
```

unsigned long wxGetProcessId ( )

Returns the number uniquely identifying the current process in the system.

If an error occurs, 0 is returned.

Include file:

```
#include <wx/utils.h>
```

int wxKill (long *pid*, wxSignal *sig* = wxSIGTERM, wxKillError * *rc* = NULL, int *flags* = wxKILL_NOCHILDREN)

Equivalent to the Unix kill function: send the given signal *sig* to the process with PID *pid*.

The valid signal values are:

```
1 enum wxSignal
2 {
3     wxSIGNONE = 0, // verify if the process exists under Unix
4     wxSIGHUP,
5     wxSIGINT,
6     wxSIGQUIT,
7     wxSIGILL,
8     wxSIGTRAP,
9     wxSIGABRT,
10     wxSIGEMT,
11     wxSIGFPE,
12     wxSIGKILL,     // forcefully kill, dangerous!
13     wxSIGBUS,
14     wxSIGSEGV,
```

```
15      wxSIGSYS,
16      wxSIGPIPE,
17      wxSIGALRM,
18      wxSIGTERM       // terminate the process gently
19 };
```

wxSIGNONE, wxSIGKILL and wxSIGTERM have the same meaning under both Unix and Windows but all the other signals are equivalent to wxSIGTERM under Windows. Moreover, under Windows, wxSIGTERM is implemented by posting a message to the application window, so it only works if the application does have windows. If it doesn't, as is notably always the case for the console applications, you need to use wxSIGKILL to actually kill the process. Of course, this doesn't allow the process to shut down gracefully and so should be avoided if possible.

Returns 0 on success, -1 on failure. If the *rc* parameter is not NULL, it will be filled with a value from the wxKill↩Error enum:

```
1 enum wxKillError
2 {
3     wxKILL_OK,              // no error
4     wxKILL_BAD_SIGNAL,      // no such signal
5     wxKILL_ACCESS_DENIED,   // permission denied
6     wxKILL_NO_PROCESS,      // no such process
7     wxKILL_ERROR            // another, unspecified error
8 };
```

The *flags* parameter can be wxKILL_NOCHILDREN (the default), or wxKILL_CHILDREN, in which case the child processes of this process will be killed too. Note that under Unix, for wxKILL_CHILDREN to work you should have created the process by passing wxEXEC_MAKE_GROUP_LEADER to wxExecute().

See also

wxProcess::Kill(), wxProcess::Exists(), External Program Execution Sample

Include file:

```
#include <wx/utils.h>
```

bool wxShell (const wxString & *command* = wxEmptyString)

Executes a command in an interactive shell window.

If no command is specified, then just the shell is spawned.

See also

wxExecute(), External Program Execution Sample

Include file:

```
#include <wx/utils.h>
```

bool wxShutdown (int *flags* = wxSHUTDOWN_POWEROFF)

This function shuts down or reboots the computer depending on the value of the *flags*.

Note

Note that performing the shutdown requires the corresponding access rights (superuser under Unix, SE_S↩HUTDOWN privilege under Windows NT) and that this function is only implemented under Unix and MSW.

Parameters

flags	One of `wxSHUTDOWN_POWEROFF`, `wxSHUTDOWN_REBOOT` or `wxSHUTDOWN_LOGOFF` (currently implemented only for MSW) possibly combined with `wxSHUTDOWN_FORCE` which forces shutdown under MSW by forcefully terminating all the applications. As doing this can result in a data loss, this flag shouldn't be used unless really necessary.

Returns

true on success, false if an error occurred.

Include file:

```
#include <wx/utils.h>
```

20.47 Ribbon User Interface

20.47.1 Detailed Description

The wxRibbon library is a set of classes for writing a ribbon user interface.

At the most generic level, this is a combination of a tab control with a toolbar. At a more functional level, it is similar to the user interface present in recent versions of Microsoft Office.

Classes

- class wxRibbonArtProvider

  wxRibbonArtProvider is responsible for drawing all the components of the ribbon interface.
- class wxRibbonBarEvent

  Event used to indicate various actions relating to a wxRibbonBar.
- class wxRibbonBar

  Top-level control in a ribbon user interface.
- class wxRibbonButtonBar

  A ribbon button bar is similar to a traditional toolbar.
- class wxRibbonButtonBarEvent

  Event used to indicate various actions relating to a button on a wxRibbonButtonBar.
- class wxRibbonControl

  wxRibbonControl serves as a base class for all controls which share the ribbon characteristics of having a ribbon art provider, and (optionally) non-continuous resizing.
- class wxRibbonGallery

  A ribbon gallery is like a wxListBox, but for bitmaps rather than strings.
- class wxRibbonGalleryEvent
- class wxRibbonPage

  Container for related ribbon panels, and a tab within a ribbon bar.
- class wxRibbonPanelEvent

  Event used to indicate various actions relating to a wxRibbonPanel.
- class wxRibbonPanel

  Serves as a container for a group of (ribbon) controls.
- class wxRibbonToolBar

  A ribbon tool bar is similar to a traditional toolbar which has no labels.

20.48 Rich Text

20.48.1 Detailed Description

wxWidgets provides a set of generic classes to edit and print simple rich text with character and paragraph formatting.

Classes

- class wxTextAttrDimension

  A class representing a rich text dimension, including units and position.
- class wxTextAttrDimensions

  A class for left, right, top and bottom dimensions.
- class wxTextAttrSize

  A class for representing width and height.
- class wxTextAttrDimensionConverter

  A class to make it easier to convert dimensions.
- class wxTextAttrBorder

  A class representing a rich text object border.
- class wxTextAttrBorders

  A class representing a rich text object's borders.
- class wxTextAttrShadow

  A class representing a shadow.
- class wxTextBoxAttr

  A class representing the box attributes of a rich text object.
- class wxRichTextAttr

  A class representing enhanced attributes for rich text objects.
- class wxRichTextProperties

  A simple property class using wxVariants.
- class wxRichTextFontTable

  Manages quick access to a pool of fonts for rendering rich text.
- class wxRichTextRange

  This stores beginning and end positions for a range of data.
- class wxRichTextSelection

  Stores selection information.
- class wxRichTextDrawingContext

  A class for passing information to drawing and measuring functions.
- class wxRichTextObject

  This is the base for drawable rich text objects.
- class wxRichTextCompositeObject

  Objects of this class can contain other objects.
- class wxRichTextParagraphLayoutBox

  This class knows how to lay out paragraphs.
- class wxRichTextBox

  This class implements a floating or inline text box, containing paragraphs.
- class wxRichTextField

  This class implements the general concept of a field, an object that represents additional functionality such as a footnote, a bookmark, a page number, a table of contents, and so on.
- class wxRichTextFieldType

  The base class for custom field types.
- class wxRichTextFieldTypeStandard

A field type that can handle fields with text or bitmap labels, with a small range of styles for implementing rectangular fields and fields that can be used for start and end tags.

- class wxRichTextLine

  This object represents a line in a paragraph, and stores offsets from the start of the paragraph representing the start and end positions of the line.
- class wxRichTextParagraph

  This object represents a single paragraph containing various objects such as text content, images, and further paragraph layout objects.
- class wxRichTextPlainText

  This object represents a single piece of text.
- class wxRichTextImageBlock

  This class stores information about an image, in binary in-memory form.
- class wxRichTextImage

  This class implements a graphic object.
- class wxRichTextBuffer

  This is a kind of paragraph layout box, used to represent the whole buffer.
- class wxRichTextObjectAddress

  A class for specifying an object anywhere in an object hierarchy, without using a pointer, necessary since wxRTC commands may delete and recreate sub-objects so physical object addresses change.
- class wxRichTextCommand

  Implements a command on the undo/redo stack.
- class wxRichTextAction

  Implements a part of a command.
- class wxRichTextFileHandler

  The base class for file handlers.
- class wxRichTextPlainTextHandler

  Implements saving a buffer to plain text.
- class wxRichTextDrawingHandler

  The base class for custom drawing handlers.
- class wxRichTextBufferDataObject

  Implements a rich text data object for clipboard transfer.
- class wxRichTextRenderer

  This class isolates some common drawing functionality.
- class wxRichTextStdRenderer

  The standard renderer for drawing bullets.
- class wxRichTextCtrl

  wxRichTextCtrl provides a generic, ground-up implementation of a text control capable of showing multiple styles and images.
- class wxRichTextEvent

  This is the event class for wxRichTextCtrl notifications.
- class wxRichTextFormattingDialogFactory

  This class provides pages for wxRichTextFormattingDialog, and allows other customization of the dialog.
- class wxRichTextFormattingDialog

  This dialog allows the user to edit a character and/or paragraph style.
- class wxRichTextHTMLHandler

  Handles HTML output (only) for wxRichTextCtrl content.
- class wxRichTextHeaderFooterData

  This class represents header and footer data to be passed to the wxRichTextPrinting and wxRichTextPrintout classes.
- class wxRichTextPrintout

  This class implements print layout for wxRichTextBuffer.
- class wxRichTextPrinting

  This class provides a simple interface for performing wxRichTextBuffer printing and previewing.

- class wxRichTextStyleOrganiserDialog

  This class shows a style sheet and allows the user to edit, add and remove styles.
- class wxRichTextStyleListCtrl

  This class incorporates a wxRichTextStyleListBox and a choice control that allows the user to select the category of style to view.
- class wxRichTextStyleDefinition

  This is a base class for paragraph and character styles.
- class wxRichTextParagraphStyleDefinition

  This class represents a paragraph style definition, usually added to a wxRichTextStyleSheet.
- class wxRichTextStyleListBox

  This is a listbox that can display the styles in a wxRichTextStyleSheet, and apply the selection to an associated wxRichTextCtrl.
- class wxRichTextStyleComboCtrl

  This is a combo control that can display the styles in a wxRichTextStyleSheet, and apply the selection to an associated wxRichTextCtrl.
- class wxRichTextCharacterStyleDefinition

  This class represents a character style definition, usually added to a wxRichTextStyleSheet.
- class wxRichTextListStyleDefinition

  This class represents a list style definition, usually added to a wxRichTextStyleSheet.
- class wxRichTextStyleSheet

  A style sheet contains named paragraph and character styles that make it easy for a user to apply combinations of attributes to a wxRichTextCtrl.
- class wxRichTextXMLHandler

  A handler for loading and saving content in an XML format specific to wxRichTextBuffer.
- class wxTextAttr

  wxTextAttr represents the character and paragraph attributes, or style, for a range of text in a wxTextCtrl or wxRich↩ TextCtrl.

20.49 Runtime Type Information (RTTI)

20.49.1 Detailed Description

wxWidgets supports runtime manipulation of class information, and dynamic creation of objects given class names.

Related Overviews: Runtime Type Information (RTTI)

Related macros/global-functions group: Runtime Type Information (RTTI)

Classes

- class wxObjectDataPtr< T >

  This is an helper template class primarily written to avoid memory leaks because of missing calls to wxRefCounter::DecRef() and wxObjectRefData::DecRef().
- class wxObjectRefData

  This class is just a typedef to wxRefCounter and is used by wxObject.
- class wxRefCounter

  This class is used to manage reference-counting providing a simple interface and a counter.
- class wxObject

  This is the root class of many of the wxWidgets classes.
- class wxClassInfo

  This class stores meta-information about classes.

20.50 Runtime Type Information (RTTI)

20.50.1 Detailed Description

wxWidgets uses its own RTTI ("run-time type identification") system which predates the current standard C++ RTTI and so is kept for backwards compatibility reasons but also because it allows some things which the standard RTTI doesn't directly support (such as creating a class from its name).

The standard C++ RTTI can be used in the user code without any problems and in general you shouldn't need to use the functions and the macros in this section unless you are thinking of modifying or adding any wxWidgets classes.

Related Overviews: Runtime Type Information (RTTI)

Related class group: Runtime Type Information (RTTI)

Macros

- #define wxDECLARE_APP(className)

  This is used in headers to create a forward declaration of the wxGetApp() function implemented by wxIMPLEMENT_APP().
- #define wxIMPLEMENT_APP(className)

  This is used in the application class implementation file to make the application class known to wxWidgets for dynamic construction.
- #define wxCLASSINFO(className)

  Returns a pointer to the wxClassInfo object associated with this class.
- #define wxDECLARE_ABSTRACT_CLASS(className)

  Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically.
- #define wxDECLARE_DYNAMIC_CLASS(className)

  Used inside a class declaration to make the class known to wxWidgets RTTI system and also declare that the objects of this class should be dynamically creatable from run-time type information.
- #define wxDECLARE_CLASS(className)

  Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically.
- #define wxIMPLEMENT_ABSTRACT_CLASS(className, baseClassName)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information.
- #define wxIMPLEMENT_ABSTRACT_CLASS2(className, baseClassName1, baseClassName2)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes.
- #define wxIMPLEMENT_DYNAMIC_CLASS(className, baseClassName)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.
- #define wxIMPLEMENT_DYNAMIC_CLASS2(className, baseClassName1, baseClassName2)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.
- #define wxIMPLEMENT_CLASS(className, baseClassName)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.
- #define wxIMPLEMENT_CLASS2(className, baseClassName1, baseClassName2)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes, and whose instances can be created dynamically.
- #define wx_const_cast(T, x)

  Same as `const_cast<T>(x)` if the compiler supports const cast or (T)x for old compilers.
- #define wx_reinterpret_cast(T, x)

Same as `reinterpret_cast<T>(x)` if the compiler supports reinterpret cast or (T)x for old compilers.

- #define wx_static_cast(T, x)

  Same as `static_cast<T>(x)` if the compiler supports static cast or (T)x for old compilers.
- #define wx_truncate_cast(T, x)

  This case doesn't correspond to any standard cast but exists solely to make casts which possibly result in a truncation of an integer value more readable.
- #define wxConstCast(ptr, classname)

  This macro expands into `const_cast<classname *>(ptr)` if the compiler supports const_cast or into an old, C-style cast, otherwise.
- #define wxDynamicCast(ptr, classname)

  *This macro returns the pointer ptr cast to the type classname * if the pointer is of this type (the check is done during the run-time) or NULL otherwise.*
- #define wxDynamicCastThis(classname)

  This macro is equivalent to `wxDynamicCast(this, classname)` but the latter provokes spurious compilation warnings from some compilers (because it tests whether `this` pointer is non-NULL which is always true), so this macro should be used to avoid them.
- #define wxStaticCast(ptr, classname)

  This macro checks that the cast is valid in debug mode (an assert failure will result if wxDynamicCast(ptr, classname) == NULL) and then returns the result of executing an equivalent of `static_cast<classname *>(ptr)`.
- #define wxGetVariantCast(var, classname)

  This macro returns a pointer to the data stored in var (wxVariant) cast to the type classname if the data is of this type (the check is done during the run-time) or NULL otherwise.

Functions

- wxObject * wxCreateDynamicObject (const wxString &className)

  Creates and returns an object of the given class, if the class has been registered with the dynamic class system using DECLARE...

20.50.2 Macro Definition Documentation

#define wx_const_cast(*T, x*)

Same as `const_cast<T>(x)` if the compiler supports const cast or (T)x for old compilers.

Unlike wxConstCast(), the cast it to the type `T` and not to `T *` and also the order of arguments is the same as for the standard cast.

Include file:

```
#include <wx/defs.h>
```

See also

wx_reinterpret_cast(), wx_static_cast()

#define wx_reinterpret_cast(*T, x*)

Same as `reinterpret_cast<T>(x)` if the compiler supports reinterpret cast or (T)x for old compilers.

Include file:

```
#include <wx/defs.h>
```

See also

wx_const_cast(), wx_static_cast()

#define wx_static_cast(*T, x*)

Same as `static_cast<T>(x)` if the compiler supports static cast or (T)x for old compilers.

Unlike wxStaticCast(), there are no checks being done and the meaning of the macro arguments is exactly the same as for the standard static cast, i.e. *T* is the full type name and star is not appended to it.

Include file:

```
#include <wx/defs.h>
```

See also

wx_const_cast(), wx_reinterpret_cast(), wx_truncate_cast()

#define wx_truncate_cast(*T, x*)

This case doesn't correspond to any standard cast but exists solely to make casts which possibly result in a truncation of an integer value more readable.

Include file:

```
#include <wx/defs.h>
```

#define wxCLASSINFO(*className*)

Returns a pointer to the wxClassInfo object associated with this class.

Include file:

```
#include <wx/object.h>
```

#define wxConstCast(*ptr, classname*)

This macro expands into `const_cast<classname *>(ptr)` if the compiler supports const_cast or into an old, C-style cast, otherwise.

Include file:

```
#include <wx/defs.h>
```

See also

wx_const_cast(), wxDynamicCast(), wxStaticCast()

#define wxDECLARE_ABSTRACT_CLASS(*className*)

Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically.

Include file:

```
#include <wx/object.h>
```

Example:

```
1 class wxCommand: public wxObject
2 {
3     wxDECLARE_ABSTRACT_CLASS(wxCommand);
4
5 private:
6     ...
7 public:
8     ...
9 };
```

#define wxDECLARE_APP(*className*)

This is used in headers to create a forward declaration of the wxGetApp() function implemented by wxIMPLEMENT_APP().

It creates the declaration `className& wxGetApp()` (requires a final semicolon).

Include file:

```
#include <wx/app.h>
```

Example:

```
1 wxDECLARE_APP(MyApp);
```

#define wxDECLARE_CLASS(*className*)

Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically.

The same as wxDECLARE_ABSTRACT_CLASS().

Include file:

```
#include <wx/object.h>
```

#define wxDECLARE_DYNAMIC_CLASS(*className*)

Used inside a class declaration to make the class known to wxWidgets RTTI system and also declare that the objects of this class should be dynamically creatable from run-time type information.

Notice that this implies that the class should have a default constructor, if this is not the case consider using wxDECLARE_ABSTRACT_CLASS().

Include file:

```
#include <wx/object.h>
```

Example:

```
1 class wxFrame: public wxWindow
2 {
3     wxDECLARE_DYNAMIC_CLASS(wxFrame);
4
5 private:
6     const wxString& frameTitle;
7 public:
8     ...
9 };
```

#define wxDynamicCast(*ptr, classname*)

This macro returns the pointer *ptr* cast to the type *classname* * if the pointer is of this type (the check is done during the run-time) or NULL otherwise.

Usage of this macro is preferred over obsoleted wxObject::IsKindOf() function.

The *ptr* argument may be NULL, in which case NULL will be returned.

Include file:

```
#include <wx/object.h>
```

Example:

```
1 wxWindow *win = wxWindow::FindFocus();
2 wxTextCtrl *text = wxDynamicCast(win, wxTextCtrl);
3 if ( text )
4 {
5     // a text control has the focus...
6 }
7 else
8 {
9     // no window has the focus or it is not a text control
10 }
```

See also

Runtime Type Information (RTTI), wxDynamicCastThis(), wxConstCast(), wxStaticCast()

#define wxDynamicCastThis(*classname*)

This macro is equivalent to `wxDynamicCast(this, classname)` but the latter provokes spurious compilation warnings from some compilers (because it tests whether `this` pointer is non-NULL which is always true), so this macro should be used to avoid them.

Include file:

```
#include <wx/object.h>
```

See also

wxDynamicCast()

#define wxGetVariantCast(*var, classname*)

This macro returns a pointer to the data stored in *var* (wxVariant) cast to the type *classname* if the data is of this type (the check is done during the run-time) or NULL otherwise.

Include file:

```
#include <wx/variant.h>
```

See also

Runtime Type Information (RTTI), wxDynamicCast()

#define wxIMPLEMENT_ABSTRACT_CLASS(*className, baseClassName*)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information.

Include file:

```
#include <wx/object.h>
```

Example:

```
1 wxIMPLEMENT_ABSTRACT_CLASS(wxCommand, wxObject);
2
3 wxCommand::wxCommand(void)
4 {
5     ...
6 }
```

#define wxIMPLEMENT_ABSTRACT_CLASS2(*className, baseClassName1, baseClassName2*)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes.

Include file:

```
#include <wx/object.h>
```

#define wxIMPLEMENT_APP(*className*)

This is used in the application class implementation file to make the application class known to wxWidgets for dynamic construction.

Note that this macro requires a final semicolon.

Include file:

```
#include <wx/app.h>
```

Example:

```
1 wxIMPLEMENT_APP(MyApp);
```

See also

wxDECLARE_APP()

#define wxIMPLEMENT_CLASS(*className, baseClassName*)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.

The same as wxIMPLEMENT_DYNAMIC_CLASS().

Include file:

```
#include <wx/object.h>
```

#define wxIMPLEMENT_CLASS2(*className, baseClassName1, baseClassName2*)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes, and whose instances can be created dynamically.

The same as wxIMPLEMENT_DYNAMIC_CLASS2().

Include file:

```
#include <wx/object.h>
```

#define wxIMPLEMENT_DYNAMIC_CLASS(*className, baseClassName*)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.

Include file:

```
#include <wx/object.h>
```

Example:

```
1 wxIMPLEMENT_DYNAMIC_CLASS(wxFrame, wxWindow);
2
3 wxFrame::wxFrame(void)
4 {
5     ...
6 }
```

#define wxIMPLEMENT_DYNAMIC_CLASS2(*className, baseClassName1, baseClassName2*)

Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.

Use this for classes derived from two base classes.

Include file:

```
#include <wx/object.h>
```

#define wxStaticCast(*ptr, classname*)

This macro checks that the cast is valid in debug mode (an assert failure will result if wxDynamicCast(ptr, classname) == NULL) and then returns the result of executing an equivalent of `static_cast<classname *>(ptr)`.

Include file:

```
#include <wx/object.h>
```

See also

wx_static_cast(), wxDynamicCast(), wxConstCast()

20.50.3 Function Documentation

wxObject* **wxCreateDynamicObject** (**const wxString &** *className*)

Creates and returns an object of the given class, if the class has been registered with the dynamic class system using DECLARE...

and IMPLEMENT... macros.

Include file:

```
#include <wx/object.h>
```

20.51 Scintilla Text Editor

20.51.1 Detailed Description

wxWidgets also provides a wrapper around the Scintilla text editor control, which is a control for plain-text editing with support for highlighting, smart indentation, etc.

Classes

- class wxStyledTextCtrl

  A wxWidgets implementation of the Scintilla source code editing component.
- class wxStyledTextEvent

  The type of events sent from wxStyledTextCtrl.

20.52 Smart Pointers

20.52.1 Detailed Description

wxWidgets provides a few smart pointer class templates.

Classes

- class wxObjectDataPtr< T >

  This is an helper template class primarily written to avoid memory leaks because of missing calls to wxRefCounter::DecRef() and wxObjectRefData::DecRef().
- class wxScopedArray< T >

  A scoped array template class.
- class wxScopedPtr< T >

  A scoped pointer template class.
- class wxSharedPtr< T >

  A smart pointer with non-intrusive reference counting.
- class wxWeakRefDynamic< T >

  wxWeakRefDynamic<T> is a template class for weak references that is used in the same way as wxWeakRef<T>.
- class wxWeakRef< T >

  wxWeakRef<T> is a template class for weak references to wxWidgets objects, such as wxEvtHandler, wxWindow and wxObject.
- class wxWindowPtr< T >

  A reference-counted smart pointer for holding wxWindow instances.
- class wxScopedPtr

  This is a simple scoped smart pointer implementation that is similar to the Boost smart pointers (see http://www.boost.org) but rewritten to use macros instead.
- class wxScopedTiedPtr

  This is a variation on the topic of wxScopedPtr.
- class wxTrackable

  Add-on base class for a trackable object.

20.53 Streams

20.53.1 Detailed Description

wxWidgets has its own set of stream classes, as an alternative to often buggy standard stream libraries, and to provide enhanced functionality.

Related overviews: Stream Classes Overview

Classes

- class wxFSInputStream

  Input stream for virtual file stream files.
- class wxArchiveInputStream

  This is an abstract base class which serves as a common interface to archive input streams such as wxZipInputStream.
- class wxArchiveOutputStream

  This is an abstract base class which serves as a common interface to archive output streams such as wxZipOutputStream.
- class wxArchiveEntry

  This is an abstract base class which serves as a common interface to archive entry classes such as wxZipEntry.
- class wxArchiveClassFactory

  Allows the creation of streams to handle archive formats such as zip and tar.
- class wxArchiveNotifier

  If you need to know when a wxArchiveInputStream updates a wxArchiveEntry object, you can create a notifier by deriving from this abstract base class, overriding wxArchiveNotifier::OnEntryUpdated.
- class wxArchiveIterator

  An input iterator template class that can be used to transfer an archive's catalogue to a container.
- class wxDataOutputStream

  This class provides functions that write binary data types in a portable way.
- class wxDataInputStream

  This class provides functions that read binary data types in a portable way.
- class wxMemoryOutputStream

  This class allows to use all methods taking a wxOutputStream reference to write to in-memory data.
- class wxMemoryInputStream

  This class allows to use all methods taking a wxInputStream reference to read in-memory data.
- class wxSocketOutputStream

  This class implements an output stream which writes data from a connected socket.
- class wxSocketInputStream

  This class implements an input stream which reads data from a connected socket.
- class wxStringInputStream

  This class implements an input stream which reads data from a string.
- class wxStringOutputStream

  This class implements an output stream which writes data either to a user-provided or internally allocated string.
- class wxStdInputStreamBuffer

  wxStdInputStreamBuffer is a std::streambuf derived stream buffer which reads from a wxInputStream.
- class wxStdInputStream

  wxStdInputStream is a std::istream derived stream which reads from a wxInputStream.
- class wxStdOutputStreamBuffer

  wxStdOutputStreamBuffer is a std::streambuf derived stream buffer which writes to a wxOutputStream.
- class wxStdOutputStream

  wxStdOutputStream is a std::ostream derived stream which writes to a wxOutputStream.

- class wxStreamBase

  This class is the base class of most stream related classes in wxWidgets.
- class wxStreamBuffer

  wxStreamBuffer is a cache manager for wxStreamBase: it manages a stream buffer linked to a stream.
- class wxOutputStream

  wxOutputStream is an abstract base class which may not be used directly.
- class wxInputStream

  wxInputStream is an abstract base class which may not be used directly.
- class wxCountingOutputStream

  wxCountingOutputStream is a specialized output stream which does not write any data anywhere, instead it counts how many bytes would get written if this were a normal stream.
- class wxBufferedInputStream

  This stream acts as a cache.
- class wxFilterClassFactory

  Allows the creation of filter streams to handle compression formats such as gzip and bzip2.
- class wxFilterOutputStream

  A filter stream has the capability of a normal stream but it can be placed on top of another stream.
- class wxFilterInputStream

  A filter stream has the capability of a normal stream but it can be placed on top of another stream.
- class wxBufferedOutputStream

  This stream acts as a cache.
- class wxWrapperInputStream

  A wrapper input stream is a kind of filter stream which forwards all the operations to its base stream.
- class wxTarInputStream

  Input stream for reading tar files.
- class wxTarClassFactory

  Class factory for the tar archive format.
- class wxTarOutputStream

  Output stream for writing tar files.
- class wxTarEntry

  Holds the meta-data for an entry in a tar.
- class wxTextInputStream

  This class provides functions that reads text data using an input stream, allowing you to read text, floats, and integers.
- class wxTextOutputStream

  This class provides functions that write text data using an output stream, allowing you to write text, floats, and integers.
- class wxTempFileOutputStream

  wxTempFileOutputStream is an output stream based on wxTempFile.
- class wxFFileOutputStream

  This class represents data written to a file.
- class wxFileOutputStream

  This class represents data written to a file.
- class wxFileInputStream

  This class represents data read in from a file.
- class wxFFileInputStream

  This class represents data read in from a file.
- class wxFFileStream

  This stream allows to both read from and write to a file using buffered STDIO functions.
- class wxFileStream

  This class represents data that can be both read from and written to a file.
- class wxZipNotifier

If you need to know when a wxZipInputStream updates a wxZipEntry, you can create a notifier by deriving from this abstract base class, overriding wxZipNotifier::OnEntryUpdated().

- class wxZipEntry

  Holds the meta-data for an entry in a zip.

- class wxZipInputStream

  Input stream for reading zip files.

- class wxZipClassFactory

  Class factory for the zip archive format.

- class wxZipOutputStream

  Output stream for writing zip files.

- class wxZlibOutputStream

  This stream compresses all data written to it.

- class wxZlibInputStream

  This filter stream decompresses a stream that is in zlib or gzip format.

20.54 Strings

20.54.1 Detailed Description

Global string functions and macros.

See wxString for the wxWidgets string class.

Please note that all functions of this group which are documented to take `char*` arrays are overloaded with `wchar_t*` variants.

Note also that wxWidgets wraps all standard CRT functions, even if the wrappers are not (all) documented.

Macros

- #define wxT(string)

  This macro can be used with character and string literals (in other words, `'x'` or `"foo"`) to automatically convert them to wide strings in Unicode builds of wxWidgets.
- #define wxT_2(string)

  Compatibility macro which expands to wxT() in wxWidgets 2 only.
- #define wxS(string)

  wxS is a macro which can be used with character and string literals (in other words, `'x'` or `"foo"`) to convert them either to wide characters or wide strings in `wchar_t`-based (UTF-16) builds, or to keep them unchanged in `char`-based (UTF-8) builds.
- #define _T(string)

  This macro is exactly the same as wxT() and is defined in wxWidgets simply because it may be more intuitive for Windows programmers as the standard Win32 headers also define it (as well as yet another name for the same macro which is _TEXT()).
- #define wxPLURAL(string, plural, n)

  This macro is identical to _() but for the plural variant of wxGetTranslation().
- #define wxTRANSLATE(string)

  This macro doesn't do anything in the program code – it simply expands to the value of its argument.

Typedefs

- typedef wxUSE_UNICODE_dependent wxChar

  wxChar is defined to be
  - `char` *when* `wxUSE_UNICODE==0`
  - `wchar_t` *when* `wxUSE_UNICODE==1` *(the default).*
- typedef wxUSE_UNICODE_dependent wxSChar

  wxSChar is defined to be
  - `signed char` *when* `wxUSE_UNICODE==0`
  - `wchar_t` *when* `wxUSE_UNICODE==1` *(the default).*
- typedef wxUSE_UNICODE_dependent wxUChar

  wxUChar is defined to be
  - `unsigned char` *when* `wxUSE_UNICODE==0`
  - `wchar_t` *when* `wxUSE_UNICODE==1` *(the default).*
- typedef wxUSE_UNICODE_WCHAR_dependent wxStringCharType

  wxStringCharType is defined to be:
  - `char` *when* `wxUSE_UNICODE==0`
  - `char` *when* `wxUSE_UNICODE_WCHAR==0` *and* `wxUSE_UNICODE==1`
  - `wchar_t` *when* `wxUSE_UNICODE_WCHAR==1` *and* `wxUSE_UNICODE==1`

Functions

- wxArrayString wxSplit (const wxString &str, const wxChar sep, const wxChar escape= '\\')

  Splits the given wxString object using the separator sep and returns the result as a wxArrayString.

- wxString wxJoin (const wxArrayString &arr, const wxChar sep, const wxChar escape= '\\')

  Concatenate all lines of the given wxArrayString object using the separator sep and returns the result as a wxString.

- template<bool(T)(const wxUniChar &c) >
  bool wxStringCheck (const wxString &val)

  Allows to extend a function with the signature:

- wxArrayString wxStringTokenize (const wxString &str, const wxString &delims=wxDEFAULT_DELIMITERS, wxStringTokenizerMode mode=wxTOKEN_DEFAULT)

  This is a convenience function wrapping wxStringTokenizer which simply returns all tokens found in the given str as an array.

- const wxString & wxGetTranslation (const wxString &string, const wxString &domain=wxEmptyString)

  This function returns the translation of string in the current `locale()`.

- const wxString & wxGetTranslation (const wxString &string, const wxString &plural, unsigned n, const wxString &domain=wxEmptyString)

  This is an overloaded version of wxGetTranslation(const wxString&, const wxString&), please see its documentation for general information.

- const wxString & _ (const wxString &string)

  Macro to be used around all literal strings that should be translated.

20.54.2 Macro Definition Documentation

#define _T(*string*)

This macro is exactly the same as wxT() and is defined in wxWidgets simply because it may be more intuitive for Windows programmers as the standard Win32 headers also define it (as well as yet another name for the same macro which is _TEXT()).

Don't confuse this macro with _()!

Note that since wxWidgets 2.9.0 the use of _T() is discouraged just like for wxT() and also that this macro may conflict with identifiers defined in standard headers of some compilers (such as Sun CC) so its use should really be avoided.

Include file:

```
#include <wx/chartype.h>
```

#define wxPLURAL(*string, plural, n*)

This macro is identical to _() but for the plural variant of wxGetTranslation().

Returns

A const wxString.

Include file:

```
#include <wx/intl.h>
```

#define wxS(*string*)

wxS is a macro which can be used with character and string literals (in other words, `'x'` or `"foo"`) to convert them either to wide characters or wide strings in `wchar_t-based` (UTF-16) builds, or to keep them unchanged in `char-based` (UTF-8) builds.

Basically this macro produces characters or strings of type wxStringCharType.

The use of this macro is optional as the translation will always be done at run-time even if there is a mismatch between the kind of the literal used and the string or character type used in the current build. However using it can be beneficial in **performance-sensitive code** to do the conversion at compile-time instead.

See also

Unicode Support in wxWidgets, wxT()

Include file:

```
#include <wx/chartype.h>
```

#define wxT(*string*)

This macro can be used with character and string literals (in other words, `'x'` or `"foo"`) to automatically convert them to wide strings in Unicode builds of wxWidgets.

This macro simply returns the value passed to it without changes in ASCII build. In fact, its definition is:

```
1 #ifdef UNICODE
2 #   define wxT(x)  L##x
3 #else // !Unicode
4 #   define wxT(x)  x
5 #endif
```

Note that since wxWidgets 2.9.0 you shouldn't use wxT() anymore in your program sources (it was previously required if you wanted to support Unicode).

See also

Unicode Support in wxWidgets, wxS()

Include file:

```
#include <wx/chartype.h>
```

#define wxT_2(*string*)

Compatibility macro which expands to wxT() in wxWidgets 2 only.

This macro can be used in code which needs to compile with both wxWidgets 2 and 3 versions, in places where the wx2 API requires a Unicode string (in Unicode build) but the wx3 API only accepts a standard narrow string, as in e.g. wxCmdLineEntryDesc structure objects initializers.

Example of use:

```
1 const wxCmdLineEntryDesc cmdLineDesc[] =
2 {
3     { wxCMD_LINE_SWITCH, wxT_2("q"), wxT_2("quiet"),
4       wxT_2("Don't output verbose messages") },
5     wxCMD_LINE_DESC_END
6 };
```

Without `wxT_2` the code above wouldn't compile with wxWidgets 2, but using `wxT` instead, it wouldn't compile with wxWidgets 3.

See also

wxT()

Since

2.8.12, 2.9.2

Include file:

```
#include <wx/chartype.h>
```

#define wxTRANSLATE(*string*)

This macro doesn't do anything in the program code – it simply expands to the value of its argument.

However it does have a purpose which is to mark the literal strings for the extraction into the message catalog created by `xgettext` program. Usually this is achieved using _() but that macro not only marks the string for extraction but also expands into a wxGetTranslation() call which means that it cannot be used in some situations, notably for static array initialization.

Here is an example which should make it more clear: suppose that you have a static array of strings containing the weekday names and which have to be translated (note that it is a bad example, really, as wxDateTime already can be used to get the localized week day names already). If you write:

```
1 static const char * const weekdays[] = { _("Mon"), ..., _("Sun") };
2 ...
3 // use weekdays[n] as usual
```

The code wouldn't compile because the function calls are forbidden in the array initializer. So instead you should do this:

```
1 static const char * const weekdays[] = { wxTRANSLATE("Mon"), ...,
2 wxTRANSLATE("Sun") };
3 ...
4 // use wxGetTranslation(weekdays[n])
```

Note that although the code **would** compile if you simply omit wxTRANSLATE() in the above, it wouldn't work as expected because there would be no translations for the weekday names in the program message catalog and wxGetTranslation() wouldn't find them.

Returns

A const wxChar*.

Include file:

```
#include <wx/intl.h>
```

20.54.3 Typedef Documentation

typedef wxUSE_UNICODE_dependent wxChar

wxChar is defined to be

- `char` when `wxUSE_UNICODE==0`
- `wchar_t` when `wxUSE_UNICODE==1` (the default).

typedef wxUSE_UNICODE_dependent wxSChar

wxSChar is defined to be

- `signed char` when `wxUSE_UNICODE==0`
- `wchar_t` when `wxUSE_UNICODE==1` (the default).

typedef wxUSE_UNICODE_WCHAR_dependent wxStringCharType

wxStringCharType is defined to be:

- `char` when `wxUSE_UNICODE==0`
- `char` when `wxUSE_UNICODE_WCHAR==0` and `wxUSE_UNICODE==1`
- `wchar_t` when `wxUSE_UNICODE_WCHAR==1` and `wxUSE_UNICODE==1`

The `wxUSE_UNICODE_WCHAR` symbol is defined to `1` when building on Windows while it's defined to `0` when building on Unix, Linux or OS X. (Note that `wxUSE_UNICODE_UTF8` symbol is defined as the opposite of `wxUSE_UNICODE_WCHAR`.)

Note that wxStringCharType (as the name says) is the type used by wxString for internal storage of the characters.

typedef wxUSE_UNICODE_dependent wxUChar

wxUChar is defined to be

- `unsigned char` when `wxUSE_UNICODE==0`
- `wchar_t` when `wxUSE_UNICODE==1` (the default).

20.54.4 Function Documentation

const wxString& _ (const wxString & *string*)

Macro to be used around all literal strings that should be translated.

This macro expands into a call to wxGetTranslation(), so it marks the message for the extraction by `xgettext` just as wxTRANSLATE() does, but also returns the translation of the string for the current locale during execution.

This macro is thread-safe.

Include file:

```
#include <wx/intl.h>
```

const wxString& wxGetTranslation (const wxString & *string,* const wxString & *domain* = wxEmptyString)

This function returns the translation of *string* in the current `locale()`.

If the string is not found in any of the loaded message catalogs (see Internationalization), the original string is returned. If you enable logging of trace messages with "i18n" mask (using wxLog::AddTraceMask()) and debug logging is enabled (see Debugging), a message is also logged in this case – which helps to find the strings which were not yet translated.

If *domain* is specified then only that domain/catalog is searched for a matching string. As this function is used very often, an alternative (and also common in Unix world) syntax is provided: the _() macro is defined to do the same thing as wxGetTranslation().

This function is thread-safe.

Note

This function is not suitable for literal strings using wxT() macro since this macro is not recognised by `xgettext`, and so such strings are not extracted to the message catalog. Instead, use the _() and wxPLURAL() macro for all literal strings.

See also

wxGetTranslation(const wxString&, const wxString&, unsigned, const wxString&)

Include file:

```
#include <wx/intl.h>
```

const wxString& wxGetTranslation (const wxString & *string,* const wxString & *plural,* unsigned *n,* const wxString & *domain* = wxEmptyString)

This is an overloaded version of wxGetTranslation(const wxString&, const wxString&), please see its documentation for general information.

This version is used when retrieving translation of string that has different singular and plural forms in English or different plural forms in some other language. Like wxGetTranslation(const wxString&,const wxString&), the *string* parameter must contain the singular form of the string to be converted and is used as the key for the search in the catalog. The *plural* parameter is the plural form (in English). The parameter *n* is used to determine the plural form. If no message catalog is found, *string* is returned if "n == 1", otherwise *plural* is returned.

See GNU gettext Manual for additional information on plural forms handling: http://www.gnu.org/software/gettext/manual/gettext.html#Plural-forms For a shorter alternative see the wxPLURAL() macro.

This function is thread-safe.

Include file:

```
#include <wx/intl.h>
```

wxString wxJoin (const wxArrayString & *arr,* const wxChar *sep,* const wxChar *escape* = '\\')

Concatenate all lines of the given wxArrayString object using the separator *sep* and returns the result as a wxString.

If the *escape* character is non-NULL, then it's used as prefix for each occurrence of *sep* in the strings contained in *arr* before joining them which is necessary in order to be able to recover the original array contents from the string later using wxSplit().

See also

wxSplit()

Include file:

```
#include <wx/arrstr.h>
```

wxArrayString wxSplit (const wxString & *str,* const wxChar *sep,* const wxChar *escape* = '\\')

Splits the given wxString object using the separator *sep* and returns the result as a wxArrayString.

If the *escape* character is non-NULL, then the occurrences of *sep* immediately prefixed with *escape* are not considered as separators. Note that empty tokens will be generated if there are two or more adjacent separators.

See also

wxJoin()

Include file:

```
#include <wx/arrstr.h>
```

template<bool(T)(const wxUniChar &c) > bool wxStringCheck (const wxString & *val*) [inline]

Allows to extend a function with the signature:

```
1 bool SomeFunc(const wxUniChar& c)
```

which operates on a single character, to an entire wxString.

E.g. if you want to check if an entire string contains only digits, you can do:

```
1 if (wxStringCheck<wxIsdigit>(myString))
2     ... // the entire string contains only digits!
3 else
4     ... // at least one character of myString is not a digit
```

Returns

true if the given function returns a non-zero value for all characters of the *val* string.

wxArrayString wxStringTokenize (const wxString & *str,* const wxString & *delims* = wxDEFAULT_DELIMITERS, wxStringTokenizerMode *mode* = wxTOKEN_DEFAULT)

This is a convenience function wrapping wxStringTokenizer which simply returns all tokens found in the given *str* as an array.

Please see wxStringTokenizer::wxStringTokenizer for the description of the other parameters.

Returns

The array with the parsed tokens.

Include file:

```
#include <wx/tokenzr.h>
```

20.55 Text Conversion

20.55.1 Detailed Description

These are the classes used for conversions between different text encodings.

Classes

- class wxEncodingConverter

  This class is capable of converting strings between two 8-bit encodings/charsets.
- class wxMBConv

  This class is the base class of a hierarchy of classes capable of converting text strings between multibyte (SBCS or DBCS) encodings and Unicode.
- class wxMBConvUTF7

  This class converts between the UTF-7 encoding and Unicode.
- class wxMBConvUTF8

  This class converts between the UTF-8 encoding and Unicode.
- class wxMBConvUTF16

  This class is used to convert between multibyte encodings and UTF-16 Unicode encoding (also known as UCS-2).
- class wxMBConvUTF32

  This class is used to convert between multibyte encodings and UTF-32 Unicode encoding (also known as UCS-4).
- class wxCSConv

  This class converts between any character set supported by the system and Unicode.

Variables

- wxMBConv * wxConvFileName

  Conversion object used for converting file names from their external representation to the one used inside the program.

20.55.2 Variable Documentation

wxMBConv* wxConvFileName

Conversion object used for converting file names from their external representation to the one used inside the program.

wxConvFileName converts filenames between filesystem multibyte encoding and Unicode. **wxConvFileName** can also be set to a something else at run-time which is used e.g. by wxGTK to use an object which checks the environment variable **G_FILESYSTEM_ENCODING** indicating that filenames should not be interpreted as UTF8 and also for converting invalid UTF8 characters (e.g. if there is a filename in iso8859_1) to strings with octal values.

Since some platforms (such as Win32) use Unicode in the filenames, and others (such as Unix) use multibyte encodings, this object should only be used directly if wxMBFILES is defined to 1. A convenience macro, `wxFNCONV`, is defined to `wxConvFileName->cWX2MB` in this case. You could use it like this:

```
1 wxChar *name = "rawfile.doc";
2 FILE *fil = fopen(wxFNCONV(name), "r");
```

(although it would be better to just use wxFopen(name, "r") in this particular case, you only need to use this object for functions taking file names not wrapped by wxWidgets.)

Library: wxBase

Category: Text Conversion

See also

wxMBConv Overview

20.56 Threading

20.56.1 Detailed Description

wxWidgets provides a set of classes to make use of the native thread capabilities of the various platforms.

Related Overviews: Multithreading Overview

Related macros/global-functions group: Threads

Classes

- class wxMessageQueue< T >

  wxMessageQueue allows passing messages between threads.
- class wxThreadEvent

  This class adds some simple functionality to wxEvent to facilitate inter-thread communication.
- class wxCondition

  wxCondition variables correspond to pthread conditions or to Win32 event objects.
- class wxCriticalSectionLocker

  This is a small helper class to be used with wxCriticalSection objects.
- class wxThreadHelper

  The wxThreadHelper class is a mix-in class that manages a single background thread, either detached or joinable (see wxThread for the differences).
- class wxCriticalSection

  A critical section object is used for exactly the same purpose as a wxMutex.
- class wxThread

  A thread is basically a path of execution through a program.
- class wxSemaphore

  wxSemaphore is a counter limiting the number of threads concurrently accessing a shared resource.
- class wxMutexLocker

  This is a small helper class to be used with wxMutex objects.
- class wxMutex

  A mutex object is a synchronization object whose state is set to signaled when it is not owned by any thread, and nonsignaled when it is owned.

Enumerations

- enum wxMessageQueueError {
  wxMSGQUEUE_NO_ERROR = 0,
  wxMSGQUEUE_TIMEOUT,
  wxMSGQUEUE_MISC_ERROR }

  Error codes for wxMessageQueue<> operations.

20.56.2 Enumeration Type Documentation

enum wxMessageQueueError

Error codes for wxMessageQueue<> operations.

This enum contains the possible return value of wxMessageQueue<> methods.

Since

2.9.0

Category: Threading

Enumerator

wxMSGQUEUE_NO_ERROR Indicates that the operation completed successfully.

wxMSGQUEUE_TIMEOUT Indicates that no messages were received before timeout expired. This return value is only used by wxMessageQueue<>::ReceiveTimeout().

wxMSGQUEUE_MISC_ERROR Some unexpected (and fatal) error has occurred.

20.57 Threads

20.57.1 Detailed Description

The functions and macros here mainly exist to make it possible to write code which may be compiled in multi thread build (wxUSE_THREADS = 1) as well as in single thread configuration (wxUSE_THREADS = 0).

For example, a static variable must be protected against simultaneous access by multiple threads in the former configuration but in the latter the extra overhead of using the critical section is not needed. To solve this problem, the wxCRITICAL_SECTION() macro may be used to create and use the critical section only when needed.

See also

wxThread, wxMutex, Multithreading Overview

Related class group: Threading

Macros

- #define wxCRIT_SECT_DECLARE(cs)

  This macro declares a (static) critical section object named cs if `wxUSE_THREADS` is 1 and does nothing if it is 0.
- #define wxCRIT_SECT_DECLARE_MEMBER(cs)

  This macro declares a critical section object named cs if `wxUSE_THREADS` is 1 and does nothing if it is 0.
- #define wxCRIT_SECT_LOCKER(name, cs)

  This macro creates a wxCriticalSectionLocker named name and associated with the critical section cs if `wxUSE_THREADS` is 1 and does nothing if it is 0.
- #define wxCRITICAL_SECTION(name)

  This macro combines wxCRIT_SECT_DECLARE() and wxCRIT_SECT_LOCKER(): it creates a static critical section object and also the lock object associated with it.
- #define wxLEAVE_CRIT_SECT(critical_section)

  This macro is equivalent to critical_section.Leave() if `wxUSE_THREADS` is 1 and does nothing if it is 0.
- #define wxENTER_CRIT_SECT(critical_section)

  This macro is equivalent to critical_section.Enter() if `wxUSE_THREADS` is 1 and does nothing if it is 0.

Functions

- bool wxIsMainThread ()

  Returns true if this thread is the main one.
- void wxMutexGuiEnter ()

  This function must be called when any thread other than the main GUI thread wants to get access to the GUI library.
- void wxMutexGuiLeave ()

  This function is only defined on platforms which support preemptive threads.

20.57.2 Macro Definition Documentation

#define wxCRIT_SECT_DECLARE(*cs*)

This macro declares a (static) critical section object named *cs* if `wxUSE_THREADS` is 1 and does nothing if it is 0.

Include file:

```
#include <wx/thread.h>
```

#define wxCRIT_SECT_DECLARE_MEMBER(*cs*)

This macro declares a critical section object named *cs* if `wxUSE_THREADS` is 1 and does nothing if it is 0.

As it doesn't include the `static` keyword (unlike wxCRIT_SECT_DECLARE()), it can be used to declare a class or struct member which explains its name.

Include file:

```
#include <wx/thread.h>
```

#define wxCRIT_SECT_LOCKER(*name, cs*)

This macro creates a wxCriticalSectionLocker named *name* and associated with the critical section *cs* if `wxUSE_THREADS` is 1 and does nothing if it is 0.

Include file:

```
#include <wx/thread.h>
```

#define wxCRITICAL_SECTION(*name*)

This macro combines wxCRIT_SECT_DECLARE() and wxCRIT_SECT_LOCKER(): it creates a static critical section object and also the lock object associated with it.

Because of this, it can be only used inside a function, not at global scope. For example:

```
1 int IncCount()
2 {
3     static int s_counter = 0;
4
5     wxCRITICAL_SECTION(counter);
6
7     return ++s_counter;
8 }
```

Note that this example assumes that the function is called the first time from the main thread so that the critical section object is initialized correctly by the time other threads start calling it, if this is not the case this approach can **not** be used and the critical section must be made a global instead.

Include file:

```
#include <wx/thread.h>
```

#define wxENTER_CRIT_SECT(*critical_section*)

This macro is equivalent to critical_section.Enter() if `wxUSE_THREADS` is 1 and does nothing if it is 0.

Include file:

```
#include <wx/thread.h>
```

#define wxLEAVE_CRIT_SECT(*critical_section*)

This macro is equivalent to critical_section.Leave() if `wxUSE_THREADS` is 1 and does nothing if it is 0.

Include file:

```
#include <wx/thread.h>
```

20.57.3 Function Documentation

bool wxIsMainThread ( )

Returns true if this thread is the main one.

Always returns true if `wxUSE_THREADS` is 0.

Include file:

```
#include <wx/thread.h>
```

void wxMutexGuiEnter ( )

This function must be called when any thread other than the main GUI thread wants to get access to the GUI library.

This function will block the execution of the calling thread until the main thread (or any other thread holding the main GUI lock) leaves the GUI library and no other thread will enter the GUI library until the calling thread calls wxMutexGuiLeave().

Typically, these functions are used like this:

```
1 void MyThread::Foo(void)
2 {
3     // before doing any GUI calls we must ensure that
4     // this thread is the only one doing it!
5
6     wxMutexGuiEnter();
7
8     // Call GUI here:
9     my_window->DrawSomething();
10
11     wxMutexGuiLeave();
12 }
```

This function is only defined on platforms which support preemptive threads and only works under some ports (wxMSW currently).

Note

Under GTK, no creation of top-level windows is allowed in any thread but the main one.

Include file:

```
#include <wx/thread.h>
```

void wxMutexGuiLeave ( )

This function is only defined on platforms which support preemptive threads.

See also

wxMutexGuiEnter()

Include file:

```
#include <wx/thread.h>
```

20.58 Time

20.58.1 Detailed Description

The functions in this section deal with getting the current time and sleeping for the specified time interval.

Functions

- int wxGetTimeZone ()

  Returns the difference between UTC and local time in seconds.
- long wxGetLocalTime ()

  Returns the number of seconds since local time 00:00:00 Jan 1st 1970.
- wxLongLong wxGetLocalTimeMillis ()

  Returns the number of milliseconds since local time 00:00:00 Jan 1st 1970.
- long wxGetUTCTime ()

  Returns the number of seconds since GMT 00:00:00 Jan 1st 1970.
- wxLongLong wxGetUTCTimeMillis ()

  Returns the number of milliseconds since GMT 00:00:00 Jan 1st 1970.
- wxLongLong wxGetUTCTimeUSec ()

  Returns the number of microseconds since GMT 00:00:00 Jan 1st 1970.
- void wxMicroSleep (unsigned long microseconds)

  Sleeps for the specified number of microseconds.
- void wxMilliSleep (unsigned long milliseconds)

  Sleeps for the specified number of milliseconds.
- wxString wxNow ()

  Returns a string representing the current date and time.
- void wxSleep (int secs)

  Sleeps for the specified number of seconds.
- void wxUsleep (unsigned long milliseconds)

20.58.2 Function Documentation

long wxGetLocalTime ( )

Returns the number of seconds since local time 00:00:00 Jan 1st 1970.

See also

wxDateTime::Now()

Include file:

```
#include <wx/time.h>
```

wxLongLong wxGetLocalTimeMillis ( )

Returns the number of milliseconds since local time 00:00:00 Jan 1st 1970.

The use of wxGetUTCTimeMillis() is preferred as it provides a usually (except for changes to the system time) monotonic clock which the local time also changes whenever DST begins or ends.

See also

wxDateTime::Now(), wxGetUTCTimeMillis(), wxGetUTCTimeUSec()

Include file:

```
#include <wx/time.h>
```

int wxGetTimeZone ( )

Returns the difference between UTC and local time in seconds.
Include file:

```
#include <wx/time.h>
```

long wxGetUTCTime ( )

Returns the number of seconds since GMT 00:00:00 Jan 1st 1970.

See also

wxDateTime::Now()

Include file:

```
#include <wx/time.h>
```

wxLongLong wxGetUTCTimeMillis ( )

Returns the number of milliseconds since GMT 00:00:00 Jan 1st 1970.
Include file:

```
#include <wx/time.h>
```

Since

2.9.3

wxLongLong wxGetUTCTimeUSec ( )

Returns the number of microseconds since GMT 00:00:00 Jan 1st 1970.
Include file:

```
#include <wx/time.h>
```

Since

2.9.3

void wxMicroSleep (unsigned long *microseconds*)

Sleeps for the specified number of microseconds.

The microsecond resolution may not, in fact, be available on all platforms (currently only Unix platforms with nanosleep(2) may provide it) in which case this is the same as calling wxMilliSleep() with the argument of *microseconds/1000*.

Include file:

```
#include <wx/utils.h>
```

void wxMilliSleep (unsigned long *milliseconds*)

Sleeps for the specified number of milliseconds.

Notice that usage of this function is encouraged instead of calling usleep(3) directly because the standard *usleep()* function is not MT safe.

Include file:

```
#include <wx/utils.h>
```

wxString wxNow ( )

Returns a string representing the current date and time.

Include file:

```
#include <wx/utils.h>
```

void wxSleep (int *secs*)

Sleeps for the specified number of seconds.

Include file:

```
#include <wx/utils.h>
```

void wxUsleep (unsigned long *milliseconds*)

Deprecated This function is deprecated because its name is misleading: notice that the argument is in milliseconds, not microseconds. Please use either wxMilliSleep() or wxMicroSleep() depending on the resolution you need.

Sleeps for the specified number of milliseconds.

Include file:

```
#include <wx/utils.h>
```

20.59 Validators

20.59.1 Detailed Description

These are the window validators, used for filtering and validating user input.

Related Overviews: wxValidator Overview

Classes

- class wxNumValidator< T >

  wxNumValidator is the common base class for numeric validator classes.
- class wxIntegerValidator< T >

  Validator for text entries used for integer entry.
- class wxFloatingPointValidator< T >

  Validator for text entries used for floating point numbers entry.
- class wxGenericValidator

  wxGenericValidator performs data transfer (but not validation or filtering) for many type of controls.
- class wxValidator

  wxValidator is the base class for a family of validator classes that mediate between a class of control, and application data.
- class wxTextValidator

  wxTextValidator validates text controls, providing a variety of filtering behaviours.

Enumerations

- enum wxNumValidatorStyle {
  wxNUM_VAL_DEFAULT = 0,
  wxNUM_VAL_THOUSANDS_SEPARATOR = 1,
  wxNUM_VAL_ZERO_AS_BLANK = 2,
  wxNUM_VAL_NO_TRAILING_ZEROES }

  Bit masks used for numeric validator styles.

20.59.2 Enumeration Type Documentation

enum wxNumValidatorStyle

Bit masks used for numeric validator styles.

A combination of these flags can be used when creating wxIntegerValidator and wxFloatingPointValidator objects and with their SetStyle() methods.

Since

2.9.2

Category: Validators

Enumerator

wxNUM_VAL_DEFAULT Indicates absence of any other flags. This value corresponds to the default behaviour.

wxNUM_VAL_THOUSANDS_SEPARATOR Use thousands separators in the numbers. When this style is used, numbers are formatted using the thousands separators after validating the user entry (if the current locale uses the thousands separators character).

wxNUM_VAL_ZERO_AS_BLANK Show a value of zero as an empty string. With this style a value of zero in the associated variable is translated to an empty string and an empty value of the control is translated to a value of zero.

wxNUM_VAL_NO_TRAILING_ZEROES Remove trailing zeroes from the fractional part of the number. This style can only be used with wxFloatingPointValidator and indicates that trailing zeroes should be removed from the control text when it is validated. By default, as many zeroes as needed to satisfy the precision used when creating the validator will be appended.

For example, without this style a wxFloatingPointValidator with a precision 3 will show the value of 1.5 as "1.500" after validation. With this style, the value will be shown as just "1.5" (while a value of e.g. 1.567 will still be shown with all the three significant digits, of course).

20.60 Versioning

20.60.1 Detailed Description

The following constants are defined in wxWidgets:

wxMAJOR_VERSION	The major version of wxWidgets
wxMINOR_VERSION	The minor version of wxWidgets
wxRELEASE_NUMBER	The release number
wxSUBRELEASE_NUMBER	The subrelease number which is 0 for all official releases

For example, the values or these constants for wxWidgets 2.8.7 are 2, 8, 7 and 0.

Additionally, wxVERSION_STRING is a user-readable string containing the full wxWidgets version and wxVERSION_NUMBER is a combination of the three version numbers above: for 2.1.15, it is 2115 and it is 2200 for wxWidgets 2.2.

The subrelease number is only used for the sources in between official releases and so normally is not useful.

Include file:

```
#include <wx/version.h>
```

Macros

- #define wxCHECK_GCC_VERSION(major, minor)

  Returns true if the compiler being used is GNU C++ and its version is at least major.minor or greater.
- #define wxCHECK_SUNCC_VERSION(major, minor)

  Returns true if the compiler being used is Sun CC Pro and its version is at least major.minor or greater.
- #define wxCHECK_VISUALC_VERSION(major)

  Returns true if the compiler being used is Visual C++ and its version is at least major or greater.
- #define wxCHECK_W32API_VERSION(major, minor)

  Returns true if the version of w32api headers used is major.minor or greater.
- #define wxCHECK_VERSION(major, minor, release)

  This is a macro which evaluates to true if the current wxWidgets version is at least major.minor.release.
- #define wxCHECK_VERSION_FULL(major, minor, release, subrel)

  Same as wxCHECK_VERSION() but also checks that wxSUBRELEASE_NUMBER is at least subrel.

Functions

- wxVersionInfo wxGetLibraryVersionInfo ()

  Get wxWidgets version information.

20.60.2 Macro Definition Documentation

#define wxCHECK_GCC_VERSION(*major, minor*)

Returns true if the compiler being used is GNU C++ and its version is at least major.minor or greater.

Returns false otherwise.

Include file:

```
#include <wx/platform.h>
```

#define wxCHECK_SUNCC_VERSION(*major, minor*)

Returns true if the compiler being used is Sun CC Pro and its version is at least major.minor or greater.

Returns false otherwise.

Include file:

```
#include <wx/platform.h>
```

#define wxCHECK_VERSION(*major, minor, release*)

This is a macro which evaluates to true if the current wxWidgets version is at least major.minor.release.

For example, to test if the program is compiled with wxWidgets 2.2 or higher, the following can be done:

```
1     wxString s;
2 #if wxCHECK_VERSION(2, 2, 0)
3     if ( s.StartsWith("foo") )
4 #else // replacement code for old version
5     if ( strncmp(s, "foo", 3) == 0 )
6 #endif
7     {
8         ...
9     }
```

Include file:

```
#include <wx/version.h>
```

#define wxCHECK_VERSION_FULL(*major, minor, release, subrel*)

Same as wxCHECK_VERSION() but also checks that wxSUBRELEASE_NUMBER is at least subrel.

Include file:

```
#include <wx/version.h>
```

#define wxCHECK_VISUALC_VERSION(*major*)

Returns true if the compiler being used is Visual C++ and its version is at least major or greater.

Returns false otherwise.

Include file:

```
#include <wx/platform.h>
```

#define wxCHECK_W32API_VERSION(*major, minor*)

Returns true if the version of w32api headers used is major.minor or greater.

Otherwise, and also if we are not compiling with MinGW32/Cygwin under Win32 at all, returns false.

Include file:

```
#include <wx/platform.h>
```

20.60.3 Function Documentation

wxVersionInfo wxGetLibraryVersionInfo ( )

Get wxWidgets version information.

Since

2.9.2

See also

wxVersionInfo

Include file:

```
#include <wx/utils.h>
```

Library: wxCore

20.61 Virtual File System

20.61.1 Detailed Description

wxWidgets provides a set of classes that implement an extensible virtual file system, used internally by the HTML classes.

Classes

- class wxFileSystem

  This class provides an interface for opening files on different file systems.
- class wxFSFile

  This class represents a single file opened by wxFileSystem.
- class wxFileSystemHandler

  Classes derived from wxFileSystemHandler are used to access virtual file systems.
- class wxMemoryFSHandler

  This wxFileSystem handler can store arbitrary data in memory stream and make them accessible via an URL.

20.62 WebView

20.62.1 Detailed Description

The wxWebView library is a set of classes for viewing complex web documents and for internet browsing.

It is built around a series of backends, and exposes common functions for them.

Classes

- class wxWebViewHistoryItem

  A simple class that contains the URL and title of an element of the history of a wxWebView.
- class wxWebViewFactory

  An abstract factory class for creating wxWebView backends.
- class wxWebViewHandler

  The base class for handling custom schemes in wxWebView, for example to allow virtual file system support.
- class wxWebView

  This control may be used to render web (HTML / CSS / javascript) documents.
- class wxWebViewEvent

  A navigation event holds information about events associated with wxWebView objects.
- class wxWebViewArchiveHandler

  A custom handler for the file scheme which also supports loading from archives.
- class wxWebViewFSHandler

  A wxWebView file system handler to support standard wxFileSystem protocols of the form `example:page.htm` *The handler allows wxWebView to use wxFileSystem in a similar fashion to its use with wxHtml.*

20.63 Window Docking (wxAUI)

20.63.1 Detailed Description

wxAUI is a set classes for writing a customizable application interface with built-in docking, floatable panes and a flexible MDI-like interface.

Related Overviews: wxAUI Overview

Classes

- class wxAuiDefaultTabArt

  Default art provider for wxAuiNotebook.
- class wxAuiToolBarEvent

  wxAuiToolBarEvent is used for the events generated by wxAuiToolBar.
- class wxAuiToolBarItem

  wxAuiToolBarItem is part of the wxAUI class framework, representing a toolbar element.
- class wxAuiToolBarArt

  wxAuiToolBarArt is part of the wxAUI class framework.
- class wxAuiDefaultToolBarArt

  wxAuiDefaultToolBarArt is part of the wxAUI class framework.
- class wxAuiToolBar

  wxAuiToolBar is a dockable toolbar, part of the wxAUI class framework.
- class wxAuiNotebook

  wxAuiNotebook is part of the wxAUI class framework, which represents a notebook control, managing multiple windows with associated tabs.
- class wxAuiTabContainerButton

  A simple class which holds information about wxAuiNotebook tab buttons and their state.
- class wxAuiTabContainer

  wxAuiTabContainer is a class which contains information about each tab.
- class wxAuiTabArt

  Tab art provider defines all the drawing functions used by wxAuiNotebook.
- class wxAuiSimpleTabArt

  Another standard tab art provider for wxAuiNotebook.
- class wxAuiDockArt

  wxAuiDockArt is part of the wxAUI class framework.
- class wxAuiManager

  wxAuiManager is the central class of the wxAUI class framework.
- class wxAuiPaneInfo

  wxAuiPaneInfo is part of the wxAUI class framework.
- class wxAuiManagerEvent

  Event used to indicate various actions taken with wxAuiManager.

Enumerations

- enum wxAuiToolBarStyle {
  wxAUI_TB_TEXT = 1 << 0,
  wxAUI_TB_NO_TOOLTIPS = 1 << 1,
  wxAUI_TB_NO_AUTORESIZE = 1 << 2,
  wxAUI_TB_GRIPPER = 1 << 3,
  wxAUI_TB_OVERFLOW = 1 << 4,
  wxAUI_TB_VERTICAL = 1 << 5,
  wxAUI_TB_HORZ_LAYOUT = 1 << 6,
  wxAUI_TB_HORIZONTAL = 1 << 7,
  wxAUI_TB_PLAIN_BACKGROUND = 1 << 8,
  wxAUI_TB_HORZ_TEXT = (wxAUI_TB_HORZ_LAYOUT | wxAUI_TB_TEXT),
  wxAUI_ORIENTATION_MASK = (wxAUI_TB_VERTICAL | wxAUI_TB_HORIZONTAL),
  wxAUI_TB_DEFAULT_STYLE = 0 }

  wxAuiToolBarStyle is part of the wxAUI class framework, used to define the appearance of a wxAuiToolBar.
- enum wxAuiToolBarArtSetting {
  wxAUI_TBART_SEPARATOR_SIZE = 0,
  wxAUI_TBART_GRIPPER_SIZE = 1,
  wxAUI_TBART_OVERFLOW_SIZE = 2 }

  wxAuiToolBarArtSetting
- enum wxAuiToolBarToolTextOrientation {
  wxAUI_TBTOOL_TEXT_LEFT = 0,
  wxAUI_TBTOOL_TEXT_RIGHT = 1,
  wxAUI_TBTOOL_TEXT_TOP = 2,
  wxAUI_TBTOOL_TEXT_BOTTOM = 3 }

  wxAuiToolBarToolTextOrientation
- enum wxAuiPaneDockArtSetting {
  wxAUI_DOCKART_SASH_SIZE = 0,
  wxAUI_DOCKART_CAPTION_SIZE = 1,
  wxAUI_DOCKART_GRIPPER_SIZE = 2,
  wxAUI_DOCKART_PANE_BORDER_SIZE = 3,
  wxAUI_DOCKART_PANE_BUTTON_SIZE = 4,
  wxAUI_DOCKART_BACKGROUND_COLOUR = 5,
  wxAUI_DOCKART_SASH_COLOUR = 6,
  wxAUI_DOCKART_ACTIVE_CAPTION_COLOUR = 7,
  wxAUI_DOCKART_ACTIVE_CAPTION_GRADIENT_COLOUR = 8,
  wxAUI_DOCKART_INACTIVE_CAPTION_COLOUR = 9,
  wxAUI_DOCKART_INACTIVE_CAPTION_GRADIENT_COLOUR = 10,
  wxAUI_DOCKART_ACTIVE_CAPTION_TEXT_COLOUR = 11,
  wxAUI_DOCKART_INACTIVE_CAPTION_TEXT_COLOUR = 12,
  wxAUI_DOCKART_BORDER_COLOUR = 13,
  wxAUI_DOCKART_GRIPPER_COLOUR = 14,
  wxAUI_DOCKART_CAPTION_FONT = 15,
  wxAUI_DOCKART_GRADIENT_TYPE = 16 }

  These are the possible pane dock art settings for wxAuiDefaultDockArt.

20.63.2 Enumeration Type Documentation

enum wxAuiPaneDockArtSetting

These are the possible pane dock art settings for wxAuiDefaultDockArt.

Library: wxAui

Category: Window Docking (wxAUI)

Enumerator

wxAUI_DOCKART_SASH_SIZE Customizes the sash size.

wxAUI_DOCKART_CAPTION_SIZE Customizes the caption size.

wxAUI_DOCKART_GRIPPER_SIZE Customizes the gripper size.

wxAUI_DOCKART_PANE_BORDER_SIZE Customizes the pane border size.

wxAUI_DOCKART_PANE_BUTTON_SIZE Customizes the pane button size.

wxAUI_DOCKART_BACKGROUND_COLOUR Customizes the background colour, which corresponds to the client area.

wxAUI_DOCKART_SASH_COLOUR Customizes the sash colour.

wxAUI_DOCKART_ACTIVE_CAPTION_COLOUR Customizes the active caption colour.

wxAUI_DOCKART_ACTIVE_CAPTION_GRADIENT_COLOUR Customizes the active caption gradient colour.

wxAUI_DOCKART_INACTIVE_CAPTION_COLOUR Customizes the inactive caption colour.

wxAUI_DOCKART_INACTIVE_CAPTION_GRADIENT_COLOUR Customizes the inactive gradient caption colour.

wxAUI_DOCKART_ACTIVE_CAPTION_TEXT_COLOUR Customizes the active caption text colour.

wxAUI_DOCKART_INACTIVE_CAPTION_TEXT_COLOUR Customizes the inactive caption text colour.

wxAUI_DOCKART_BORDER_COLOUR Customizes the border colour.

wxAUI_DOCKART_GRIPPER_COLOUR Customizes the gripper colour.

wxAUI_DOCKART_CAPTION_FONT Customizes the caption font.

wxAUI_DOCKART_GRADIENT_TYPE Customizes the gradient type (no gradient, vertical or horizontal)

enum wxAuiToolBarArtSetting

wxAuiToolBarArtSetting

Library: wxAui

Category: Window Docking (wxAUI)

Enumerator

wxAUI_TBART_SEPARATOR_SIZE wxAuiToolBar seperator size.

wxAUI_TBART_GRIPPER_SIZE wxAuiToolBar gripper size.

wxAUI_TBART_OVERFLOW_SIZE Overflow button size in wxAuiToolBar.

enum wxAuiToolBarStyle

wxAuiToolBarStyle is part of the wxAUI class framework, used to define the appearance of a wxAuiToolBar.

See also wxAUI Overview.

Library: wxAui

Category: Window Docking (wxAUI)

Enumerator

wxAUI_TB_TEXT Shows the text in the toolbar buttons; by default only icons are shown.

wxAUI_TB_NO_TOOLTIPS Don't show tooltips on wxAuiToolBar items.

wxAUI_TB_NO_AUTORESIZE Do not auto-resize the wxAuiToolBar.

wxAUI_TB_GRIPPER Shows a gripper on the wxAuiToolBar.

wxAUI_TB_OVERFLOW The wxAuiToolBar can contain overflow items.

wxAUI_TB_VERTICAL Using this style forces the toolbar to be vertical and be only dockable to the left or right sides of the window whereas by default it can be horizontal or vertical and be docked anywhere.

wxAUI_TB_HORZ_LAYOUT Shows the text and the icons alongside, not vertically stacked. This style must be used with wxAUI_TB_TEXT

wxAUI_TB_HORIZONTAL Analogous to wxAUI_TB_VERTICAL, but forces the toolbar to be horizontal, docking to the top or bottom of the window.

wxAUI_TB_PLAIN_BACKGROUND Draw a plain background (based on parent) instead of the default gradient background.

Since

2.9.5

wxAUI_TB_HORZ_TEXT Shows the text alongside the icons, not vertically stacked.

wxAUI_ORIENTATION_MASK Shows the text in the toolbar buttons; by default only icons are shown.

wxAUI_TB_DEFAULT_STYLE By default only icons are shown.

enum wxAuiToolBarToolTextOrientation

wxAuiToolBarToolTextOrientation

Library: wxAui

Category: Window Docking (wxAUI)

Enumerator

wxAUI_TBTOOL_TEXT_LEFT Text in wxAuiToolBar items is left aligned, currently unused/unimplemented.

wxAUI_TBTOOL_TEXT_RIGHT Text in wxAuiToolBar items is right aligned.

wxAUI_TBTOOL_TEXT_TOP Text in wxAuiToolBar items is top aligned, currently unused/unimplemented.

wxAUI_TBTOOL_TEXT_BOTTOM Text in wxAuiToolBar items is bottom aligned.

20.64 Window Layout

20.64.1 Detailed Description

wxWidgets makes window layout and sizing easy and painless using a set of classes known as "sizers".

Sizers allow for flexible window positioning and sizes that can help with automatically handling localization differences, as well as making it easy to write user resizable windows.

Related Overviews: Sizers Overview

Classes

- class wxDialogLayoutAdapter

  This abstract class is the base for classes that help wxWidgets perform run-time layout adaptation of dialogs.
- class wxGBPosition

  This class represents the position of an item in a virtual grid of rows and columns managed by a wxGridBagSizer.
- class wxGridBagSizer

  A wxSizer that can lay out items in a virtual grid like a wxFlexGridSizer but in this case explicit positioning of the items is allowed using wxGBPosition, and items can optionally span more than one row and/or column using wxGBSpan.
- class wxGBSizerItem

  The wxGBSizerItem class is used by the wxGridBagSizer for tracking the items in the sizer.
- class wxGBSpan

  This class is used to hold the row and column spanning attributes of items in a wxGridBagSizer.
- class wxLayoutAlgorithm

  wxLayoutAlgorithm implements layout of subwindows in MDI or SDI frames.
- class wxSizer

  wxSizer is the abstract base class used for laying out subwindows in a window.
- class wxStdDialogButtonSizer

  This class creates button layouts which conform to the standard button spacing and ordering defined by the platform or toolkit's user interface guidelines (if such things exist).
- class wxSizerItem

  The wxSizerItem class is used to track the position, size and other attributes of each item managed by a wxSizer.
- class wxSizerFlags

  Container for sizer items flags providing readable names for them.
- class wxFlexGridSizer

  A flex grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the wxGridSizer.
- class wxGridSizer

  A grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields having the same size, i.e.
- class wxStaticBoxSizer

  wxStaticBoxSizer is a sizer derived from wxBoxSizer but adds a static box around the sizer.
- class wxBoxSizer

  The basic idea behind a box sizer is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.
- class wxWrapSizer

  A wrap sizer lays out its items in a single line, like a box sizer – as long as there is space available in that direction.

20.65 Wrappers of CRT functions

20.65.1 Detailed Description

For documentation of these functions please refer to the documentation of the standard CRT functions (see e.g. http://www.cppreference.com/wiki/c/start).

Functions

- bool wxIsEmpty (const char *s)
- bool wxIsEmpty (const wchar_t *s)
- bool wxIsEmpty (const wxCharBuffer &s)
- bool wxIsEmpty (const wxWCharBuffer &s)
- bool wxIsEmpty (const wxString &s)
- bool wxIsEmpty (const wxCStrData &s)
- wxChar * wxTmemchr (const wxChar *s, wxChar c, size_t l)
- int wxTmemcmp (const wxChar *sz1, const wxChar *sz2, size_t len)
- wxChar * wxTmemcpy (wxChar *szOut, const wxChar *szIn, size_t len)
- wxChar * wxTmemmove (wxChar *szOut, const wxChar *szIn, size_t len)
- wxChar * wxTmemset (wxChar *szOut, const wxChar cIn, size_t len)
- char * wxTmemchr (const char *s, char c, size_t len)
- int wxTmemcmp (const char *sz1, const char *sz2, size_t len)
- char * wxTmemcpy (char *szOut, const char *szIn, size_t len)
- char * wxTmemmove (char *szOut, const char *szIn, size_t len)
- char * wxTmemset (char *szOut, const char cIn, size_t len)
- char * wxSetlocale (int category, const wxCharBuffer &locale)
- char * wxSetlocale (int category, const wxString &locale)
- char * wxSetlocale (int category, const wxCStrData &locale)
- size_t wxStrlen (const wxCharBuffer &s)
- size_t wxStrlen (const wxWCharBuffer &s)
- size_t wxStrlen (const wxString &s)
- size_t wxStrlen (const wxCStrData &s)
- size_t wxStrnlen (const char *str, size_t maxlen)
- size_t wxStrnlen (const wchar_t *str, size_t maxlen)
- char * wxStrdup (const wxCharBuffer &s)
- wchar_t * wxStrdup (const wxWCharBuffer &s)
- char * wxStrdup (const wxString &s)
- char * wxStrdup (const wxCStrData &s)
- char * wxStrcpy (char *dest, const char *src)
- wchar_t * wxStrcpy (wchar_t *dest, const wchar_t *src)
- char * wxStrcpy (char *dest, const wxString &src)
- char * wxStrcpy (char *dest, const wxCStrData &src)
- char * wxStrcpy (char *dest, const wxCharBuffer &src)
- wchar_t * wxStrcpy (wchar_t *dest, const wxString &src)
- wchar_t * wxStrcpy (wchar_t *dest, const wxCStrData &src)
- wchar_t * wxStrcpy (wchar_t *dest, const wxWCharBuffer &src)
- char * wxStrcpy (char *dest, const wchar_t *src)
- wchar_t * wxStrcpy (wchar_t *dest, const char *src)
- char * wxStrncpy (char *dest, const char *src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wchar_t *src, size_t n)
- char * wxStrncpy (char *dest, const wxString &src, size_t n)
- char * wxStrncpy (char *dest, const wxCStrData &src, size_t n)
- char * wxStrncpy (char *dest, const wxCharBuffer &src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wxString &src, size_t n)

- wchar_t * wxStrncpy (wchar_t *dest, const wxCStrData &src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wxWCharBuffer &src, size_t n)
- char * wxStrncpy (char *dest, const wchar_t *src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const char *src, size_t n)
- size_t wxStrlcpy (char *dest, const char *src, size_t n)
- size_t wxStrlcpy (wchar_t *dest, const wchar_t *src, size_t n)
- char * wxStrcat (char *dest, const char *src)
- wchar_t * wxStrcat (wchar_t *dest, const wchar_t *src)
- char * wxStrcat (char *dest, const wxString &src)
- char * wxStrcat (char *dest, const wxCStrData &src)
- char * wxStrcat (char *dest, const wxCharBuffer &src)
- wchar_t * wxStrcat (wchar_t *dest, const wxString &src)
- wchar_t * wxStrcat (wchar_t *dest, const wxCStrData &src)
- wchar_t * wxStrcat (wchar_t *dest, const wxWCharBuffer &src)
- char * wxStrcat (char *dest, const wchar_t *src)
- wchar_t * wxStrcat (wchar_t *dest, const char *src)
- char * wxStrncat (char *dest, const char *src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wchar_t *src, size_t n)
- char * wxStrncat (char *dest, const wxString &src, size_t n)
- char * wxStrncat (char *dest, const wxCStrData &src, size_t n)
- char * wxStrncat (char *dest, const wxCharBuffer &src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wxString &src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wxCStrData &src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wxWCharBuffer &src, size_t n)
- char * wxStrncat (char *dest, const wchar_t *src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const char *src, size_t n)
- int wxStrcmp_String (const wxString &s1, const T &s2)
- int wxStricmp_String (const wxString &s1, const T &s2)
- int wxStrcoll_String (const wxString &s1, const T &s2)
- size_t wxStrspn_String (const wxString &s1, const T &s2)
- size_t wxStrcspn_String (const wxString &s1, const T &s2)
- int wxStrncmp_String (const wxString &s1, const T &s2, size_t n)
- int wxStrnicmp_String (const wxString &s1, const T &s2, size_t n)
- size_t wxStrxfrm (char *dest, const char *src, size_t n)
- size_t wxStrxfrm (wchar_t *dest, const wchar_t *src, size_t n)
- size_t wxStrxfrm (T *dest, const wxCharTypeBuffer< T > &src, size_t n)
- size_t wxStrxfrm (char *dest, const wxString &src, size_t n)
- size_t wxStrxfrm (wchar_t *dest, const wxString &src, size_t n)
- size_t wxStrxfrm (char *dest, const wxCStrData &src, size_t n)
- size_t wxStrxfrm (wchar_t *dest, const wxCStrData &src, size_t n)
- char * wxStrtok (char *str, const char *delim, char **saveptr)
- wchar_t * wxStrtok (wchar_t *str, const wchar_t *delim, wchar_t **saveptr)
- char * wxStrtok (char *str, const wxCStrData &delim, char **saveptr)
- wchar_t * wxStrtok (wchar_t *str, const wxCStrData &delim, wchar_t **saveptr)
- char * wxStrtok (char *str, const wxString &delim, char **saveptr)
- wchar_t * wxStrtok (wchar_t *str, const wxString &delim, wchar_t **saveptr)
- const char * wxStrstr (const char *haystack, const char *needle)
- const wchar_t * wxStrstr (const wchar_t *haystack, const wchar_t *needle)
- const char * wxStrstr (const char *haystack, const wxString &needle)
- const wchar_t * wxStrstr (const wchar_t *haystack, const wxString &needle)
- const char * wxStrstr (const wxString &haystack, const wxString &needle)
- const char * wxStrstr (const wxCStrData &haystack, const wxString &needle)
- const char * wxStrstr (const wxCStrData &haystack, const wxCStrData &needle)
- const char * wxStrstr (const wxString &haystack, const char *needle)
- const char * wxStrstr (const wxCStrData &haystack, const char *needle)

- const wchar_t * wxStrstr (const wxString &haystack, const wchar_t *needle)
- const wchar_t * wxStrstr (const wxCStrData &haystack, const wchar_t *needle)
- const char * wxStrchr (const char *s, char c)
- const wchar_t * wxStrchr (const wchar_t *s, wchar_t c)
- const char * wxStrrchr (const char *s, char c)
- const wchar_t * wxStrrchr (const wchar_t *s, wchar_t c)
- const char * wxStrchr (const char *s, const wxUniChar &c)
- const wchar_t * wxStrchr (const wchar_t *s, const wxUniChar &c)
- const char * wxStrrchr (const char *s, const wxUniChar &c)
- const wchar_t * wxStrrchr (const wchar_t *s, const wxUniChar &c)
- const char * wxStrchr (const char *s, const wxUniCharRef &c)
- const wchar_t * wxStrchr (const wchar_t *s, const wxUniCharRef &c)
- const char * wxStrrchr (const char *s, const wxUniCharRef &c)
- const wchar_t * wxStrrchr (const wchar_t *s, const wxUniCharRef &c)
- const T * wxStrchr (const wxCharTypeBuffer< T > &s, T c)
- const T * wxStrrchr (const wxCharTypeBuffer< T > &s, T c)
- const T * wxStrchr (const wxCharTypeBuffer< T > &s, const wxUniChar &c)
- const T * wxStrrchr (const wxCharTypeBuffer< T > &s, const wxUniChar &c)
- const T * wxStrchr (const wxCharTypeBuffer< T > &s, const wxUniCharRef &c)
- const T * wxStrrchr (const wxCharTypeBuffer< T > &s, const wxUniCharRef &c)
- const char * wxStrchr (const wxString &s, char c)
- const char * wxStrrchr (const wxString &s, char c)
- const char * wxStrchr (const wxString &s, int c)
- const char * wxStrrchr (const wxString &s, int c)
- const char * wxStrchr (const wxString &s, const wxUniChar &c)
- const char * wxStrrchr (const wxString &s, const wxUniChar &c)
- const char * wxStrchr (const wxString &s, const wxUniCharRef &c)
- const char * wxStrrchr (const wxString &s, const wxUniCharRef &c)
- const wchar_t * wxStrchr (const wxString &s, wchar_t c)
- const wchar_t * wxStrrchr (const wxString &s, wchar_t c)
- const char * wxStrchr (const wxCStrData &s, char c)
- const char * wxStrrchr (const wxCStrData &s, char c)
- const char * wxStrchr (const wxCStrData &s, int c)
- const char * wxStrrchr (const wxCStrData &s, int c)
- const char * wxStrchr (const wxCStrData &s, const wxUniChar &c)
- const char * wxStrrchr (const wxCStrData &s, const wxUniChar &c)
- const char * wxStrchr (const wxCStrData &s, const wxUniCharRef &c)
- const char * wxStrrchr (const wxCStrData &s, const wxUniCharRef &c)
- const wchar_t * wxStrchr (const wxCStrData &s, wchar_t c)
- const wchar_t * wxStrrchr (const wxCStrData &s, wchar_t c)
- const char * wxStrpbrk (const char *s, const char *accept)
- const wchar_t * wxStrpbrk (const wchar_t *s, const wchar_t *accept)
- const char * wxStrpbrk (const char *s, const wxString &accept)
- const char * wxStrpbrk (const char *s, const wxCStrData &accept)
- const wchar_t * wxStrpbrk (const wchar_t *s, const wxString &accept)
- const wchar_t * wxStrpbrk (const wchar_t *s, const wxCStrData &accept)
- const char * wxStrpbrk (const wxString &s, const wxString &accept)
- const char * wxStrpbrk (const wxString &s, const char *accept)
- const wchar_t * wxStrpbrk (const wxString &s, const wchar_t *accept)
- const char * wxStrpbrk (const wxString &s, const wxCStrData &accept)
- const char * wxStrpbrk (const wxCStrData &s, const wxString &accept)
- const char * wxStrpbrk (const wxCStrData &s, const char *accept)
- const wchar_t * wxStrpbrk (const wxCStrData &s, const wchar_t *accept)
- const char * wxStrpbrk (const wxCStrData &s, const wxCStrData &accept)
- const T * wxStrpbrk (const S &s, const wxCharTypeBuffer< T > &accept)

- char * wxStrstr (char *haystack, const char *needle)
- wchar_t * wxStrstr (wchar_t *haystack, const wchar_t *needle)
- char * wxStrstr (char *haystack, const wxString &needle)
- wchar_t * wxStrstr (wchar_t *haystack, const wxString &needle)
- char * wxStrchr (char *s, char c)
- char * wxStrrchr (char *s, char c)
- wchar_t * wxStrchr (wchar_t *s, wchar_t c)
- wchar_t * wxStrrchr (wchar_t *s, wchar_t c)
- char * wxStrpbrk (char *s, const char *accept)
- wchar_t * wxStrpbrk (wchar_t *s, const wchar_t *accept)
- char * wxStrpbrk (char *s, const wxString &accept)
- wchar_t * wxStrpbrk (wchar_t *s, const wxString &accept)
- FILE * wxFopen (const wxString &path, const wxString &mode)
- FILE * wxFreopen (const wxString &path, const wxString &mode, FILE *stream)
- int wxRemove (const wxString &path)
- int wxRename (const wxString &oldpath, const wxString &newpath)
- char * wxFgets (char *s, int size, FILE *stream)
- int wxFgetc (FILE *stream)
- int wxUngetc (int c, FILE *stream)
- int wxAtoi (const wxString &str)
- long wxAtol (const wxString &str)
- double wxAtof (const wxString &str)
- double wxStrtod (const char *nptr, char **endptr)
- double wxStrtod (const wchar_t *nptr, wchar_t **endptr)
- double wxStrtod (const wxCharTypeBuffer< T > &nptr, T **endptr)
- double wxStrtod (const wxString &nptr, T endptr)
- double wxStrtod (const wxCStrData &nptr, T endptr)
- int wxSystem (const wxString &str)
- char * wxGetenv (const char *name)
- wchar_t * wxGetenv (const wchar_t *name)
- char * wxGetenv (const wxString &name)
- char * wxGetenv (const wxCStrData &name)
- char * wxGetenv (const wxCharBuffer &name)
- wchar_t * wxGetenv (const wxWCharBuffer &name)
- size_t wxStrftime (char *s, size_t max, size_t max, const wxString &format, const struct tm *tm)
- size_t wxStrftime (wchar_t *s, size_t max, size_t max, const wxString &format, const struct tm *tm)
- bool wxIsalnum (const wxUniChar &c)
- bool wxIsalpha (const wxUniChar &c)
- bool wxIscntrl (const wxUniChar &c)
- bool wxIsdigit (const wxUniChar &c)
- bool wxIsgraph (const wxUniChar &c)
- bool wxIslower (const wxUniChar &c)
- bool wxIsprint (const wxUniChar &c)
- bool wxIspunct (const wxUniChar &c)
- bool wxIsspace (const wxUniChar &c)
- bool wxIsupper (const wxUniChar &c)
- bool wxIsxdigit (const wxUniChar &c)
- wxUniChar wxTolower (const wxUniChar &c)
- wxUniChar wxToupper (const wxUniChar &c)
- int wxIsctrl (const wxUniChar &c)

20.65.2 Function Documentation

double wxAtof (const **wxString** & *str*)

int wxAtoi (const **wxString** & *str*)

long wxAtol (const **wxString** & *str*)

int wxFgetc (FILE * *stream*)

char* wxFgets (char * *s*, int *size*, FILE * *stream*)

FILE* wxFopen (const **wxString** & *path*, const **wxString** & *mode*)

FILE* wxFreopen (const **wxString** & *path*, const **wxString** & *mode*, FILE * *stream*)

char* wxGetenv (const char * *name*)

wchar_t* wxGetenv (const wchar_t * *name*)

char* wxGetenv (const **wxString** & *name*)

char* wxGetenv (const wxCStrData & *name*)

char* wxGetenv (const **wxCharBuffer** & *name*)

wchar_t* wxGetenv (const **wxWCharBuffer** & *name*)

bool wxIsalnum (const **wxUniChar** & *c*)

bool wxIsalpha (const **wxUniChar** & *c*)

bool wxIscntrl (const **wxUniChar** & *c*)

int wxIsctrl (const **wxUniChar** & *c*)

bool wxIsdigit (const **wxUniChar** & *c*)

bool wxIsEmpty (const char * *s*)

bool wxIsEmpty (const wchar_t * *s*)

bool wxIsEmpty (const **wxCharBuffer** & *s*)

bool wxIsEmpty (const **wxWCharBuffer** & *s*)

bool wxIsEmpty (const **wxString** & *s*)

bool wxIsEmpty (const wxCStrData & *s*)

bool wxIsgraph (const **wxUniChar** & *c*)

bool wxIslower (const **wxUniChar** & *c*)

bool wxIsprint (const **wxUniChar** & *c*)

bool wxIspunct (const wxUniChar & *c*)

bool wxIsspace (const wxUniChar & *c*)

bool wxIsupper (const wxUniChar & *c*)

bool wxIsxdigit (const wxUniChar & *c*)

int wxRemove (const wxString & *path*)

int wxRename (const wxString & *oldpath,* const wxString & *newpath*)

char* wxSetlocale (int *category,* const wxCharBuffer & *locale*)

char* wxSetlocale (int *category,* const wxString & *locale*)

char* wxSetlocale (int *category,* const wxCStrData & *locale*)

char* wxStrcat (char * *dest,* const char * *src*)

wchar_t* wxStrcat (wchar_t * *dest,* const wchar_t * *src*)

char* wxStrcat (char * *dest,* const wxString & *src*)

char* wxStrcat (char * *dest,* const wxCStrData & *src*)

char* wxStrcat (char * *dest,* const wxCharBuffer & *src*)

wchar_t* wxStrcat (wchar_t * *dest,* const wxString & *src*)

wchar_t* wxStrcat (wchar_t * *dest,* const wxCStrData & *src*)

wchar_t* wxStrcat (wchar_t * *dest,* const wxWCharBuffer & *src*)

char* wxStrcat (char * *dest,* const wchar_t * *src*)

wchar_t* wxStrcat (wchar_t * *dest,* const char * *src*)

const char* wxStrchr (const char * *s,* char *c*)

const wchar_t* wxStrchr (const wchar_t * *s,* wchar_t *c*)

const char* wxStrchr (const char * *s,* const wxUniChar & *c*)

const wchar_t* wxStrchr (const wchar_t * *s,* const wxUniChar & *c*)

const char* wxStrchr (const char * *s,* const wxUniCharRef & *c*)

const wchar_t* wxStrchr (const wchar_t * *s,* const wxUniCharRef & *c*)

const T* wxStrchr (const wxCharTypeBuffer< T > & *s,* T *c*)

const T* wxStrchr (const wxCharTypeBuffer< T > & *s,* const wxUniChar & *c*)

const T* wxStrchr (const wxCharTypeBuffer< T > & *s,* const wxUniCharRef & *c*)

const char* wxStrchr (const wxString & *s,* char *c*)

const char* wxStrchr (const wxString & *s,* int *c*)

const char* wxStrchr (const wxString & *s,* const wxUniChar & *c*)

const char* wxStrchr (const wxString & *s,* const wxUniCharRef & *c*)

const wchar_t* wxStrchr (const wxString & *s,* wchar_t *c*)

const char* wxStrchr (const wxCStrData & *s,* char *c*)

const char* wxStrchr (const wxCStrData & *s,* int *c*)

const char* wxStrchr (const wxCStrData & *s,* const wxUniChar & *c*)

const char* wxStrchr (const wxCStrData & *s,* const wxUniCharRef & *c*)

const wchar_t* wxStrchr (const wxCStrData & *s,* wchar_t *c*)

char* wxStrchr (char * *s,* char *c*)

wchar_t* wxStrchr (wchar_t * *s,* wchar_t *c*)

int wxStrcmp_String (const wxString & *s1,* const T & *s2*)

int wxStrcoll_String (const wxString & *s1,* const T & *s2*)

char* wxStrcpy (char * *dest,* const char * *src*)

wchar_t* wxStrcpy (wchar_t * *dest,* const wchar_t * *src*)

char* wxStrcpy (char * *dest,* const wxString & *src*)

char* wxStrcpy (char * *dest,* const wxCStrData & *src*)

char* wxStrcpy (char * *dest,* const wxCharBuffer & *src*)

wchar_t* wxStrcpy (wchar_t * *dest,* const wxString & *src*)

wchar_t* wxStrcpy (wchar_t * *dest,* const wxCStrData & *src*)

wchar_t* wxStrcpy (wchar_t * *dest,* const wxWCharBuffer & *src*)

char* wxStrcpy (char * *dest,* const wchar_t * *src*)

wchar_t* wxStrcpy (wchar_t * *dest,* const char * *src*)

size_t wxStrcspn_String (const wxString & *s1,* const T & *s2*)

char* wxStrdup (const wxCharBuffer & *s*)

wchar_t* wxStrdup (const wxWCharBuffer & *s*)

char* wxStrdup (const wxString & *s*)

char* wxStrdup (const wxCStrData & *s*)

size_t wxStrftime (char * *s,* size_t *max,* size_t *max,* const wxString & *format,* const struct tm * *tm*)

size_t wxStrftime (wchar_t * *s,* size_t *max,* size_t *max,* const wxString & *format,* const struct tm * *tm*)

int wxStricmp_String (const wxString & *s1,* const T & *s2*)

size_t wxStrlcpy (char * *dest,* const char * *src,* size_t *n*)

size_t wxStrlcpy (wchar_t * *dest,* const wchar_t * *src,* size_t *n*)

size_t wxStrlen (const wxCharBuffer & *s*)

size_t wxStrlen (const wxWCharBuffer & *s*)

size_t wxStrlen (const wxString & *s*)

size_t wxStrlen (const wxCStrData & *s*)

char* wxStrncat (char * *dest,* const char * *src,* size_t *n*)

wchar_t* wxStrncat (wchar_t * *dest,* const wchar_t * *src,* size_t *n*)

char* wxStrncat (char * *dest,* const wxString & *src,* size_t *n*)

char* wxStrncat (char * *dest,* const wxCStrData & *src,* size_t *n*)

char* wxStrncat (char * *dest,* const wxCharBuffer & *src,* size_t *n*)

wchar_t* wxStrncat (wchar_t * *dest,* const wxString & *src,* size_t *n*)

wchar_t* wxStrncat (wchar_t * *dest,* const wxCStrData & *src,* size_t *n*)

wchar_t* wxStrncat (wchar_t * *dest,* const wxWCharBuffer & *src,* size_t *n*)

char* wxStrncat (char * *dest,* const wchar_t * *src,* size_t *n*)

wchar_t* wxStrncat (wchar_t * *dest,* const char * *src,* size_t *n*)

int wxStrncmp_String (const wxString & *s1,* const T & *s2,* size_t *n*)

char* wxStrncpy (char * *dest,* const char * *src,* size_t *n*)

wchar_t* wxStrncpy (wchar_t * *dest,* const wchar_t * *src,* size_t *n*)

char* wxStrncpy (char * *dest,* const wxString & *src,* size_t *n*)

char* wxStrncpy (char * *dest,* const wxCStrData & *src,* size_t *n*)

char* wxStrncpy (char * *dest,* const wxCharBuffer & *src,* size_t *n*)

wchar_t* wxStrncpy (wchar_t * *dest,* const wxString & *src,* size_t *n*)

wchar_t* wxStrncpy (wchar_t * *dest,* const wxCStrData & *src,* size_t *n*)

wchar_t* wxStrncpy (wchar_t * *dest,* const **wxWCharBuffer** & *src,* size_t *n*)

char* wxStrncpy (char * *dest,* const wchar_t * *src,* size_t *n*)

wchar_t* wxStrncpy (wchar_t * *dest,* const char * *src,* size_t *n*)

int wxStrnicmp_String (const **wxString** & *s1,* const T & *s2,* size_t *n*)

size_t wxStrnlen (const char * *str,* size_t *maxlen*)

size_t wxStrnlen (const wchar_t * *str,* size_t *maxlen*)

const char* wxStrpbrk (const char * *s,* const char * *accept*)

const wchar_t* wxStrpbrk (const wchar_t * *s,* const wchar_t * *accept*)

const char* wxStrpbrk (const char * *s,* const **wxString** & *accept*)

const char* wxStrpbrk (const char * *s,* const wxCStrData & *accept*)

const wchar_t* wxStrpbrk (const wchar_t * *s,* const **wxString** & *accept*)

const wchar_t* wxStrpbrk (const wchar_t * *s,* const wxCStrData & *accept*)

const char* wxStrpbrk (const **wxString** & *s,* const **wxString** & *accept*)

const char* wxStrpbrk (const **wxString** & *s,* const char * *accept*)

const wchar_t* wxStrpbrk (const **wxString** & *s,* const wchar_t * *accept*)

const char* wxStrpbrk (const **wxString** & *s,* const wxCStrData & *accept*)

const char* wxStrpbrk (const wxCStrData & *s,* const **wxString** & *accept*)

const char* wxStrpbrk (const wxCStrData & *s,* const char * *accept*)

const wchar_t* wxStrpbrk (const wxCStrData & *s,* const wchar_t * *accept*)

const char* wxStrpbrk (const wxCStrData & *s,* const wxCStrData & *accept*)

const T* wxStrpbrk (const S & *s,* const **wxCharTypeBuffer**< T > & *accept*)

char* wxStrpbrk (char * *s,* const char * *accept*)

wchar_t* wxStrpbrk (wchar_t * *s,* const wchar_t * *accept*)

char* wxStrpbrk (char * *s,* const **wxString** & *accept*)

wchar_t* wxStrpbrk (wchar_t * *s,* const **wxString** & *accept*)

const char* wxStrrchr (const char * *s,* char *c*)

const wchar_t* wxStrrchr (const wchar_t * *s,* wchar_t *c*)

const char* wxStrrchr (const char * *s,* const **wxUniChar** & *c*)

const wchar_t* wxStrrchr (const wchar_t * *s*, const **wxUniChar** & *c*)

const char* wxStrrchr (const char * *s*, const **wxUniCharRef** & *c*)

const wchar_t* wxStrrchr (const wchar_t * *s*, const **wxUniCharRef** & *c*)

const T* wxStrrchr (const **wxCharTypeBuffer**< T > & *s*, T *c*)

const T* wxStrrchr (const **wxCharTypeBuffer**< T > & *s*, const **wxUniChar** & *c*)

const T* wxStrrchr (const **wxCharTypeBuffer**< T > & *s*, const **wxUniCharRef** & *c*)

const char* wxStrrchr (const **wxString** & *s*, char *c*)

const char* wxStrrchr (const **wxString** & *s*, int *c*)

const char* wxStrrchr (const **wxString** & *s*, const **wxUniChar** & *c*)

const char* wxStrrchr (const **wxString** & *s*, const **wxUniCharRef** & *c*)

const wchar_t* wxStrrchr (const **wxString** & *s*, wchar_t *c*)

const char* wxStrrchr (const wxCStrData & *s*, char *c*)

const char* wxStrrchr (const wxCStrData & *s*, int *c*)

const char* wxStrrchr (const wxCStrData & *s*, const **wxUniChar** & *c*)

const char* wxStrrchr (const wxCStrData & *s*, const **wxUniCharRef** & *c*)

const wchar_t* wxStrrchr (const wxCStrData & *s*, wchar_t *c*)

char* wxStrrchr (char * *s*, char *c*)

wchar_t* wxStrrchr (wchar_t * *s*, wchar_t *c*)

size_t wxStrspn_String (const **wxString** & *s1*, const T & *s2*)

const char* wxStrstr (const char * *haystack*, const char * *needle*)

const wchar_t* wxStrstr (const wchar_t * *haystack*, const wchar_t * *needle*)

const char* wxStrstr (const char * *haystack*, const **wxString** & *needle*)

const wchar_t* wxStrstr (const wchar_t * *haystack*, const **wxString** & *needle*)

const char* wxStrstr (const **wxString** & *haystack*, const **wxString** & *needle*)

const char* wxStrstr (const wxCStrData & *haystack*, const **wxString** & *needle*)

const char* wxStrstr (const wxCStrData & *haystack*, const wxCStrData & *needle*)

const char* wxStrstr (const **wxString** & *haystack*, const char * *needle*)

const char* wxStrstr (const wxCStrData & *haystack*, const char * *needle*)

const wchar_t* wxStrstr (const wxString & *haystack,* const wchar_t * *needle*)

const wchar_t* wxStrstr (const wxCStrData & *haystack,* const wchar_t * *needle*)

char* wxStrstr (char * *haystack,* const char * *needle*)

wchar_t* wxStrstr (wchar_t * *haystack,* const wchar_t * *needle*)

char* wxStrstr (char * *haystack,* const wxString & *needle*)

wchar_t* wxStrstr (wchar_t * *haystack,* const wxString & *needle*)

double wxStrtod (const char * *nptr,* char ** *endptr*)

double wxStrtod (const wchar_t * *nptr,* wchar_t ** *endptr*)

double wxStrtod (const wxCharTypeBuffer< T > & *nptr,* T ** *endptr*)

double wxStrtod (const wxString & *nptr,* T *endptr*)

double wxStrtod (const wxCStrData & *nptr,* T *endptr*)

char* wxStrtok (char * *str,* const char * *delim,* char ** *saveptr*)

wchar_t* wxStrtok (wchar_t * *str,* const wchar_t * *delim,* wchar_t ** *saveptr*)

char* wxStrtok (char * *str,* const wxCStrData & *delim,* char ** *saveptr*)

wchar_t* wxStrtok (wchar_t * *str,* const wxCStrData & *delim,* wchar_t ** *saveptr*)

char* wxStrtok (char * *str,* const wxString & *delim,* char ** *saveptr*)

wchar_t* wxStrtok (wchar_t * *str,* const wxString & *delim,* wchar_t ** *saveptr*)

size_t wxStrxfrm (char * *dest,* const char * *src,* size_t *n*)

size_t wxStrxfrm (wchar_t * *dest,* const wchar_t * *src,* size_t *n*)

size_t wxStrxfrm (T * *dest,* const wxCharTypeBuffer< T > & *src,* size_t *n*)

size_t wxStrxfrm (char * *dest,* const wxString & *src,* size_t *n*)

size_t wxStrxfrm (wchar_t * *dest,* const wxString & *src,* size_t *n*)

size_t wxStrxfrm (char * *dest,* const wxCStrData & *src,* size_t *n*)

size_t wxStrxfrm (wchar_t * *dest,* const wxCStrData & *src,* size_t *n*)

int wxSystem (const wxString & *str*)

wxChar* wxTmemchr (const wxChar * *s,* wxChar *c,* size_t *l*)

char* wxTmemchr (const char * *s,* char *c,* size_t *len*)

int wxTmemcmp (const wxChar * *sz1,* const wxChar * *sz2,* size_t *len*)

int wxTmemcmp (const char * *sz1,* const char * *sz2,* size_t *len*)

wxChar* wxTmemcpy (wxChar * *szOut,* const wxChar * *szIn,* size_t *len*)

char* wxTmemcpy (char * *szOut,* const char * *szIn,* size_t *len*)

wxChar* wxTmemmove (wxChar * *szOut,* const wxChar * *szIn,* size_t *len*)

char* wxTmemmove (char * *szOut,* const char * *szIn,* size_t *len*)

wxChar* wxTmemset (wxChar * *szOut,* const wxChar *cIn,* size_t *len*)

char* wxTmemset (char * *szOut,* const char *cIn,* size_t *len*)

wxUniChar wxTolower (const wxUniChar & *c*)

wxUniChar wxToupper (const wxUniChar & *c*)

int wxUngetc (int *c,* FILE * *stream*)

20.66 XML

20.66.1 Detailed Description

Group of classes loading and saving XML documents (http://www.w3.org/XML/).

Classes

- class wxXmlNode

  Represents a node in an XML document.
- class wxXmlAttribute

  Represents a node attribute.
- class wxXmlDocument

  This class holds XML data/document as parsed by XML parser in the root node.

20.67 XML Based Resource System (XRC)

20.67.1 Detailed Description

Resources allow your application to create controls and other user interface elements from specifications stored in an XML format.

Related Overviews: XML Based Resource System (XRC)

Classes

- class wxXmlResourceHandler

  wxSizerXmlHandler is a class for resource handlers capable of creating a wxSizer object from an XML node.
- class wxXmlResource

  This is the main class for interacting with the XML-based resource system.

20.68 wxDataViewCtrl Related Classes

20.68.1 Detailed Description

These are all classes used or provided for use with wxDataViewCtrl.

Classes

- class wxDataViewChoiceRenderer

  A wxDataViewCtrl renderer using wxChoice control and values of strings in it.
- class wxDataViewChoiceByIndexRenderer

  A wxDataViewCtrl renderer using wxChoice control and indexes into it.
- class wxDataViewModel

  wxDataViewModel is the base class for all data model to be displayed by a wxDataViewCtrl.
- class wxDataViewListModel

  Base class with abstract API for wxDataViewIndexListModel and wxDataViewVirtualListModel.
- class wxDataViewIndexListModel

  wxDataViewIndexListModel is a specialized data model which lets you address an item by its position (row) rather than its wxDataViewItem (which you can obtain from this class).
- class wxDataViewVirtualListModel

  wxDataViewVirtualListModel is a specialized data model which lets you address an item by its position (row) rather than its wxDataViewItem and as such offers the exact same interface as wxDataViewIndexListModel.
- class wxDataViewItemAttr

  This class is used to indicate to a wxDataViewCtrl that a certain item (see wxDataViewItem) has extra font attributes for its renderer.
- class wxDataViewItem

  wxDataViewItem is a small opaque class that represents an item in a wxDataViewCtrl in a persistent way, i.e.
- class wxDataViewCtrl

  wxDataViewCtrl is a control to display data either in a tree like fashion or in a tabular form or both.
- class wxDataViewModelNotifier

  A wxDataViewModelNotifier instance is owned by a wxDataViewModel and mirrors its notification interface.
- class wxDataViewRenderer

  This class is used by wxDataViewCtrl to render the individual cells.
- class wxDataViewTextRenderer

  wxDataViewTextRenderer is used for rendering text.
- class wxDataViewIconTextRenderer

  The wxDataViewIconTextRenderer class is used to display text with a small icon next to it as it is typically done in a file manager.
- class wxDataViewProgressRenderer

  This class is used by wxDataViewCtrl to render progress bars.
- class wxDataViewSpinRenderer

  This is a specialized renderer for rendering integer values.
- class wxDataViewToggleRenderer

  This class is used by wxDataViewCtrl to render toggle controls.
- class wxDataViewDateRenderer

  This class is used by wxDataViewCtrl to render calendar controls.
- class wxDataViewCustomRenderer

  You need to derive a new class from wxDataViewCustomRenderer in order to write a new renderer.
- class wxDataViewBitmapRenderer

  This class is used by wxDataViewCtrl to render bitmap controls.
- class wxDataViewColumn

This class represents a column in a wxDataViewCtrl.

- class wxDataViewListCtrl

  This class is a wxDataViewCtrl which internally uses a wxDataViewListStore and forwards most of its API to that class.

- class wxDataViewTreeCtrl

  This class is a wxDataViewCtrl which internally uses a wxDataViewTreeStore and forwards most of its API to that class.

- class wxDataViewListStore

  wxDataViewListStore is a specialised wxDataViewModel for storing a simple table of data.

- class wxDataViewTreeStore

  wxDataViewTreeStore is a specialised wxDataViewModel for storing simple trees very much like wxTreeCtrl does and it offers a similar API.

- class wxDataViewIconText

  wxDataViewIconText is used by wxDataViewIconTextRenderer for data transfer.

- class wxDataViewEvent

  This is the event class for the wxDataViewCtrl notifications.

20.69 wxPropertyGrid

20.69.1 Detailed Description

wxPropertyGrid is a specialized grid for editing properties (that is, name=value pairs).

This style of control has also been known as property sheet or object grid.

Related Overviews: wxPropertyGrid Overview

Classes

- class wxPropertyGridIterator
- class wxPGEditor

  Base class for custom wxPropertyGrid editors.
- class wxPGMultiButton

  This class can be used to have multiple buttons in a property editor.
- class wxPropertyGridPage

  Holder of property grid page information.
- class wxPropertyGridManager

  wxPropertyGridManager is an efficient multi-page version of wxPropertyGrid, which can optionally have toolbar for mode and page selection, a help text box, and a header.
- class wxPGProperty

  wxPGProperty is base class for all wxPropertyGrid properties.
- class wxPGCell

  Base class for wxPropertyGrid cell information.
- class wxPGChoices

  Helper class for managing choices of wxPropertyGrid properties.
- class wxPropertyGrid

  wxPropertyGrid is a specialized grid for editing properties - in other words name = value pairs.
- class wxPropertyGridEvent

  A property grid event holds information about events associated with wxPropertyGrid objects.
- class wxPropertyGridInterface

  Most of the shared property manipulation interface shared by wxPropertyGrid, wxPropertyGridPage, and wx↩ PropertyGridManager is defined in this class.

Chapter 21

Class Documentation

21.1 wxMessageDialog::ButtonLabel Class Reference

`#include <wx/msgdlg.h>`

21.1.1 Detailed Description

Helper class allowing to use either stock id or string labels.

This class should never be used explicitly and is not really part of wxWidgets API but rather is just an implementation helper allowing the methods such as SetYesNoLabels() and SetOKCancelLabels() below to be callable with either stock ids (e.g. wxID_CLOSE) or strings ("&Close").

Public Member Functions

- ButtonLabel (int stockId)

  Construct the label from a stock id.
- ButtonLabel (const wxString &label)

  Construct the label from the specified string.
- wxString GetAsString () const

  Return the associated label as string.
- int GetStockId () const

  Return the stock id or wxID_NONE if this is not a stock label.

21.1.2 Constructor & Destructor Documentation

wxMessageDialog::ButtonLabel::ButtonLabel (int *stockId*)

Construct the label from a stock id.

wxMessageDialog::ButtonLabel::ButtonLabel (const wxString & *label*)

Construct the label from the specified string.

21.1.3 Member Function Documentation

wxString wxMessageDialog::ButtonLabel::GetAsString ( ) const

Return the associated label as string.

Get the string label, whether it was originally specified directly or as a stock id – this is only useful for platforms without native stock items id support

int wxMessageDialog::ButtonLabel::GetStockId ( ) const

Return the stock id or wxID_NONE if this is not a stock label.

21.2 wxWindow::ChildrenRepositioningGuard Class Reference

`#include <wx/window.h>`

21.2.1 Detailed Description

Helper for ensuring EndRepositioningChildren() is called correctly.

This class wraps the calls to BeginRepositioningChildren() and EndRepositioningChildren() by performing the former in its constructor and the latter in its destructor if, and only if, the first call returned true. This is the simplest way to call these methods and if this class is created as a local variable, it also ensures that EndRepositioningChildren() is correctly called (or not) on scope exit, so its use instead of calling these methods manually is highly recommended.

Since

2.9.5

Public Member Functions

- ChildrenRepositioningGuard (wxWindow *win)

  Constructor calls wxWindow::BeginRepositioningChildren().
- ~ChildrenRepositioningGuard ()

  Destructor calls wxWindow::EndRepositioningChildren() if necessary.

21.2.2 Constructor & Destructor Documentation

wxWindow::ChildrenRepositioningGuard::ChildrenRepositioningGuard (wxWindow * *win*) [explicit]

Constructor calls wxWindow::BeginRepositioningChildren().

Parameters

win	The window to call BeginRepositioningChildren() on. If it is NULL, nothing is done.

wxWindow::ChildrenRepositioningGuard::~ChildrenRepositioningGuard ( )

Destructor calls wxWindow::EndRepositioningChildren() if necessary.

EndRepositioningChildren() is called only if a valid window was passed to the constructor and if BeginRepositioningChildren() returned true.

21.3 wxImage::HSVValue Class Reference

`#include <wx/image.h>`

21.3.1 Detailed Description

A simple class which stores hue, saturation and value as doubles in the range 0.0-1.0.

Public Member Functions

- HSVValue (double h=0.0, double s=0.0, double v=0.0)

  Constructor for HSVValue, an object that contains values for hue, saturation and value which represent the value of a color.

Public Attributes

- double hue
- double saturation
- double value

21.3.2 Constructor & Destructor Documentation

wxImage::HSVValue::HSVValue (double *h* = 0.0, double *s* = 0.0, double *v* = 0.0)

Constructor for HSVValue, an object that contains values for hue, saturation and value which represent the value of a color.

It is used by wxImage::HSVtoRGB() and wxImage::RGBtoHSV(), which convert between HSV color space and RGB color space.

21.3.3 Member Data Documentation

double wxImage::HSVValue::hue

double wxImage::HSVValue::saturation

double wxImage::HSVValue::value

21.4 wxPixelData< Image, PixelFormat >::Iterator Class Reference

`#include <wx/rawbmp.h>`

21.4.1 Detailed Description

template<class Image, class PixelFormat = wxPixelFormatFor<Image>>class wxPixelData< Image, PixelFormat >::Iterator

The iterator of class wxPixelData.

Public Member Functions

- void Reset (const PixelData &data)

  Reset the iterator to point to (0, 0).
- Iterator (PixelData &data)

  Initializes the iterator to point to the origin of the given pixel data.
- Iterator (wxBitmap &bmp, PixelData &data)

  Initializes the iterator to point to the origin of the given Bitmap.
- Iterator ()

  Default constructor.
- bool IsOk () const

  Return true if this iterator is valid.
- Iterator & operator++ ()

  Advance the iterator to the next pixel, prefix version.
- Iterator operator++ (int)

  Advance the iterator to the next pixel, postfix (hence less efficient – don't use it unless you absolutely must) version.
- void Offset (const PixelData &data, int x, int y)

  Move x pixels to the right and y down.
- void OffsetX (const PixelData &data, int x)

  Move x pixels to the right.
- void OffsetY (const PixelData &data, int y)

  Move y rows to the bottom.
- void MoveTo (const PixelData &data, int x, int y)

  Go to the given position.

- ChannelType & Red ()

  Data Access: Access to individual colour components.
- ChannelType & Green ()

  Data Access: Access to individual colour components.
- ChannelType & Blue ()

  Data Access: Access to individual colour components.
- ChannelType & Alpha ()

  Data Access: Access to individual colour components.

21.4.2 Constructor & Destructor Documentation

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::Iterator::Iterator (PixelData & *data*)

Initializes the iterator to point to the origin of the given pixel data.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::Iterator::Iterator (wxBitmap & *bmp,* PixelData & *data*)

Initializes the iterator to point to the origin of the given Bitmap.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::Iterator::Iterator ( )

Default constructor.

21.4.3 Member Function Documentation

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> ChannelType& wxPixelData< Image, PixelFormat >::Iterator::Alpha ( )

Data Access: Access to individual colour components.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> ChannelType& wxPixelData< Image, PixelFormat >::Iterator::Blue ( )

Data Access: Access to individual colour components.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> ChannelType& wxPixelData< Image, PixelFormat >::Iterator::Green ( )

Data Access: Access to individual colour components.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> bool wxPixelData< Image, PixelFormat >::Iterator::IsOk ( ) const

Return true if this iterator is valid.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> void wxPixelData< Image, PixelFormat >::Iterator::MoveTo (const PixelData & *data,* int *x,* int *y*)

Go to the given position.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> void wxPixelData< Image, PixelFormat >::Iterator::Offset (const PixelData & *data,* int *x,* int *y*)

Move *x* pixels to the right and *y* down.

Note

The rows won't wrap automatically.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> void wxPixelData< Image, PixelFormat >::Iterator::OffsetX (const PixelData & *data,* int *x*)

Move *x* pixels to the right.

Note

The rows won't wrap automatically.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> void wxPixelData< Image, PixelFormat >::Iterator::OffsetY (const PixelData & *data,* int *y*)

Move *y* rows to the bottom.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> Iterator& wxPixelData< Image, PixelFormat >::Iterator::operator++ ( )

Advance the iterator to the next pixel, prefix version.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> Iterator wxPixelData< Image, PixelFormat >::Iterator::operator++ (int)

Advance the iterator to the next pixel, postfix (hence less efficient – don't use it unless you absolutely must) version.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> ChannelType& wxPixelData< Image, PixelFormat >::Iterator::Red ( )

Data Access: Access to individual colour components.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> void wxPixelData< Image, PixelFormat >::Iterator::Reset (const PixelData & *data*)

Reset the iterator to point to (0, 0).

21.5 wxFileType::MessageParameters Class Reference

`#include <wx/mimetype.h>`

21.5.1 Detailed Description

Class representing message parameters.

An object of this class may be passed to wxFileType::GetOpenCommand() and GetPrintCommand() if more than the file name needs to be specified.

Public Member Functions

- MessageParameters ()
  Trivial default constructor.
- MessageParameters (const wxString &filename, const wxString &mimetype=wxEmptyString)
  Constructor taking a filename and a mime type.
- const wxString & GetFileName () const
  Return the filename.
- const wxString & GetMimeType () const
  Return the MIME type.
- virtual wxString GetParamValue (const wxString &name) const
  Overridable method for derived classes. Returns empty string by default.
- virtual ~MessageParameters ()
  Trivial but virtual dtor as this class can be inherited from.

21.5.2 Constructor & Destructor Documentation

wxFileType::MessageParameters::MessageParameters ( )

Trivial default constructor.

wxFileType::MessageParameters::MessageParameters (const wxString & *filename,* const wxString & *mimetype =* wxEmptyString)

Constructor taking a filename and a mime type.

virtual wxFileType::MessageParameters::~MessageParameters ( ) [virtual]

Trivial but virtual dtor as this class can be inherited from.

21.5.3 Member Function Documentation

const wxString& wxFileType::MessageParameters::GetFileName ( ) const

Return the filename.

const wxString& wxFileType::MessageParameters::GetMimeType ( ) const

Return the MIME type.

virtual wxString wxFileType::MessageParameters::GetParamValue (const wxString & *name*) const [virtual]

Overridable method for derived classes. Returns empty string by default.

21.6 wxImage::RGBValue Class Reference

`#include <wx/image.h>`

21.6.1 Detailed Description

A simple class which stores red, green and blue values as 8 bit unsigned integers in the range of 0-255.

Public Member Functions

- RGBValue (unsigned char r=0, unsigned char g=0, unsigned char b=0)
  Constructor for RGBValue, an object that contains values for red, green and blue which represent the value of a color.

Public Attributes

- unsigned char red
- unsigned char green
- unsigned char blue

21.6.2 Constructor & Destructor Documentation

wxImage::RGBValue::RGBValue (unsigned char *r* = 0, unsigned char *g* = 0, unsigned char *b* = 0)

Constructor for RGBValue, an object that contains values for red, green and blue which represent the value of a color.

It is used by wxImage::HSVtoRGB and wxImage::RGBtoHSV, which convert between HSV color space and RGB color space.

21.6.3 Member Data Documentation

unsigned char wxImage::RGBValue::blue

unsigned char wxImage::RGBValue::green

unsigned char wxImage::RGBValue::red

21.7 wxDateTime::TimeZone Class Reference

`#include <wx/datetime.h>`

21.7.1 Detailed Description

Class representing a time zone.

The representation is simply the offset, in seconds, from UTC.

Public Member Functions

- TimeZone (TZ tz)
  Constructor for a named time zone.
- TimeZone (long offset=0)
  Constructor for the given offset in seconds.
- long GetOffset () const
  Return the offset of this time zone from UTC, in seconds.

Static Public Member Functions

- static TimeZone Make (long offset)
  Create a time zone with the given offset in seconds.

21.7.2 Constructor & Destructor Documentation

wxDateTime::TimeZone::TimeZone (TZ *tz*)

Constructor for a named time zone.

wxDateTime::TimeZone::TimeZone (long *offset* = `0`)

Constructor for the given offset in seconds.

21.7.3 Member Function Documentation

long wxDateTime::TimeZone::GetOffset ( ) const

Return the offset of this time zone from UTC, in seconds.

static TimeZone wxDateTime::TimeZone::Make (long *offset*) `[static]`

Create a time zone with the given offset in seconds.

21.8 wxDateTime::Tm Struct Reference

`#include <wx/datetime.h>`

21.8.1 Detailed Description

Contains broken down date-time representation.

This struct is analogous to standard C `struct tm` and uses the same, not always immediately obvious, conventions for its members: notably its mon and mday fields count from 0 while yday counts from 1.

Public Member Functions

- bool IsValid () const

  Check if the given date/time is valid (in Gregorian calendar).
- WeekDay GetWeekDay ()

  Return the week day corresponding to this date.

Public Attributes

- wxDateTime_t msec

  Number of milliseconds.
- wxDateTime_t sec

  Seconds in 0..59 (60 with leap seconds) range.
- wxDateTime_t min

  Minutes in 0..59 range.
- wxDateTime_t hour

  Hours since midnight in 0..23 range.
- wxDateTime_t mday

  Day of the month in 1..31 range.
- wxDateTime_t yday

  Day of the year in 0..365 range.
- Month mon

  Month, as an enumerated constant.
- int year

  Year.

21.8.2 Member Function Documentation

WeekDay wxDateTime::Tm::GetWeekDay ( )

Return the week day corresponding to this date.

Unlike the other fields, the week day is not always available and so must be accessed using this method as it is computed on demand when it is called.

bool wxDateTime::Tm::IsValid ( ) const

Check if the given date/time is valid (in Gregorian calendar).

Return false if the components don't correspond to a correct date.

21.8.3 Member Data Documentation

wxDateTime_t wxDateTime::Tm::hour

Hours since midnight in 0..23 range.

wxDateTime_t wxDateTime::Tm::mday

Day of the month in 1..31 range.

wxDateTime_t wxDateTime::Tm::min

Minutes in 0..59 range.

Month wxDateTime::Tm::mon

Month, as an enumerated constant.

wxDateTime_t wxDateTime::Tm::msec

Number of milliseconds.

wxDateTime_t wxDateTime::Tm::sec

Seconds in 0..59 (60 with leap seconds) range.

wxDateTime_t wxDateTime::Tm::yday

Day of the year in 0..365 range.

int wxDateTime::Tm::year

Year.

21.9 wxAboutDialogInfo Class Reference

`#include <wx/aboutdlg.h>`

21.9.1 Detailed Description

wxAboutDialogInfo contains information shown in the standard *About* dialog displayed by the wxAboutBox() function.

This class contains the general information about the program, such as its name, version, copyright and so on, as well as lists of the program developers, documentation writers, artists and translators. The simple properties from the former group are represented as a string with the exception of the program icon and the program web site, while the lists from the latter group are stored as wxArrayString and can be either set entirely at once using wx↩AboutDialogInfo::SetDevelopers and similar functions or built one by one using wxAboutDialogInfo::AddDeveloper etc.

Please also notice that while all the main platforms have the native implementation of the about dialog, they are often more limited than the generic version provided by wxWidgets and so the generic version is used if wxAboutDialogInfo has any fields not supported by the native version. Currently GTK+ version supports all the possible fields natively but MSW and Mac versions don't support URLs, licence text nor custom icons in the about dialog and if either of those is used, wxAboutBox() will automatically use the generic version so you should avoid specifying these fields to achieve more native look and feel.

Example of usage:

```
void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxAboutDialogInfo aboutInfo;
    aboutInfo.SetName("MyApp");
    aboutInfo.SetVersion(MY_APP_VERSION_STRING);
    aboutInfo.SetDescription(_("My wxWidgets-based application!"));
    aboutInfo.SetCopyright("(C) 1992-2010");
    aboutInfo.SetWebSite("http://myapp.org");
    aboutInfo.AddDeveloper("My Self");

    wxAboutBox(aboutInfo);
}
```

Library: wxAdvanced

Category: Common Dialogs, Data Structures

See also

wxAboutDialogInfo::SetArtists

Public Member Functions

- wxAboutDialogInfo ()

  Default constructor leaves all fields are initially uninitialized, in general you should call at least SetVersion(), SetCopyright() and SetDescription().
- void AddArtist (const wxString &artist)

  Adds an artist name to be shown in the program credits.
- void AddDeveloper (const wxString &developer)

  Adds a developer name to be shown in the program credits.
- void AddDocWriter (const wxString &docwriter)

  Adds a documentation writer name to be shown in the program credits.
- void AddTranslator (const wxString &translator)

  Adds a translator name to be shown in the program credits.
- wxString GetName () const

  Get the name of the program.
- bool HasDescription () const

  Returns true if a description string has been specified.
- const wxString & GetDescription ()

  Get the description string.
- bool HasCopyright () const

  Returns true if a copyright string has been specified.
- const wxString & GetCopyright () const

  Get the copyright string.
- void SetArtists (const wxArrayString &artists)

  Sets the list of artists to be shown in the program credits.
- void SetCopyright (const wxString &copyright)

Set the short string containing the program copyright information.

- void SetDescription (const wxString &desc)

  Set brief, but possibly multiline, description of the program.
- void SetDevelopers (const wxArrayString &developers)

  Set the list of developers of the program.
- void SetDocWriters (const wxArrayString &docwriters)

  Set the list of documentation writers.
- void SetIcon (const wxIcon &icon)

  Set the icon to be shown in the dialog.
- void SetLicence (const wxString &licence)

  Set the long, multiline string containing the text of the program licence.
- void SetLicense (const wxString &licence)

  This is the same as SetLicence().
- void SetName (const wxString &name)

  Set the name of the program.
- void SetTranslators (const wxArrayString &translators)

  Set the list of translators.
- void SetVersion (const wxString &version, const wxString &longVersion=wxString())

  Set the version of the program.
- void SetWebSite (const wxString &url, const wxString &desc=wxEmptyString)

  Set the web site for the program and its description (which defaults to url itself if empty).

21.9.2 Constructor & Destructor Documentation

wxAboutDialogInfo::wxAboutDialogInfo ( )

Default constructor leaves all fields are initially uninitialized, in general you should call at least SetVersion(), SetCopyright() and SetDescription().

21.9.3 Member Function Documentation

void wxAboutDialogInfo::AddArtist (const wxString & *artist*)

Adds an artist name to be shown in the program credits.

See also

SetArtists()

void wxAboutDialogInfo::AddDeveloper (const wxString & *developer*)

Adds a developer name to be shown in the program credits.

See also

SetDevelopers()

void wxAboutDialogInfo::AddDocWriter (const wxString & *docwriter*)

Adds a documentation writer name to be shown in the program credits.

See also

SetDocWriters()

void wxAboutDialogInfo::AddTranslator (const wxString & *translator*)

Adds a translator name to be shown in the program credits.

Notice that if no translator names are specified explicitly, wxAboutBox() will try to use the translation of the string `translator-credits` from the currently used message catalog – this can be used to show just the name of the translator of the program in the current language.

See also

SetTranslators()

const wxString& wxAboutDialogInfo::GetCopyright ( ) const

Get the copyright string.

Returns

The copyright string

const wxString& wxAboutDialogInfo::GetDescription ( )

Get the description string.

Returns

The description string, free-form.

wxString wxAboutDialogInfo::GetName ( ) const

Get the name of the program.

Returns

Name of the program

See also

SetName()

bool wxAboutDialogInfo::HasCopyright ( ) const

Returns true if a copyright string has been specified.

See also

GetCopyright()

bool wxAboutDialogInfo::HasDescription ( ) const

Returns true if a description string has been specified.

See also

GetDescription()

void wxAboutDialogInfo::SetArtists (const wxArrayString & *artists*)

Sets the list of artists to be shown in the program credits.

See also

AddArtist()

void wxAboutDialogInfo::SetCopyright (const wxString & *copyright*)

Set the short string containing the program copyright information.

Notice that any occurrences of `"(C)"` in *copyright* will be replaced by the copyright symbol (circled C) automatically, which means that you can avoid using this symbol in the program source code which can be problematic,

void wxAboutDialogInfo::SetDescription (const wxString & *desc*)

Set brief, but possibly multiline, description of the program.

void wxAboutDialogInfo::SetDevelopers (const wxArrayString & *developers*)

Set the list of developers of the program.

See also

AddDeveloper()

void wxAboutDialogInfo::SetDocWriters (const wxArrayString & *docwriters*)

Set the list of documentation writers.

See also

AddDocWriter()

void wxAboutDialogInfo::SetIcon (const wxIcon & *icon*)

Set the icon to be shown in the dialog.

By default the icon of the main frame will be shown if the native about dialog supports custom icons. If it doesn't but a valid icon is specified using this method, the generic about dialog is used instead so you should avoid calling this function for maximally native look and feel.

void wxAboutDialogInfo::SetLicence (const wxString & *licence*)

Set the long, multiline string containing the text of the program licence.

Only GTK+ version supports showing the licence text in the native about dialog currently so the generic version will be used under all the other platforms if this method is called. To preserve the native look and feel it is advised that you do not call this method but provide a separate menu item in the `"Help"` menu for displaying the text of your program licence.

void wxAboutDialogInfo::SetLicense (const wxString & *licence*)

This is the same as SetLicence().

void wxAboutDialogInfo::SetName (const wxString & *name*)

Set the name of the program.

If this method is not called, the string returned by wxApp::GetAppName will be shown in the dialog.

void wxAboutDialogInfo::SetTranslators (const wxArrayString & *translators*)

Set the list of translators.

Please see AddTranslator() for additional discussion.

void wxAboutDialogInfo::SetVersion (const wxString & *version,* const wxString & *longVersion* = wxString ())

Set the version of the program.

The word "version" shouldn't be included in *version*. Example *version* values: "1.2" and "RC2". In about dialogs with more space set aside for version information, *longVersion* is used. Example *longVersion* values: "Version 1.2" and "Release Candidate 2". If *version* is non-empty but *longVersion* is empty, a long version is constructed automatically, using *version* (by simply prepending "Version " to *version*).

The generic about dialog and native GTK+ dialog use *version* only, as a suffix to the program name. The native MSW and OS X about dialogs use the long version.

void wxAboutDialogInfo::SetWebSite (const wxString & *url,* const wxString & *desc* = wxEmptyString)

Set the web site for the program and its description (which defaults to *url* itself if empty).

Please notice that only GTK+ version currently supports showing the link in the native about dialog so if this method is called, the generic version will be used under all the other platforms.

21.10 wxAcceleratorEntry Class Reference

`#include <wx/accel.h>`

21.10.1 Detailed Description

An object used by an application wishing to create an accelerator table (see wxAcceleratorTable).

Library: wxCore

Category: Data Structures

See also

wxAcceleratorTable, wxWindow::SetAcceleratorTable

Public Member Functions

- wxAcceleratorEntry (int flags=0, int keyCode=0, int cmd=0, wxMenuItem *item=NULL)
  Constructor.
- wxAcceleratorEntry (const wxAcceleratorEntry &entry)
  Copy ctor.

- int GetCommand () const

  Returns the command identifier for the accelerator table entry.
- int GetFlags () const

  Returns the flags for the accelerator table entry.
- int GetKeyCode () const

  Returns the keycode for the accelerator table entry.
- wxMenuItem * GetMenuItem () const

  Returns the menu item associated with this accelerator entry.
- void Set (int flags, int keyCode, int cmd, wxMenuItem *item=NULL)

  Sets the accelerator entry parameters.
- bool IsOk () const

  Returns true if this object is correctly initialized.
- wxString ToString () const

  Returns a textual representation of this accelerator.
- wxString ToRawString () const

  Returns a textual representation of this accelerator which is appropriate for saving in configuration files.
- bool FromString (const wxString &str)

  Parses the given string and sets the accelerator accordingly.
- wxAcceleratorEntry & operator= (const wxAcceleratorEntry &entry)
- bool operator== (const wxAcceleratorEntry &entry) const
- bool operator!= (const wxAcceleratorEntry &entry) const

21.10.2 Constructor & Destructor Documentation

wxAcceleratorEntry::wxAcceleratorEntry (int *flags* = `0`, int *keyCode* = `0`, int *cmd* = `0`, wxMenuItem * *item* = `NULL`)

Constructor.

Parameters

flags	A combination of the wxAcceleratorEntryFlags values, which indicates which modifier keys are held down.
keyCode	The keycode to be detected. See wxKeyCode for a full list of keycodes.
cmd	The menu or control command identifier (ID).
item	The menu item associated with this accelerator.

wxAcceleratorEntry::wxAcceleratorEntry (const wxAcceleratorEntry & *entry*)

Copy ctor.

21.10.3 Member Function Documentation

bool wxAcceleratorEntry::FromString (const wxString & *str*)

Parses the given string and sets the accelerator accordingly.

Parameters

str	This string may be either in the same format as returned by ToString(), i.e. contain the accelerator itself only, or have the format of a full menu item text with i.e. `Label TAB Accelerator`. In the latter case, the part of the string before the TAB is ignored. Notice that the latter format is only supported for the compatibility with the previous wxWidgets versions and the new code should pass only the accelerator string itself to this function.

Returns

true if the given string correctly initialized this object (i.e. if IsOk() returns true after this call)

int wxAcceleratorEntry::GetCommand ( ) const

Returns the command identifier for the accelerator table entry.

int wxAcceleratorEntry::GetFlags ( ) const

Returns the flags for the accelerator table entry.

int wxAcceleratorEntry::GetKeyCode ( ) const

Returns the keycode for the accelerator table entry.

wxMenuItem* wxAcceleratorEntry::GetMenuItem ( ) const

Returns the menu item associated with this accelerator entry.

bool wxAcceleratorEntry::IsOk ( ) const

Returns true if this object is correctly initialized.

bool wxAcceleratorEntry::operator!= (const wxAcceleratorEntry & *entry*) const

wxAcceleratorEntry& wxAcceleratorEntry::operator= (const wxAcceleratorEntry & *entry*)

bool wxAcceleratorEntry::operator== (const wxAcceleratorEntry & *entry*) const

void wxAcceleratorEntry::Set (int *flags*, int *keyCode*, int *cmd*, wxMenuItem * *item* = NULL)

Sets the accelerator entry parameters.

Parameters

flags	A combination of the wxAcceleratorEntryFlags values, which indicates which modifier keys are held down.
keyCode	The keycode to be detected. See wxKeyCode for a full list of keycodes.
cmd	The menu or control command identifier (ID).
item	The menu item associated with this accelerator.

wxString wxAcceleratorEntry::ToRawString ( ) const

Returns a textual representation of this accelerator which is appropriate for saving in configuration files.

Unlike the string returned by ToString(), this one is never translated so, while it's not suitable for showing to the user, it can be used to uniquely identify the accelerator independently of the user language.

The returned string can still be parsed by FromString().

Since

2.9.4

wxString wxAcceleratorEntry::ToString ( ) const

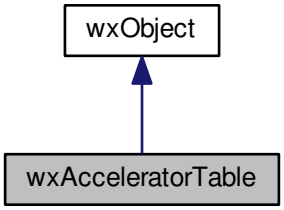
Returns a textual representation of this accelerator.

The returned string is of the form `[Alt+][Ctrl+][RawCtrl+][Shift+]Key` where the modifier keys are present only if the corresponding flag is set.

21.11 wxAcceleratorTable Class Reference

`#include <wx/accel.h>`

Inheritance diagram for wxAcceleratorTable:



21.11.1 Detailed Description

An accelerator table allows the application to specify a table of keyboard shortcuts for menu or button commands.

The object wxNullAcceleratorTable is defined to be a table with no data, and is the initial accelerator table for a window.

Example:

```
wxAcceleratorEntry entries[4];
entries[0].Set(wxACCEL_CTRL, (int) 'N', ID_NEW_WINDOW);
entries[1].Set(wxACCEL_CTRL, (int) 'X', wxID_EXIT);
entries[2].Set(wxACCEL_SHIFT, (int) 'A', ID_ABOUT);
entries[3].Set(wxACCEL_NORMAL, WXK_DELETE, wxID_CUT);

wxAcceleratorTable accel(4, entries);
frame->SetAcceleratorTable(accel);
```

Remarks

An accelerator takes precedence over normal processing and can be a convenient way to program some event handling. For example, you can use an accelerator table to enable a dialog with a multi-line text control to accept CTRL-Enter as meaning 'OK'.

Library: wxCore

Category: Data Structures

Predefined objects/pointers: wxNullAcceleratorTable

See also

wxAcceleratorEntry, wxWindow::SetAcceleratorTable

Public Member Functions

- wxAcceleratorTable ()

  Default ctor.
- wxAcceleratorTable (int n, const wxAcceleratorEntry entries[])

  Initializes the accelerator table from an array of wxAcceleratorEntry.
- wxAcceleratorTable (const wxString &resource)

  Loads the accelerator table from a Windows resource (Windows only).
- virtual ~wxAcceleratorTable ()

  Destroys the wxAcceleratorTable object.
- bool IsOk () const

  Returns true if the accelerator table is valid.

Additional Inherited Members

21.11.2 Constructor & Destructor Documentation

wxAcceleratorTable::wxAcceleratorTable ( )

Default ctor.

wxAcceleratorTable::wxAcceleratorTable (int *n*, const wxAcceleratorEntry *entries[]*)

Initializes the accelerator table from an array of wxAcceleratorEntry.

Parameters

n	Number of accelerator entries.
entries	The array of entries.

wxPerl Note: The wxPerl constructor accepts a list of either Wx::AcceleratorEntry objects or references to 3-element arrays [flags, keyCode, cmd] , like the parameters of Wx::AcceleratorEntry::new.

wxAcceleratorTable::wxAcceleratorTable (const wxString & *resource*)

Loads the accelerator table from a Windows resource (Windows only).

Availability: only available for the wxMSW port.

Parameters

resource	Name of a Windows accelerator.

virtual wxAcceleratorTable::~wxAcceleratorTable ( ) [virtual]

Destroys the wxAcceleratorTable object.

See Object Destruction for more info.

21.11.3 Member Function Documentation

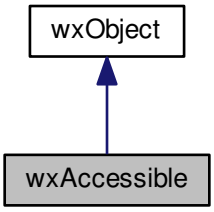
bool wxAcceleratorTable::IsOk ( ) const

Returns true if the accelerator table is valid.

21.12 wxAccessible Class Reference

`#include <wx/access.h>`

Inheritance diagram for wxAccessible:



21.12.1 Detailed Description

The wxAccessible class allows wxWidgets applications, and wxWidgets itself, to return extended information about user interface elements to client applications such as screen readers.

This is the main way in which wxWidgets implements accessibility features.

At present, only Microsoft Active Accessibility is supported by this class.

To use this class, derive from wxAccessible, implement appropriate functions, and associate an object of the class with a window using wxWindow::SetAccessible.

All functions return an indication of success, failure, or not implemented using values of the wxAccStatus enum type.

If you return `wxACC_NOT_IMPLEMENTED` from any function, the system will try to implement the appropriate functionality. However this will not work with all functions.

Most functions work with an object *id*, which can be zero to refer to 'this' UI element, or greater than zero to refer to the nth child element. This allows you to specify elements that don't have a corresponding wxWindow or wxAccessible; for example, the sash of a splitter window.

For details on the semantics of functions and types, please refer to the Microsoft Active Accessibility 1.2 documentation.

This class is compiled into wxWidgets only if the wxUSE_ACCESSIBILITY setup symbol is set to 1.

Availability: only available for the wxMSW port.

Library: wxCore

Category: Miscellaneous

See also

Accessibility Sample

Public Member Functions

- wxAccessible (wxWindow *win=NULL)

  Constructor, taking an optional window.
- ~wxAccessible ()

  Destructor.
- virtual wxAccStatus DoDefaultAction (int childId)

  Performs the default action for the object.
- virtual wxAccStatus GetChild (int childId, wxAccessible **child)

  Gets the specified child (starting from 1).
- virtual wxAccStatus GetChildCount (int *childCount)

  Returns the number of children in childCount.
- virtual wxAccStatus GetDefaultAction (int childId, wxString *actionName)

  Gets the default action for this object (0) or a child (greater than 0).
- virtual wxAccStatus GetDescription (int childId, wxString *description)

  Returns the description for this object or a child.
- virtual wxAccStatus GetFocus (int *childId, wxAccessible **child)

  Gets the window with the keyboard focus.
- virtual wxAccStatus GetHelpText (int childId, wxString *helpText)

  Returns help text for this object or a child, similar to tooltip text.
- virtual wxAccStatus GetKeyboardShortcut (int childId, wxString *shortcut)

  Returns the keyboard shortcut for this object or child.
- virtual wxAccStatus GetLocation (wxRect &rect, int elementId)

  Returns the rectangle for this object (id is 0) or a child element (id is greater than 0).
- virtual wxAccStatus GetName (int childId, wxString *name)

  Gets the name of the specified object.
- virtual wxAccStatus GetParent (wxAccessible **parent)

  Returns the parent of this object, or NULL.
- virtual wxAccStatus GetRole (int childId, wxAccRole *role)

  Returns a role constant describing this object.
- virtual wxAccStatus GetSelections (wxVariant *selections)

  Gets a variant representing the selected children of this object.
- virtual wxAccStatus GetState (int childId, long *state)

  Returns a state constant.
- virtual wxAccStatus GetValue (int childId, wxString *strValue)

  Returns a localized string representing the value for the object or child.
- wxWindow * GetWindow ()

  Returns the window associated with this object.
- virtual wxAccStatus HitTest (const wxPoint &pt, int *childId, wxAccessible **childObject)

  Returns a status value and object id to indicate whether the given point was on this or a child object.
- virtual wxAccStatus Navigate (wxNavDir navDir, int fromId, int *toId, wxAccessible **toObject)

  Navigates from fromId to toId or to toObject.
- virtual wxAccStatus Select (int childId, wxAccSelectionFlags selectFlags)

  Selects the object or child.
- void SetWindow (wxWindow *window)

  Sets the window associated with this object.

Static Public Member Functions

- static void NotifyEvent (int eventType, wxWindow *window, wxAccObject objectType, int objectType)
  Allows the application to send an event when something changes in an accessible object.

Additional Inherited Members

21.12.2 Constructor & Destructor Documentation

wxAccessible::wxAccessible (wxWindow * *win* = NULL)

Constructor, taking an optional window.

The object can be associated with a window later.

wxAccessible::~wxAccessible ( )

Destructor.

21.12.3 Member Function Documentation

virtual wxAccStatus wxAccessible::DoDefaultAction (int *childId*) [virtual]

Performs the default action for the object.

childId is 0 (the action for this object) or greater than 0 (the action for a child).

Returns

wxACC_NOT_SUPPORTED if there is no default action for this window (e.g. an edit control).

virtual wxAccStatus wxAccessible::GetChild (int *childId*, wxAccessible ** *child*) [virtual]

Gets the specified child (starting from 1).

If *child* is NULL and the return value is wxACC_OK, this means that the child is a simple element and not an accessible object.

virtual wxAccStatus wxAccessible::GetChildCount (int * *childCount*) [virtual]

Returns the number of children in *childCount*.

virtual wxAccStatus wxAccessible::GetDefaultAction (int *childId*, wxString * *actionName*) [virtual]

Gets the default action for this object (0) or a child (greater than 0).

Return wxACC_OK even if there is no action. *actionName* is the action, or the empty string if there is no action. The retrieved string describes the action that is performed on an object, not what the object does as a result. For example, a toolbar button that prints a document has a default action of "Press" rather than "Prints the current document."

virtual wxAccStatus wxAccessible::GetDescription (int *childId*, wxString * *description*) [virtual]

Returns the description for this object or a child.

virtual wxAccStatus wxAccessible::GetFocus (int * *childId,* wxAccessible ** *child*) [virtual]

Gets the window with the keyboard focus.

If childId is 0 and child is NULL, no object in this subhierarchy has the focus. If this object has the focus, child should be 'this'.

virtual wxAccStatus wxAccessible::GetHelpText (int *childId,* wxString * *helpText*) [virtual]

Returns help text for this object or a child, similar to tooltip text.

virtual wxAccStatus wxAccessible::GetKeyboardShortcut (int *childId,* wxString * *shortcut*) [virtual]

Returns the keyboard shortcut for this object or child.

Returns e.g. ALT+K.

virtual wxAccStatus wxAccessible::GetLocation (wxRect & *rect,* int *elementId*) [virtual]

Returns the rectangle for this object (id is 0) or a child element (id is greater than 0).

rect is in screen coordinates.

virtual wxAccStatus wxAccessible::GetName (int *childId,* wxString * *name*) [virtual]

Gets the name of the specified object.

virtual wxAccStatus wxAccessible::GetParent (wxAccessible ** *parent*) [virtual]

Returns the parent of this object, or NULL.

virtual wxAccStatus wxAccessible::GetRole (int *childId,* wxAccRole * *role*) [virtual]

Returns a role constant describing this object.

See wxAccRole for a list of these roles.

virtual wxAccStatus wxAccessible::GetSelections (wxVariant * *selections*) [virtual]

Gets a variant representing the selected children of this object.

Acceptable values are:

- a null variant (IsNull() returns true)
- a list variant (GetType() == "list")
- an integer representing the selected child element, or 0 if this object is selected (GetType() == "long")
- a "void*" pointer to a wxAccessible child object

virtual wxAccStatus wxAccessible::GetState (int *childId,* long * *state*) [virtual]

Returns a state constant.

See wxAccStatus for a list of these states.

virtual wxAccStatus wxAccessible::GetValue (int *childId,* wxString * *strValue*) [virtual]

Returns a localized string representing the value for the object or child.

wxWindow* wxAccessible::GetWindow ( )

Returns the window associated with this object.

virtual wxAccStatus wxAccessible::HitTest (const wxPoint & *pt,* int * *childId,* wxAccessible ** *childObject*) [virtual]

Returns a status value and object id to indicate whether the given point was on this or a child object.

Can return either a child object, or an integer representing the child element, starting from 1.

pt is in screen coordinates.

virtual wxAccStatus wxAccessible::Navigate (wxNavDir *navDir,* int *fromId,* int * *toId,* wxAccessible ** *toObject*) [virtual]

Navigates from *fromId* to *toId* or to *toObject*.

static void wxAccessible::NotifyEvent (int *eventType,* wxWindow * *window,* wxAccObject *objectType,* int *objectType*) [static]

Allows the application to send an event when something changes in an accessible object.

virtual wxAccStatus wxAccessible::Select (int *childId,* wxAccSelectionFlags *selectFlags*) [virtual]

Selects the object or child.

See wxAccSelectionFlags for a list of the selection actions.

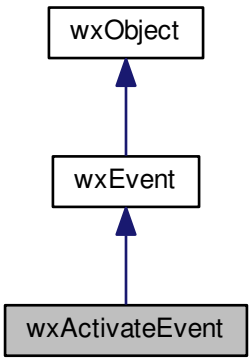
void wxAccessible::SetWindow (wxWindow * *window*)

Sets the window associated with this object.

21.13 wxActivateEvent Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxActivateEvent:



21.13.1 Detailed Description

An activate event is sent when a window or application is being activated or deactivated.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxActivateEvent& event)

Event macros:

- EVT_ACTIVATE(func): Process a `wxEVT_ACTIVATE` event.
- EVT_ACTIVATE_APP(func): Process a `wxEVT_ACTIVATE_APP` event. This event is received by the wx↩ App-derived instance only.
- EVT_HIBERNATE(func): Process a hibernate event, supplying the member function. This event applies to wxApp only, and only on Windows SmartPhone and PocketPC. It is generated when the system is low on memory; the application should free up as much memory as possible, and restore full working state when it receives a `wxEVT_ACTIVATE` or `wxEVT_ACTIVATE_APP` event.

Library: wxCore

Category: Events

See also

Events and Event Handling, wxApp::IsActive

Public Types

- enum Reason {
  Reason_Mouse,
  Reason_Unknown }

Specifies the reason for the generation of this event.

Public Member Functions

- wxActivateEvent (wxEventType eventType=wxEVT_NULL, bool active=true, int id=0, Reason Activation↩ Reason=Reason_Unknown)

  Constructor.
- bool GetActive () const

  Returns true if the application or window is being activated, false otherwise.
- Reason GetActivationReason () const

  Allows to check if the window was activated by clicking it with the mouse or in some other way.

Additional Inherited Members

21.13.2 Member Enumeration Documentation

enum wxActivateEvent::Reason

Specifies the reason for the generation of this event.

See GetActivationReason().

Since

3.0

Enumerator

Reason_Mouse Window activated by mouse click.

Reason_Unknown Window was activated with some other method than mouse click.

21.13.3 Constructor & Destructor Documentation

wxActivateEvent::wxActivateEvent (wxEventType *eventType* = wxEVT_NULL, bool *active* = `true`, int *id* = `0`, Reason *ActivationReason* = Reason_Unknown)

Constructor.

21.13.4 Member Function Documentation

Reason wxActivateEvent::GetActivationReason ( ) const

Allows to check if the window was activated by clicking it with the mouse or in some other way.

This method is currently only implemented in wxMSW and returns `Reason_Mouse` there if the window was activated by a mouse click and `Reason_Unknown` if it was activated in any other way (e.g. from keyboard or programmatically).

Under all the other platforms, `Reason_Unknown` is always returned.

Since

3.0

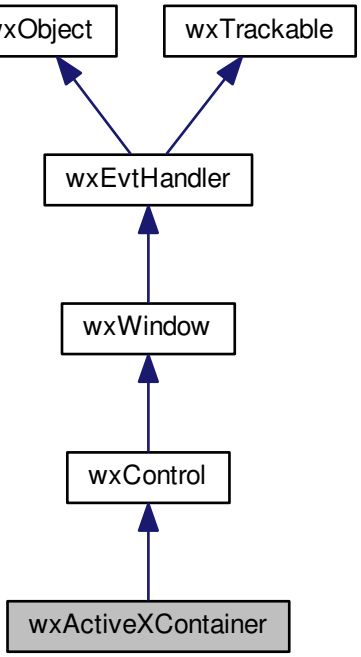
bool wxActivateEvent::GetActive ( ) const

Returns true if the application or window is being activated, false otherwise.

21.14 wxActiveXContainer Class Reference

`#include <wx/msw/ole/activex.h>`

Inheritance diagram for wxActiveXContainer:



21.14.1 Detailed Description

wxActiveXContainer is a host for an ActiveX control on Windows (and as such is a platform-specific class).

Note that the HWND that the class contains is the actual HWND of the ActiveX control so using dynamic events and connecting to `wxEVT_SIZE`, for example, will receive the actual size message sent to the control.

It is somewhat similar to the ATL class CAxWindow in operation.

The size of the ActiveX control's content is generally guaranteed to be that of the client size of the parent of this wxActiveXContainer.

You can also process ActiveX events through wxActiveXEvent.

21.14.2 Example

This is an example of how to use the Adobe Acrobat Reader ActiveX control to read PDF files (requires Acrobat Reader 4 and up). Controls like this are typically found and dumped from OLEVIEW.exe that is distributed with

Microsoft Visual C++. This example also demonstrates how to create a backend for wxMediaCtrl.

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//
// wxPDFMediaBackend
//
// http://partners.adobe.com/public/developer/en/acrobat/sdk/pdf/iac/IACOverview.pdf
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#include "wx/mediactrl.h"       // wxMediaBackendCommonBase
#include "wx/msw/ole/activex.h" // wxActiveXContainer
#include "wx/msw/ole/automtn.h" // wxAutomationObject

const IID DIID__DPdf = {0xCA8A9781,0x280D,0x11CF,{0xA2,0x4D,0x44,0x45,0x53,0x54,0x00,0x00}};
const IID DIID__DPdfEvents = {0xCA8A9782,0x280D,0x11CF,{0xA2,0x4D,0x44,0x45,0x53,0x54,0x00,0x00}};
const CLSID CLSID_Pdf = {0xCA8A9780,0x280D,0x11CF,{0xA2,0x4D,0x44,0x45,0x53,0x54,0x00,0x00}};

class WXDLLIMPEXP_MEDIA wxPDFMediaBackend : public wxMediaBackendCommonBase
{
public:
    wxPDFMediaBackend() : m_pAX(NULL) {}
    virtual ~wxPDFMediaBackend()
    {
        if(m_pAX)
        {
            m_pAX->DissociateHandle();
            delete m_pAX;
        }
    }
    virtual bool CreateControl(wxControl* ctrl, wxWindow* parent,
                               wxWindowID id,
                               const wxPoint& pos,
                               const wxSize& size,
                               long style,
                               const wxValidator& validator,
                               const wxString& name)
    {
        IDispatch* pDispatch;
        if( ::CoCreateInstance(CLSID_Pdf, NULL,
                           CLSCTX_INPROC_SERVER,
                           DIID__DPdf, (void**)&pDispatch) != 0 )
            return false;

        m_PDF.SetDispatchPtr(pDispatch); // wxAutomationObject will release itself

        if ( !ctrl->wxControl::Create(parent, id, pos, size,
                           (style & ~wxBORDER_MASK) |
      wxBORDER_NONE,
                           validator, name) )
            return false;

        m_ctrl = wxStaticCast(ctrl, wxMediaCtrl);
        m_pAX = new wxActiveXContainer(ctrl,
                    DIID__DPdf,
                    pDispatch);

        wxPDFMediaBackend::ShowPlayerControls(wxMEDIACTRLPLAYERCONTROLS_NONE)
      ;
        return true;
    }

    virtual bool Play()
    {
        return true;
    }
    virtual bool Pause()
    {
        return true;
    }
    virtual bool Stop()
    {
        return true;
    }

    virtual bool Load(const wxString& fileName)
    {
        if(m_PDF.CallMethod("LoadFile", fileName).GetBool())
        {
            m_PDF.CallMethod("setCurrentPage", wxVariant((long)0));
            NotifyMovieLoaded(); // initial refresh
            wxSizeEvent event;
            m_pAX->OnSize(event);
            return true;
        }

        return false;
    }
```

```
    virtual bool Load(const wxURI& location)
    {
        return m_PDF.CallMethod("LoadFile", location.BuildUnescapedURI()).GetBool();
    }
    virtual bool Load(const wxURI& WXUNUSED(location),
                      const wxURI& WXUNUSED(proxy))
    {
        return false;
    }

    virtual wxMediaState GetState()
    {
        return wxMEDIASTATE_STOPPED;
    }

    virtual bool SetPosition(wxLongLong where)
    {
        m_PDF.CallMethod("setCurrentPage", wxVariant((long)where.
      GetValue()));
        return true;
    }
    virtual wxLongLong GetPosition()
    {
        return 0;
    }
    virtual wxLongLong GetDuration()
    {
        return 0;
    }

    virtual void Move(int WXUNUSED(x), int WXUNUSED(y),
                      int WXUNUSED(w), int WXUNUSED(h))
    {
    }
    wxSize GetVideoSize() const
    {
        return wxDefaultSize;
    }

    virtual double GetPlaybackRate()
    {
        return 0;
    }
    virtual bool SetPlaybackRate(double)
    {
        return false;
    }

    virtual double GetVolume()
    {
        return 0;
    }
    virtual bool SetVolume(double)
    {
        return false;
    }

    virtual bool ShowPlayerControls(wxMediaCtrlPlayerControls flags)
    {
        if(flags)
        {
            m_PDF.CallMethod("setShowToolbar", true);
            m_PDF.CallMethod("setShowScrollbars", true);
        }
        else
        {
            m_PDF.CallMethod("setShowToolbar", false);
            m_PDF.CallMethod("setShowScrollbars", false);
        }

        return true;
    }

    wxActiveXContainer* m_pAX;
    wxAutomationObject m_PDF;

    wxDECLARE_DYNAMIC_CLASS(wxPDFMediaBackend)
};

wxIMPLEMENT_DYNAMIC_CLASS(wxPDFMediaBackend, wxMediaBackend);

// Put this in one of your existing source files and then create a wxMediaCtrl with
wxMediaCtrl* mymediactrl = new wxMediaCtrl(this, "myfile.pdf",
      wxID_ANY,
                                         wxDefaultPosition,
      wxSize(300,300),
                                         0, "wxPDFMediaBackend");
```

```
// [this] is the parent window, "myfile.pdf" is the PDF file to open
```

Availability: only available for the wxMSW port.

Library: wxCore

Category: Controls, Interprocess Communication

See also

wxActiveXEvent, Flash Sample

Public Member Functions

- wxActiveXContainer (wxWindow *parent, REFIID iid, IUnknown *pUnk)

  Creates this ActiveX container.
- virtual bool QueryClientSiteInterface (REFIID iid, void **_interface, const char *&desc)

  Queries host's site for interface.

Additional Inherited Members

21.14.3 Constructor & Destructor Documentation

wxActiveXContainer::wxActiveXContainer (wxWindow * *parent,* REFIID *iid,* IUnknown * *pUnk*)

Creates this ActiveX container.

Parameters

parent	parent of this control. Must not be NULL.
iid	COM IID of pUnk to query. Must be a valid interface to an ActiveX control.
pUnk	Interface of ActiveX control.

21.14.4 Member Function Documentation

virtual bool wxActiveXContainer::QueryClientSiteInterface (REFIID *iid,* void ** *_interface,* const char *& *desc*)
`[virtual]`

Queries host's site for interface.

Parameters

iid	The iid of the required interface.
_interface	Double pointer to outgoing interface. Supply your own interface if desired.
desc	The description of the outgoing interface.

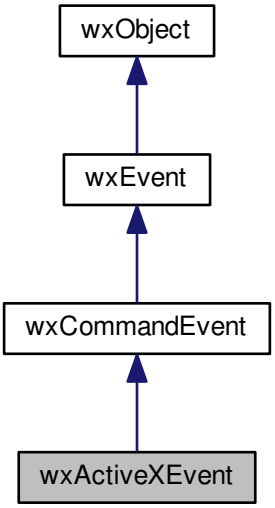
Returns

bool Return true if interface supplied else return false.

21.15 wxActiveXEvent Class Reference

```
#include <wx/msw/ole/activex.h>
```

Inheritance diagram for wxActiveXEvent:



21.15.1 Detailed Description

An event class for handling ActiveX events passed from wxActiveXContainer.

ActiveX events are basically a function call with the parameters passed through an array of wxVariants along with a return value that is a wxVariant itself. What type the parameters or return value are depends on the context (i.e. what the .idl specifies).

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxActiveXEvent& event)

Event macros:

- EVT_ACTIVEX(func): Sent when the ActiveX control hosted by wxActiveXContainer receives an ActiveX event.

ActiveX event parameters can get extremely complex and may be beyond the abilities of wxVariant. If 'operator[]' fails, prints an error messages or crashes the application, event handlers should use GetNativeParameters() instead to obtain the original event information. Calls to operator[] and GetNativeParmeters() can be mixed. It is valid to handle some parameters of an event with operator[] and others directly through GetNativeParameters(). It is **not** valid however to manipulate the same parameter using both approaches at the same time.

Availability: only available for the wxMSW port.

Library: wxCore

Category: Events

Public Member Functions

- DISPID GetDispatchId (int idx) const

  Returns the dispatch id of this ActiveX event.
- size_t ParamCount () const

  Obtains the number of parameters passed through the ActiveX event.
- wxString ParamName (size_t idx) const

  Obtains the param name of the param number idx specifies as a string.
- wxString ParamType (size_t idx) const

  Obtains the param type of the param number idx specifies as a string.
- wxVariant operator[] (size_t idx)

  Obtains the actual parameter value specified by idx.
- wxActiveXEventNativeMSW * GetNativeParameters () const

  Obtain the original MSW parameters for the event.

Additional Inherited Members

21.15.2 Member Function Documentation

DISPID wxActiveXEvent::GetDispatchId (int *idx*) const

Returns the dispatch id of this ActiveX event.

This is the numeric value from the .idl file specified by the id().

wxActiveXEventNativeMSW* wxActiveXEvent::GetNativeParameters ( ) const

Obtain the original MSW parameters for the event.

Event handlers can use this information to handle complex event parameters that are beyond the scope of wx↩ Variant. The information returned here is the information passed to the original 'Invoke' method call.

Returns

a pointer to a struct containing the original MSW event parameters

wxVariant wxActiveXEvent::operator[] (size_t *idx*)

Obtains the actual parameter value specified by idx.

size_t wxActiveXEvent::ParamCount ( ) const

Obtains the number of parameters passed through the ActiveX event.

wxString wxActiveXEvent::ParamName (size_t *idx*) const

Obtains the param name of the param number idx specifies as a string.

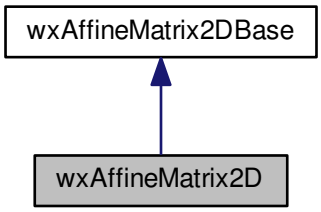
wxString wxActiveXEvent::ParamType (size_t *idx*) const

Obtains the param type of the param number idx specifies as a string.

21.16 wxAffineMatrix2D Class Reference

`#include <wx/affinematrix2d.h>`

Inheritance diagram for wxAffineMatrix2D:



21.16.1 Detailed Description

A 3x2 matrix representing an affine 2D transformation.

Library: wxCore

Category: Miscellaneous

Since

2.9.2

Public Member Functions

- wxAffineMatrix2D ()

  Default constructor.
- void Get (wxMatrix2D *mat2D, wxPoint2DDouble *tr) const

  Get the component values of the matrix.
- void Set (const wxMatrix2D &mat2D, const wxPoint2DDouble &tr)

  Set all elements of this matrix.
- void Concat (const wxAffineMatrix2DBase &t)

  Concatenate this matrix with another one.
- bool Invert ()

  Invert this matrix.
- bool IsIdentity () const

  Check if this is the identity matrix.
- bool operator!= (const wxAffineMatrix2DBase &t) const

  Check that this matrix differs from t.
- void Translate (wxDouble dx, wxDouble dy)

  Add the translation to this matrix.
- void Scale (wxDouble xScale, wxDouble yScale)

Add scaling to this matrix.

- void Mirror (int direction=wxHORIZONTAL)

  Add mirroring to this matrix.

- void Rotate (wxDouble cRadians)

  Add clockwise rotation to this matrix.

- wxPoint2DDouble TransformPoint (const wxPoint2DDouble &p) const

  Applies this matrix to the point.

- void TransformPoint (wxDouble *x, wxDouble *y) const
- wxPoint2DDouble TransformDistance (const wxPoint2DDouble &p) const

  Applies the linear part of this matrix, i.e. without translation.

- void TransformDistance (wxDouble *dx, wxDouble *dy) const

- void IsEqual (const wxAffineMatrix2DBase &t)

  Check that this matrix is identical with t.

- bool operator== (const wxAffineMatrix2DBase &t) const

  Check that this matrix is identical with t.

21.16.2 Constructor & Destructor Documentation

wxAffineMatrix2D::wxAffineMatrix2D ( )

Default constructor.

The matrix elements are initialize to the identity matrix.

21.16.3 Member Function Documentation

void wxAffineMatrix2D::Concat (const wxAffineMatrix2DBase & *t*) [virtual]

Concatenate this matrix with another one.

The parameter matrix is the multiplicand.

Parameters

t	The multiplicand.

```
//           | t.m_11  t.m_12  0 |   | m_11  m_12   0 |
// matrix' = | t.m_21  t.m_22  0 | x | m_21  m_22   0 |
//           | t.m_tx  t.m_ty  1 |   | m_tx  m_ty   1 |
```

Implements wxAffineMatrix2DBase.

void wxAffineMatrix2D::Get (wxMatrix2D * *mat2D*, wxPoint2DDouble * *tr*) const [virtual]

Get the component values of the matrix.

Parameters

mat2D	The rotational components of the matrix (upper 2 x 2), must be non-NULL.
tr	The translational components of the matrix, may be NULL.

Implements wxAffineMatrix2DBase.

bool wxAffineMatrix2D::Invert ( ) [virtual]

Invert this matrix.

If the matrix is not invertible, i.e. if its determinant is 0, returns false and doesn't modify it.

```
//           | m_11  m_12  0 |
// Invert    | m_21  m_22  0 |
//           | m_tx  m_ty  1 |
```

Implements wxAffineMatrix2DBase.

void wxAffineMatrix2D::IsEqual (const wxAffineMatrix2DBase & *t*)

Check that this matrix is identical with *t*.

Parameters

t	The matrix compared with this.

bool wxAffineMatrix2D::IsIdentity ( ) const [virtual]

Check if this is the identity matrix.

Implements wxAffineMatrix2DBase.

void wxAffineMatrix2D::Mirror (int *direction* = wxHORIZONTAL)

Add mirroring to this matrix.

Parameters

direction	The direction(s) used for mirroring. One of wxHORIZONTAL, wxVERTICAL or their combination wxBOTH.

bool wxAffineMatrix2D::operator!= (const wxAffineMatrix2DBase & *t*) const

Check that this matrix differs from *t*.

Parameters

t	The matrix compared with this.

bool wxAffineMatrix2D::operator== (const wxAffineMatrix2DBase & *t*) const

Check that this matrix is identical with *t*.

Parameters

t	The matrix compared with this.

void wxAffineMatrix2D::Rotate (wxDouble *cRadians*) [virtual]

Add clockwise rotation to this matrix.

Parameters

cRadians	Rotation angle in radians, clockwise.

```
//           | cos    sin   0 |   | m_11  m_12   0 |
// matrix' = | -sin   cos   0 | x | m_21  m_22   0 |
//           |  0      0    1 |   | m_tx  m_ty   1 |
```

Implements wxAffineMatrix2DBase.

void wxAffineMatrix2D::Scale (wxDouble *xScale,* wxDouble *yScale*) `[virtual]`

Add scaling to this matrix.

Parameters

xScale	Scaling in x direction.
yScale	Scaling in y direction.

```
//           | xScale   0      0 |   | m_11  m_12   0 |
// matrix' = |   0    yScale   0 | x | m_21  m_22   0 |
//           |   0      0      1 |   | m_tx  m_ty   1 |
```

Implements wxAffineMatrix2DBase.

void wxAffineMatrix2D::Set (const wxMatrix2D & *mat2D,* const wxPoint2DDouble & *tr*) `[virtual]`

Set all elements of this matrix.

Parameters

mat2D	The rotational components of the matrix (upper 2 x 2).
tr	The translational components of the matrix.

Implements wxAffineMatrix2DBase.

wxPoint2DDouble wxAffineMatrix2D::TransformDistance (const wxPoint2DDouble & *p*) const

Applies the linear part of this matrix, i.e. without translation.

Parameters

p	The source receiving the transformations.

Returns

The source with the transformations applied.

```
//                                | m_11  m_12   0 |
// dist' = | src.m_x  src._my  0 | x | m_21  m_22   0 |
//                                | m_tx  m_ty   1 |
```

void wxAffineMatrix2D::TransformDistance (wxDouble * *dx,* wxDouble * *dy*) const

wxPoint2DDouble wxAffineMatrix2D::TransformPoint (const wxPoint2DDouble & *p*) const

Applies this matrix to the point.

Parameters

p	The point receiving the transformations.

Returns

The point with the transformations applied.

```
//                                  | m_11  m_12   0 |
// point' = | src.m_x  src._my  1 | x | m_21  m_22   0 |
//                                  | m_tx  m_ty   1 |
```

void wxAffineMatrix2D::TransformPoint (wxDouble * *x*, wxDouble * *y*) const

void wxAffineMatrix2D::Translate (wxDouble *dx*, wxDouble *dy*) `[virtual]`

Add the translation to this matrix.

Parameters

dx	The translation in x direction.
dy	The translation in y direction.

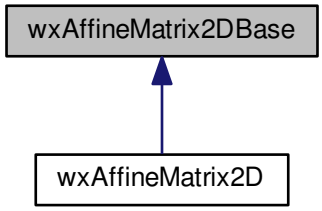
```
//           |  1   0   0 |   | m_11  m_12   0 |
// matrix' = |  0   1   0 | x | m_21  m_22   0 |
//           | dx  dy   1 |   | m_tx  m_ty   1 |
```

Implements wxAffineMatrix2DBase.

21.17 wxAffineMatrix2DBase Class Reference

`#include <wx/affinematrix2dbase.h>`

Inheritance diagram for wxAffineMatrix2DBase:



21.17.1 Detailed Description

A 2x3 matrix representing an affine 2D transformation.

This is an abstract base class implemented by wxAffineMatrix2D only so far, but in the future we also plan to derive wxGraphicsMatrix from it.

Library: wxCore

Category: Miscellaneous

Since

2.9.2

Public Member Functions

- wxAffineMatrix2DBase ()

  Default constructor.
- virtual ~wxAffineMatrix2DBase ()
- virtual void Set (const wxMatrix2D &mat2D, const wxPoint2DDouble &tr)=0

  Set all elements of this matrix.
- virtual void Get (wxMatrix2D *mat2D, wxPoint2DDouble *tr) const =0

  Get the component values of the matrix.
- virtual void Concat (const wxAffineMatrix2DBase &t)=0

  Concatenate this matrix with another one.
- virtual bool Invert ()=0

  Invert this matrix.
- virtual bool IsIdentity () const =0

  Check if this is the identity matrix.
- bool operator!= (const wxAffineMatrix2DBase &t) const

  Check that this matrix differs from t.
- virtual void Translate (wxDouble dx, wxDouble dy)=0

  Add the translation to this matrix.
- virtual void Scale (wxDouble xScale, wxDouble yScale)=0

  Add scaling to this matrix.
- virtual void Rotate (wxDouble cRadians)=0

  Add clockwise rotation to this matrix.
- void Mirror (int direction=wxHORIZONTAL)

  Add mirroring to this matrix.
- wxPoint2DDouble TransformPoint (const wxPoint2DDouble &p) const

  Applies this matrix to the point.
- void TransformPoint (wxDouble *x, wxDouble *y) const
- wxPoint2DDouble TransformDistance (const wxPoint2DDouble &p) const

  Applies the linear part of this matrix, i.e. without translation.
- void TransformDistance (wxDouble *dx, wxDouble *dy) const

- virtual bool IsEqual (const wxAffineMatrix2DBase &t) const =0

  Check that this matrix is identical with t.
- bool operator== (const wxAffineMatrix2DBase &t) const

  Check that this matrix is identical with t.

21.17.2 Constructor & Destructor Documentation

wxAffineMatrix2DBase::wxAffineMatrix2DBase ( )

Default constructor.

The matrix elements are initialize to the identity matrix.

virtual wxAffineMatrix2DBase::~wxAffineMatrix2DBase ( ) [virtual]

21.17.3 Member Function Documentation

virtual void wxAffineMatrix2DBase::Concat (const wxAffineMatrix2DBase & *t*) [pure virtual]

Concatenate this matrix with another one.

The parameter matrix is the multiplicand.

Parameters

t	The multiplicand.

```
//           | t.m_11  t.m_12  0 |    | m_11  m_12   0 |
// matrix' = | t.m_21  t.m_22  0 | x  | m_21  m_22   0 |
//           | t.m_tx  t.m_ty  1 |    | m_tx  m_ty   1 |
```

Implemented in wxAffineMatrix2D.

virtual void wxAffineMatrix2DBase::Get (wxMatrix2D * *mat2D*, wxPoint2DDouble * *tr*) const [pure virtual]

Get the component values of the matrix.

Parameters

mat2D	The rotational components of the matrix (upper 2 x 2), must be non-NULL.
tr	The translational components of the matrix, may be NULL.

Implemented in wxAffineMatrix2D.

virtual bool wxAffineMatrix2DBase::Invert ( ) [pure virtual]

Invert this matrix.

If the matrix is not invertible, i.e. if its determinant is 0, returns false and doesn't modify it.

```
//           | m_11  m_12  0 |
// Invert    | m_21  m_22  0 |
//           | m_tx  m_ty  1 |
```

Implemented in wxAffineMatrix2D.

virtual bool wxAffineMatrix2DBase::IsEqual (const wxAffineMatrix2DBase & *t*) const [pure virtual]

Check that this matrix is identical with *t*.

Parameters

t	The matrix compared with this.

virtual bool wxAffineMatrix2DBase::IsIdentity ( ) const [pure virtual]

Check if this is the identity matrix.

Implemented in wxAffineMatrix2D.

void wxAffineMatrix2DBase::Mirror (int *direction* = wxHORIZONTAL)

Add mirroring to this matrix.

Parameters

direction	The direction(s) used for mirroring. One of wxHORIZONTAL, wxVERTICAL or their combination wxBOTH.

bool wxAffineMatrix2DBase::operator!= (const wxAffineMatrix2DBase & *t*) const

Check that this matrix differs from *t*.

Parameters

t	The matrix compared with this.

bool wxAffineMatrix2DBase::operator== (const wxAffineMatrix2DBase & *t*) const

Check that this matrix is identical with *t*.

Parameters

t	The matrix compared with this.

virtual void wxAffineMatrix2DBase::Rotate (wxDouble *cRadians*) [pure virtual]

Add clockwise rotation to this matrix.

Parameters

cRadians	Rotation angle in radians, clockwise.

Implemented in wxAffineMatrix2D.

virtual void wxAffineMatrix2DBase::Scale (wxDouble *xScale,* wxDouble *yScale*) [pure virtual]

Add scaling to this matrix.

Parameters

xScale	Scaling in x direction.
yScale	Scaling in y direction.

Implemented in wxAffineMatrix2D.

virtual void wxAffineMatrix2DBase::Set (const wxMatrix2D & *mat2D,* const wxPoint2DDouble & *tr*) [pure virtual]

Set all elements of this matrix.

Parameters

mat2D	The rotational components of the matrix (upper 2 x 2).
tr	The translational components of the matrix.

Implemented in wxAffineMatrix2D.

wxPoint2DDouble wxAffineMatrix2DBase::TransformDistance (const wxPoint2DDouble & *p*) const

Applies the linear part of this matrix, i.e. without translation.

Parameters

p	The source receiving the transformations.

Returns

The source with the transformations applied.

void wxAffineMatrix2DBase::TransformDistance (wxDouble * *dx*, wxDouble * *dy*) const

wxPoint2DDouble wxAffineMatrix2DBase::TransformPoint (const wxPoint2DDouble & *p*) const

Applies this matrix to the point.

Parameters

p	The point receiving the transformations.

Returns

The point with the transformations applied.

void wxAffineMatrix2DBase::TransformPoint (wxDouble * *x*, wxDouble * *y*) const

virtual void wxAffineMatrix2DBase::Translate (wxDouble *dx*, wxDouble *dy*) `[pure virtual]`

Add the translation to this matrix.

Parameters

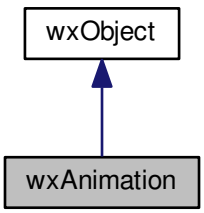
dx	The translation in x direction.
dy	The translation in y direction.

Implemented in wxAffineMatrix2D.

21.18 wxAnimation Class Reference

`#include <wx/animate.h>`

Inheritance diagram for wxAnimation:



21.18.1 Detailed Description

This class encapsulates the concept of a platform-dependent animation.

An animation is a sequence of frames of the same size. Sound is not supported by wxAnimation.

Note that on wxGTK wxAnimation is capable of loading the formats supported by the internally-used `gdk-pixbuf` library (typically this means only `wxANIMATION_TYPE_GIF`). On other platforms wxAnimation is always capable of loading both GIF and ANI formats (i.e. both `wxANIMATION_TYPE_GIF` and `wxANIMATION_TYPE_ANI`).

Library: wxAdvanced

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullAnimation

See also

wxAnimationCtrl, Animation Sample

Public Member Functions

- wxAnimation ()
  Default ctor.
- wxAnimation (const wxAnimation &anim)
  Copy ctor.
- wxAnimation (const wxString &name, wxAnimationType type=wxANIMATION_TYPE_ANY)
  Loads an animation from a file.
- virtual ~wxAnimation ()
  Destructor.
- virtual int GetDelay (unsigned int i) const
  Returns the delay for the i-th frame in milliseconds.
- virtual wxImage GetFrame (unsigned int i) const
  Returns the i-th frame as a wxImage.
- virtual unsigned int GetFrameCount () const
  Returns the number of frames for this animation.
- virtual wxSize GetSize () const
  Returns the size of the animation.
- virtual bool IsOk () const
  Returns true if animation data is present.
- virtual bool Load (wxInputStream &stream, wxAnimationType type=wxANIMATION_TYPE_ANY)
  Loads an animation from the given stream.
- virtual bool LoadFile (const wxString &name, wxAnimationType type=wxANIMATION_TYPE_ANY)
  Loads an animation from a file.
- wxAnimation & operator= (const wxAnimation &brush)
  Assignment operator, using reference counting.

Additional Inherited Members

21.18.2 Constructor & Destructor Documentation

wxAnimation::wxAnimation ( )

Default ctor.

wxAnimation::wxAnimation (const wxAnimation & *anim*)

Copy ctor.

wxAnimation::wxAnimation (const wxString & *name*, wxAnimationType *type* = wxANIMATION_TYPE_ANY)

Loads an animation from a file.

Parameters

name	The name of the file to load.
type	See LoadFile() for more info.

virtual wxAnimation::~wxAnimation ( ) [virtual]

Destructor.

See Object Destruction for more info.

21.18.3 Member Function Documentation

virtual int wxAnimation::GetDelay (unsigned int *i*) const [virtual]

Returns the delay for the i-th frame in milliseconds.

If `-1` is returned the frame is to be displayed forever.

virtual wxImage wxAnimation::GetFrame (unsigned int *i*) const [virtual]

Returns the i-th frame as a wxImage.

This method is not implemented in the native wxGTK implementation of this class and always returns an invalid image there.

virtual unsigned int wxAnimation::GetFrameCount ( ) const [virtual]

Returns the number of frames for this animation.

This method is not implemented in the native wxGTK implementation of this class and always returns 0 there.

virtual wxSize wxAnimation::GetSize ( ) const [virtual]

Returns the size of the animation.

virtual bool wxAnimation::IsOk ( ) const [virtual]

Returns true if animation data is present.

virtual bool wxAnimation::Load (wxInputStream & *stream*, wxAnimationType *type* = wxANIMATION_TYPE_ANY) [virtual]

Loads an animation from the given stream.

Parameters

stream	The stream to use to load the animation. Under wxGTK may be any kind of stream; under other platforms this must be a seekable stream.
type	One of the wxAnimationType enumeration values.

Returns

true if the operation succeeded, false otherwise.

virtual bool wxAnimation::LoadFile (const wxString & *name,* wxAnimationType *type* = wxANIMATION_TYPE_ANY) `[virtual]`

Loads an animation from a file.

Parameters

name	A filename.
type	One of the wxAnimationType values; wxANIMATION_TYPE_ANY means that the function should try to autodetect the filetype.

Returns

true if the operation succeeded, false otherwise.

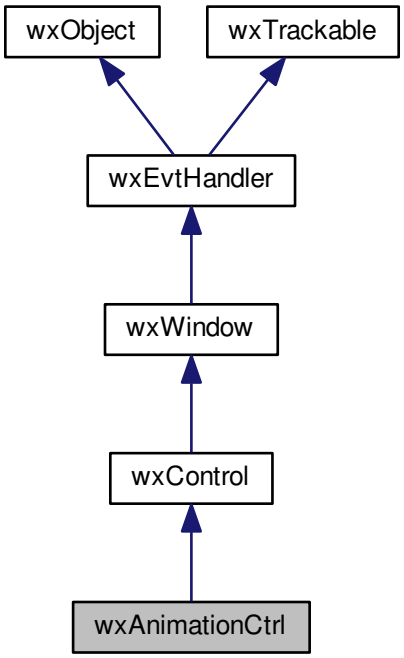
wxAnimation& wxAnimation::operator= (const wxAnimation & *brush*)

Assignment operator, using reference counting.

21.19 wxAnimationCtrl Class Reference

`#include <wx/animate.h>`

Inheritance diagram for wxAnimationCtrl:



21.19.1 Detailed Description

This is a static control which displays an animation.

wxAnimationCtrl API is as simple as possible and won't give you full control on the animation; if you need it then use wxMediaCtrl.

This control is useful to display a (small) animation while doing a long task (e.g. a "throbber").

It is only available if `wxUSE_ANIMATIONCTRL` is set to 1 (the default).

Styles

This class supports the following styles:

- wxAC_DEFAULT_STYLE: The default style: wxBORDER_NONE.
- wxAC_NO_AUTORESIZE: By default, the control will adjust its size to exactly fit to the size of the animation when SetAnimation is called. If this style flag is given, the control will not change its size

Library: wxAdvanced

Category: Controls

Implementations: native under wxGTK, wxMSW ports; a generic implementation is used elsewhere.

See also

wxAnimation, Animation Sample

Public Member Functions

- wxAnimationCtrl (wxWindow *parent, wxWindowID id, const wxAnimation &anim=wxNullAnimation, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxAC_DEFAULT_STYLE, const wxString &name=wxAnimationCtrlNameStr)

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxAnimation &anim=wxNullAnimation, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxAC_DEFAULT_STYLE, const wxString &name=wxAnimationCtrlNameStr)

  Creates the control with the given anim animation.
- virtual wxAnimation GetAnimation () const

  Returns the animation associated with this control.
- wxBitmap GetInactiveBitmap () const

  Returns the inactive bitmap shown in this control when the; see SetInactiveBitmap() for more info.
- virtual bool IsPlaying () const

  Returns true if the animation is being played.
- virtual bool LoadFile (const wxString &file, wxAnimationType animType=wxANIMATION_TYPE_ANY)

  Loads the animation from the given file and calls SetAnimation().
- virtual bool Load (wxInputStream &file, wxAnimationType animType=wxANIMATION_TYPE_ANY)

  Loads the animation from the given stream and calls SetAnimation().
- virtual bool Play ()

  Starts playing the animation.
- virtual void SetAnimation (const wxAnimation &anim)

  Sets the animation to play in this control.
- virtual void SetInactiveBitmap (const wxBitmap &bmp)

  Sets the bitmap to show on the control when it's not playing an animation.
- virtual void Stop ()

  Stops playing the animation.

Additional Inherited Members

21.19.2 Constructor & Destructor Documentation

wxAnimationCtrl::wxAnimationCtrl (wxWindow * *parent*, wxWindowID *id*, const wxAnimation & *anim* = wxNullAnimation, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxAC_DEFAULT_STYLE, const wxString & *name* = `wxAnimationCtrlNameStr`)

Initializes the object and calls Create() with all the parameters.

21.19.3 Member Function Documentation

bool wxAnimationCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxAnimation & *anim* = wxNullAnimation, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxAC_DEFAULT_STYLE, const wxString & *name* = `wxAnimationCtrlNameStr`)

Creates the control with the given *anim* animation.

After control creation you must explicitly call Play() to start to play the animation. Until that function won't be called, the first frame of the animation is displayed.

Parameters

parent	Parent window, must be non-NULL.
id	The identifier for the control.
anim	The initial animation shown in the control.
pos	Initial position.
size	Initial size.
style	The window style, see wxAC_∗ flags.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

virtual wxAnimation wxAnimationCtrl::GetAnimation ( ) const [virtual]

Returns the animation associated with this control.

wxBitmap wxAnimationCtrl::GetInactiveBitmap ( ) const

Returns the inactive bitmap shown in this control when the; see SetInactiveBitmap() for more info.

virtual bool wxAnimationCtrl::IsPlaying ( ) const [virtual]

Returns true if the animation is being played.

virtual bool wxAnimationCtrl::Load (wxInputStream & *file*, wxAnimationType *animType* = wxANIMATION_TYPE_ANY) [virtual]

Loads the animation from the given stream and calls SetAnimation().

See wxAnimation::Load() for more info.

virtual bool wxAnimationCtrl::LoadFile (const wxString & *file*, wxAnimationType *animType* = wxANIMATION_TYPE_ANY) [virtual]

Loads the animation from the given file and calls SetAnimation().

See wxAnimation::LoadFile for more info.

virtual bool wxAnimationCtrl::Play ( ) [virtual]

Starts playing the animation.

The animation is always played in loop mode (unless the last frame of the animation has an infinite delay time) and always start from the first frame even if you stopped it while some other frame was displayed.

virtual void wxAnimationCtrl::SetAnimation (const wxAnimation & *anim*) [virtual]

Sets the animation to play in this control.

If the previous animation is being played, it's Stop() stopped. Until Play() isn't called, a static image, the first frame of the given animation or the background colour will be shown (see SetInactiveBitmap() for more info).

virtual void wxAnimationCtrl::SetInactiveBitmap (const wxBitmap & *bmp*) `[virtual]`

Sets the bitmap to show on the control when it's not playing an animation.

If you set as inactive bitmap wxNullBitmap (which is the default), then the first frame of the animation is instead shown when the control is inactive; in this case, if there's no valid animation associated with the control (see SetAnimation()), then the background colour of the window is shown.

If the control is not playing the animation, the given bitmap will be immediately shown, otherwise it will be shown as soon as Stop() is called.

Note that the inactive bitmap, if smaller than the control's size, will be centered in the control; if bigger, it will be stretched to fit it.

virtual void wxAnimationCtrl::Stop ( ) `[virtual]`

Stops playing the animation.

The control will show the first frame of the animation, a custom static image or the window's background colour as specified by the last SetInactiveBitmap() call.

21.20 wxAny Class Reference

`#include <wx/any.h>`

21.20.1 Detailed Description

The wxAny class represents a container for any type.

Its value can be changed at run time, possibly to a different type of value.

wxAny is a backwards-incompatible (but convertible) successor class for wxVariant, essentially doing the same thing in a more modern, template- based manner and with transparent support for any user data type.

Some pseudo-code'ish example of use with arbitrary user data:

```
void SomeFunction()
{
    MyClass myObject;
    wxAny any = myObject;

    // Do something
    // ...

    // Let's do a sanity check to make sure that any still holds
    // data of correct type.
    if ( any.CheckType<MyClass>() )
    {
        // Thank goodness, still a correct type.
        MyClass myObject2 = any.As<MyClass>();
    }
    else
    {
        // Something has gone horribly wrong!
        wxFAIL();
    }
}
```

When compared to wxVariant, there are various internal implementation differences as well. For instance, wxAny only allocates separate data object in heap for large objects (i.e. ones with size more than WX_ANY_VALUE_BUFFER_SIZE, which at the time of writing is 16 bytes).

Note

When performing conversions between strings and floating point numbers, the representation of numbers in C locale is always used. I.e.

```
wxAny("1.23").GetAs<double>()
```

will always work, even if the current locale uses comma as decimal separator.

Library: wxBase

Category: Data Structures

See also

wxAnyValueType, wxVariant, Caveats When Not Using C++ RTTI

Public Member Functions

- wxAny ()

  Default constructor.
- template<typename T >
  wxAny (const T &value)

  Constructs wxAny from data.
- wxAny (const wxAny &any)

  Constructs wxAny from another wxAny.
- wxAny (const wxVariant &variant)

  Constructs wxAny, converting value from wxVariant.
- ~wxAny ()

  Destructor.
- template<typename T >
  T As () const

  This template function converts wxAny into given type.
- template<typename T >
  bool CheckType () const

  Use this template function for checking if this wxAny holds a specific C++ data type.
- template<typename T >
  bool GetAs (T *value) const

  Template function that retrieves and converts the value of this wxAny to the type that T value is.*
- bool GetAs (wxVariant *value) const

  Specialization of GetAs() that allows conversion of wxAny into wxVariant.
- const wxAnyValueType * GetType () const

  Returns the value type as wxAnyValueType instance.
- bool HasSameType (const wxAny &other) const

  Returns true if this and another wxAny have the same value type.
- bool IsNull () const

  Tests if wxAny is null (that is, whether there is no data).
- void MakeNull ()

  Makes wxAny null (that is, clears it).

Assignment operators

- template<typename T >
  wxAny & operator= (const T &value)

- wxAny & operator= (const wxAny &any)
- wxAny & operator= (const wxVariant &variant)

Equality operators

- bool operator== (signed char value) const
- bool operator== (signed short value) const
- bool operator== (signed int value) const
- bool operator== (signed long value) const
- bool operator== (wxLongLong_t value) const
- bool operator== (unsigned char value) const
- bool operator== (unsigned short value) const
- bool operator== (unsigned int value) const
- bool operator== (unsigned long value) const
- bool operator== (wxULongLong_t value) const
- bool operator== (float value) const
- bool operator== (double value) const
- bool operator== (bool value) const
- bool operator== (const char *value) const
- bool operator== (const wchar_t *value) const
- bool operator== (const wxString &value) const

Inequality operators

- bool operator!= (signed char value) const
- bool operator!= (signed short value) const
- bool operator!= (signed int value) const
- bool operator!= (signed long value) const
- bool operator!= (wxLongLong_t value) const
- bool operator!= (unsigned char value) const
- bool operator!= (unsigned short value) const
- bool operator!= (unsigned int value) const
- bool operator!= (unsigned long value) const
- bool operator!= (wxULongLong_t value) const
- bool operator!= (float value) const
- bool operator!= (double value) const
- bool operator!= (bool value) const
- bool operator!= (const char *value) const
- bool operator!= (const wchar_t *value) const
- bool operator!= (const wxString &value) const

21.20.2 Constructor & Destructor Documentation

wxAny::wxAny ( )

Default constructor.

It seeds the object with a null value.

template<typename T > wxAny::wxAny (const T & *value*)

Constructs wxAny from data.

wxAny::wxAny (const wxAny & *any*)

Constructs wxAny from another wxAny.

wxAny::wxAny (const wxVariant & *variant*)

Constructs wxAny, converting value from wxVariant.

Remarks

Because of this conversion, it is not usually possible to have wxAny that actually holds a wxVariant. If wxVariant cannot be converted to a specific data type, wxAny will then hold and manage reference to wxVariantData* similar to how wxVariant does.

wxAny::~wxAny ( )

Destructor.

21.20.3 Member Function Documentation

template<typename T > T wxAny::As ( ) const

This template function converts wxAny into given type.

In most cases no type conversion is performed, so if the type is incorrect an assertion failure will occur.

Remarks

For convenience, conversion is done when T is wxString. This is useful when a string literal (which are treated as const char* and const wchar_t*) has been assigned to wxAny.

template<typename T > bool wxAny::CheckType ( ) const

Use this template function for checking if this wxAny holds a specific C++ data type.

See also

wxAnyValueType::CheckType()

template<typename T > bool wxAny::GetAs (T * *value*) const

Template function that retrieves and converts the value of this wxAny to the type that T* value is.

Returns

Returns true if conversion was successful.

bool wxAny::GetAs (wxVariant * *value*) const

Specialization of GetAs() that allows conversion of wxAny into wxVariant.

Returns

Returns true if conversion was successful. Conversion usually only fails if variant used custom wxVariantData that did not implement the wxAny to wxVariant conversion functions.

const wxAnyValueType* wxAny::GetType ( ) const

Returns the value type as wxAnyValueType instance.

Remarks

You cannot reliably test whether two wxAnys are of same value type by simply comparing return values of wxAny::GetType(). Instead, use wxAny::HasSameType().

See also

HasSameType()

bool wxAny::HasSameType (const wxAny & *other*) const

Returns true if this and another wxAny have the same value type.

bool wxAny::IsNull ( ) const

Tests if wxAny is null (that is, whether there is no data).

void wxAny::MakeNull ( )

Makes wxAny null (that is, clears it).

bool wxAny::operator!= (signed char *value*) const

bool wxAny::operator!= (signed short *value*) const

bool wxAny::operator!= (signed int *value*) const

bool wxAny::operator!= (signed long *value*) const

bool wxAny::operator!= (wxLongLong_t *value*) const

bool wxAny::operator!= (unsigned char *value*) const

bool wxAny::operator!= (unsigned short *value*) const

bool wxAny::operator!= (unsigned int *value*) const

bool wxAny::operator!= (unsigned long *value*) const

bool wxAny::operator!= (wxULongLong_t *value*) const

bool wxAny::operator!= (float *value*) const

bool wxAny::operator!= (double *value*) const

bool wxAny::operator!= (bool *value*) const

bool wxAny::operator!= (const char * *value*) const

bool wxAny::operator!= (const wchar_t * *value*) const

bool wxAny::operator!= (const **wxString** & *value*) const

template<typename T > **wxAny**& wxAny::operator= (const T & *value*)

wxAny& wxAny::operator= (const **wxAny** & *any*)

wxAny& wxAny::operator= (const **wxVariant** & *variant*)

bool wxAny::operator== (signed char *value*) const

bool wxAny::operator== (signed short *value*) const

bool wxAny::operator== (signed int *value*) const

bool wxAny::operator== (signed long *value*) const

bool wxAny::operator== (wxLongLong_t *value*) const

bool wxAny::operator== (unsigned char *value*) const

bool wxAny::operator== (unsigned short *value*) const

bool wxAny::operator== (unsigned int *value*) const

bool wxAny::operator== (unsigned long *value*) const

bool wxAny::operator== (wxULongLong_t *value*) const

bool wxAny::operator== (float *value*) const

bool wxAny::operator== (double *value*) const

bool wxAny::operator== (bool *value*) const

bool wxAny::operator== (const char * *value*) const

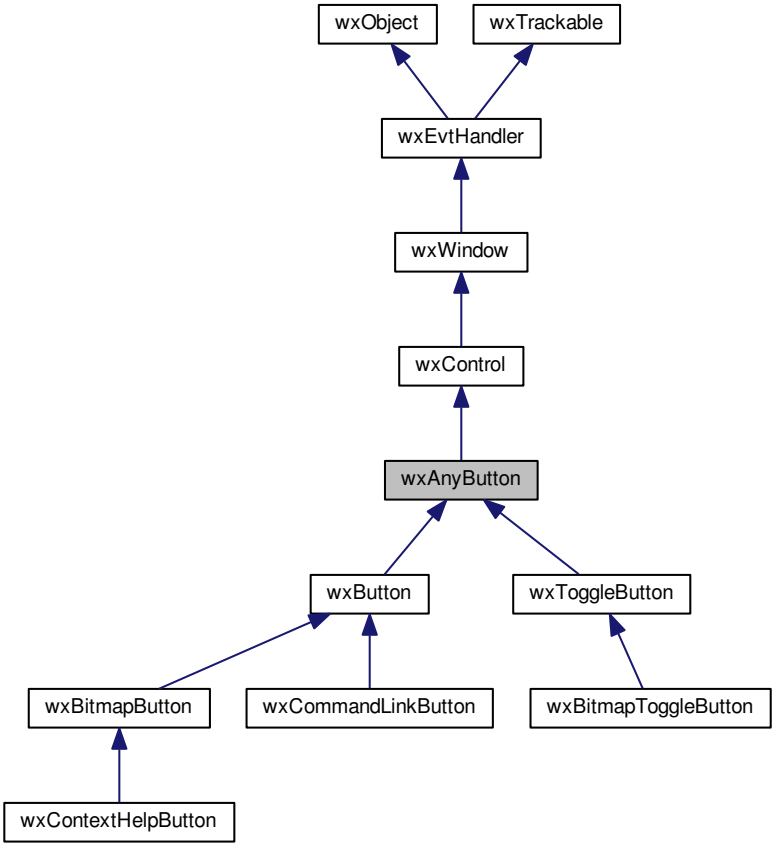
bool wxAny::operator== (const wchar_t * *value*) const

bool wxAny::operator== (const **wxString** & *value*) const

21.21 wxAnyButton Class Reference

#include <wx/anybutton.h>

Inheritance diagram for wxAnyButton:



21.21.1 Detailed Description

A class for common button functionality used as the base for the various button classes.

Public Member Functions

- wxAnyButton ()
- ~wxAnyButton ()
- wxBitmap GetBitmap () const

  Return the bitmap shown by the button.
- wxBitmap GetBitmapCurrent () const

  Returns the bitmap used when the mouse is over the button, which may be invalid.
- wxBitmap GetBitmapDisabled () const

  Returns the bitmap for the disabled state, which may be invalid.
- wxBitmap GetBitmapFocus () const

  Returns the bitmap for the focused state, which may be invalid.
- wxBitmap GetBitmapLabel () const

  Returns the bitmap for the normal state.
- wxBitmap GetBitmapPressed () const

Returns the bitmap for the pressed state, which may be invalid.

- void SetBitmap (const wxBitmap &bitmap, wxDirection dir=wxLEFT)

  Sets the bitmap to display in the button.
- void SetBitmapCurrent (const wxBitmap &bitmap)

  Sets the bitmap to be shown when the mouse is over the button.
- void SetBitmapDisabled (const wxBitmap &bitmap)

  Sets the bitmap for the disabled button appearance.
- void SetBitmapFocus (const wxBitmap &bitmap)

  Sets the bitmap for the button appearance when it has the keyboard focus.
- void SetBitmapLabel (const wxBitmap &bitmap)

  Sets the bitmap label for the button.
- void SetBitmapPressed (const wxBitmap &bitmap)

  Sets the bitmap for the selected (depressed) button appearance.
- wxSize GetBitmapMargins ()

  Get the margins between the bitmap and the text of the button.
- void SetBitmapPosition (wxDirection dir)

  Set the position at which the bitmap is displayed.

- void SetBitmapMargins (wxCoord x, wxCoord y)

  Set the margins between the bitmap and the text of the button.
- void SetBitmapMargins (const wxSize &sz)

  Set the margins between the bitmap and the text of the button.

Additional Inherited Members

21.21.2 Constructor & Destructor Documentation

wxAnyButton::wxAnyButton ( )

wxAnyButton::~wxAnyButton ( )

21.21.3 Member Function Documentation

wxBitmap wxAnyButton::GetBitmap ( ) const

Return the bitmap shown by the button.

The returned bitmap may be invalid only if the button doesn't show any images.

See also

SetBitmap()

Since

2.9.1

wxBitmap wxAnyButton::GetBitmapCurrent ( ) const

Returns the bitmap used when the mouse is over the button, which may be invalid.

See also

SetBitmapCurrent()

Since

2.9.1 (available as wxBitmapButton::GetBitmapHover() in previous versions)

wxBitmap wxAnyButton::GetBitmapDisabled ( ) const

Returns the bitmap for the disabled state, which may be invalid.

See also

SetBitmapDisabled()

Since

2.9.1 (available in wxBitmapButton only in previous versions)

wxBitmap wxAnyButton::GetBitmapFocus ( ) const

Returns the bitmap for the focused state, which may be invalid.

See also

SetBitmapFocus()

Since

2.9.1 (available in wxBitmapButton only in previous versions)

wxBitmap wxAnyButton::GetBitmapLabel ( ) const

Returns the bitmap for the normal state.

This is exactly the same as GetBitmap() but uses a name backwards-compatible with wxBitmapButton.

See also

SetBitmap(), SetBitmapLabel()

Since

2.9.1 (available in wxBitmapButton only in previous versions)

wxSize wxAnyButton::GetBitmapMargins ( )

Get the margins between the bitmap and the text of the button.

See also

SetBitmapMargins()

Since

2.9.1

wxBitmap wxAnyButton::GetBitmapPressed ( ) const

Returns the bitmap for the pressed state, which may be invalid.

See also

SetBitmapPressed()

Since

2.9.1 (available as wxBitmapButton::GetBitmapSelected() in previous versions)

void wxAnyButton::SetBitmap (const wxBitmap & *bitmap*, wxDirection *dir* = wxLEFT)

Sets the bitmap to display in the button.

The bitmap is displayed together with the button label. This method sets up a single bitmap which is used in all button states, use SetBitmapDisabled(), SetBitmapPressed(), SetBitmapCurrent() or SetBitmapFocus() to change the individual images used in different states.

Parameters

bitmap	The bitmap to display in the button. If the bitmap is invalid, any currently shown bitmaps are removed from the button.
dir	The position of the bitmap inside the button. By default it is positioned to the left of the text, near to the left button border. Other possible values include wxRIGHT, wxTOP and wxBOTTOM.

See also

SetBitmapPosition(), SetBitmapMargins()

Since

2.9.1

void wxAnyButton::SetBitmapCurrent (const wxBitmap & *bitmap*)

Sets the bitmap to be shown when the mouse is over the button.

If *bitmap* is invalid, the normal bitmap will be used in the current state.

See also

GetBitmapCurrent()

Since

2.9.1 (available as wxBitmapButton::SetBitmapHover() in previous versions)

void wxAnyButton::SetBitmapDisabled (const wxBitmap & *bitmap*)

Sets the bitmap for the disabled button appearance.

If *bitmap* is invalid, the disabled bitmap is set to the automatically generated greyed out version of the normal bitmap, i.e. the same bitmap as is used by default if this method is not called at all. Use SetBitmap() with an invalid bitmap to remove the bitmap completely (for all states).

See also

GetBitmapDisabled(), SetBitmapLabel(), SetBitmapPressed(), SetBitmapFocus()

Since

2.9.1 (available in wxBitmapButton only in previous versions)

void wxAnyButton::SetBitmapFocus (const wxBitmap & *bitmap*)

Sets the bitmap for the button appearance when it has the keyboard focus.

If *bitmap* is invalid, the normal bitmap will be used in the focused state.

See also

GetBitmapFocus(), SetBitmapLabel(), SetBitmapPressed(), SetBitmapDisabled()

Since

2.9.1 (available in wxBitmapButton only in previous versions)

void wxAnyButton::SetBitmapLabel (const wxBitmap & *bitmap*)

Sets the bitmap label for the button.

Remarks

This is the bitmap used for the unselected state, and for all other states if no other bitmaps are provided.

See also

SetBitmap(), GetBitmapLabel()

Since

2.9.1 (available in wxBitmapButton only in previous versions)

void wxAnyButton::SetBitmapMargins (wxCoord *x*, wxCoord *y*)

Set the margins between the bitmap and the text of the button.

This method is currently only implemented under MSW. If it is not called, default margin is used around the bitmap.

See also

SetBitmap(), SetBitmapPosition()

Since

2.9.1

void wxAnyButton::SetBitmapMargins (const wxSize & *sz*)

Set the margins between the bitmap and the text of the button.

This method is currently only implemented under MSW. If it is not called, default margin is used around the bitmap.

See also

SetBitmap(), SetBitmapPosition()

Since

2.9.1

void wxAnyButton::SetBitmapPosition (wxDirection *dir*)

Set the position at which the bitmap is displayed.

This method should only be called if the button does have an associated bitmap.

Since

2.9.1

Parameters

dir	Direction in which the bitmap should be positioned, one of wxLEFT, wxRIGHT, wxTOP or wxBOTTOM.

void wxAnyButton::SetBitmapPressed (const wxBitmap & *bitmap*)

Sets the bitmap for the selected (depressed) button appearance.

Since

2.9.1 (available as wxBitmapButton::SetBitmapSelected() in previous versions)

21.22 wxAnyValueBuffer Union Reference

`#include <wx/any.h>`

21.22.1 Detailed Description

Type for buffer within wxAny for holding data.

Public Attributes

- void * m_ptr
- wxByte m_buffer [WX_ANY_VALUE_BUFFER_SIZE]

21.22.2 Member Data Documentation

wxByte wxAnyValueBuffer::m_buffer[WX_ANY_VALUE_BUFFER_SIZE]

void* wxAnyValueBuffer::m_ptr

21.23 wxAnyValueType Class Reference

`#include <wx/any.h>`

21.23.1 Detailed Description

wxAnyValueType is base class for value type functionality for C++ data types used with wxAny.

Usually the default template will create a satisfactory wxAnyValueType implementation for a data type, but sometimes you may need to add some customization. To do this you will need to add specialized template of wxAnyValueTypeImpl<>. Often your only need may be to add dynamic type conversion which would be done like this:

```
template<>
class wxAnyValueTypeImpl<MyClass> :
    public wxAnyValueTypeImplBase<MyClass>
{
    WX_DECLARE_ANY_VALUE_TYPE(wxAnyValueTypeImpl<MyClass>)
public:
    wxAnyValueTypeImpl() :
        wxAnyValueTypeImplBase<MyClass>() { }
    virtual ~wxAnyValueTypeImpl() { }

    virtual bool ConvertValue(const wxAnyValueBuffer& src,
                              wxAnyValueType* dstType,
                              wxAnyValueBuffer& dst) const
    {
        // GetValue() is a static member function implemented
        // in wxAnyValueTypeImplBase<>.
        MyClass value = GetValue(src);

        // TODO: Convert value from src buffer to destination
        //       type and buffer. If cannot be done, return
        //       false. This is a simple sample.
        if ( dstType->CheckType<wxString>() )
        {
            wxString s = value.ToString();
            wxAnyValueTypeImpl<wxString>::SetValue(s, dst);
        }
        else
        {
            return false;
        }
    }
};

//
// Following must be placed somewhere in your source code
WX_IMPLEMENT_ANY_VALUE_TYPE(wxAnyValueTypeImpl<MyClass>)
```

wxAnyValueTypeImplBase<> template, from which we inherit in the above example, contains the bulk of the default wxAnyValueTypeImpl<> template implementation, and as such allows you to easily add some minor customization.

If you need a have complete control over the type interpretation, you will need to derive a class directly from wxAnyValueType, like this:

```
template <>
class wxAnyValueTypeImpl<MyClass> : public wxAnyValueType
{
    WX_DECLARE_ANY_VALUE_TYPE(wxAnyValueTypeImpl<MyClass>)
public:
    virtual void DeleteValue(wxAnyValueBuffer& buf) const
    {
        // TODO: Free the data in buffer
        // It is important to clear the buffer like this
        // at the end of DeleteValue().
```

```
        buf.m_ptr = NULL;
    }

    virtual void CopyBuffer(const wxAnyValueBuffer& src,
                            wxAnyValueBuffer& dst) const
    {
        // TODO: Copy value from one buffer to another.
        //       dst is already uninitialized and does not
        //       need to be freed.
    }

    virtual bool ConvertValue(const wxAnyValueBuffer& src,
                              wxAnyValueType* dstType,
                              wxAnyValueBuffer& dst) const
    {
        // TODO: Convert value from src buffer to destination
        //       type and buffer.
    }

    //
    // Following static functions must be implemented
    //

    static void SetValue(const T& value,
                         wxAnyValueBuffer& buf)
    {
        // TODO: Store value into buf.
    }

    static const T& GetValue(const wxAnyValueBuffer& buf)
    {
        // TODO: Return reference to value stored in buffer.
    }
};

//
// Following must be placed somewhere in your source code
WX_IMPLEMENT_ANY_VALUE_TYPE(wxAnyValueTypeImpl<MyClass>)
```

Library: wxBase

Category: Data Structures

See also

wxAny

Public Member Functions

- wxAnyValueType ()

  Default constructor.
- virtual ~wxAnyValueType ()

  Destructor.
- template<typename T >
  bool CheckType () const

  Use this template function for checking if wxAnyValueType represents a specific C++ data type.
- virtual bool ConvertValue (const wxAnyValueBuffer &src, wxAnyValueType *dstType, wxAnyValueBuffer &dst) const =0

  Convert value into buffer of different type.
- virtual void CopyBuffer (const wxAnyValueBuffer &src, wxAnyValueBuffer &dst) const =0

  Implement this for buffer-to-buffer copy.
- virtual void DeleteValue (wxAnyValueBuffer &buf) const =0

  This function is called every time the data in wxAny buffer needs to be freed.
- virtual bool IsSameType (const wxAnyValueType *otherType) const =0

  This function is used for internal type matching.

21.23.2 Constructor & Destructor Documentation

wxAnyValueType::wxAnyValueType ( )

Default constructor.

virtual wxAnyValueType::~wxAnyValueType ( ) [virtual]

Destructor.

21.23.3 Member Function Documentation

template<typename T > bool wxAnyValueType::CheckType ( ) const

Use this template function for checking if wxAnyValueType represents a specific C++ data type.

See also

wxAny::CheckType()

virtual bool wxAnyValueType::ConvertValue (const wxAnyValueBuffer & *src*, wxAnyValueType * *dstType*, wxAnyValueBuffer & *dst*) const [pure virtual]

Convert value into buffer of different type.

Return false if not possible.

virtual void wxAnyValueType::CopyBuffer (const wxAnyValueBuffer & *src*, wxAnyValueBuffer & *dst*) const [pure virtual]

Implement this for buffer-to-buffer copy.

Parameters

src	This is the source data buffer.
dst	This is the destination data buffer that is in either uninitialized or freed state.

virtual void wxAnyValueType::DeleteValue (wxAnyValueBuffer & *buf*) const [pure virtual]

This function is called every time the data in wxAny buffer needs to be freed.

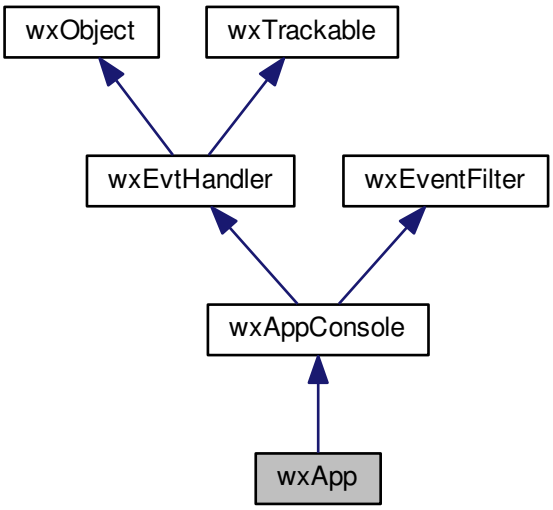
virtual bool wxAnyValueType::IsSameType (const wxAnyValueType * *otherType*) const [pure virtual]

This function is used for internal type matching.

21.24 wxApp Class Reference

```
#include <wx/app.h>
```

Inheritance diagram for wxApp:



21.24.1 Detailed Description

The wxApp class represents the application itself when `wxUSE_GUI=1`.

In addition to the features provided by wxAppConsole it keeps track of the *top window* (see SetTopWindow()) and adds support for video modes (see SetVideoMode()).

In general, application-wide settings for GUI-only apps are accessible from wxApp (or from wxSystemSettings or wxSystemOptions classes).

Events emitted by this class

Event macros for events emitted by this class:

- EVT_QUERY_END_SESSION(func): Process a query end session event, supplying the member function. See wxCloseEvent.
- EVT_END_SESSION(func): Process an end session event, supplying the member function. See wxCloseEvent.
- EVT_ACTIVATE_APP(func): Process a `wxEVT_ACTIVATE_APP` event. See wxActivateEvent.
- EVT_HIBERNATE(func): Process a hibernate event. See wxActivateEvent.
- EVT_DIALUP_CONNECTED(func): A connection with the network was established. See wxDialUpEvent.
- EVT_DIALUP_DISCONNECTED(func): The connection with the network was lost. See wxDialUpEvent.
- EVT_IDLE(func): Process a `wxEVT_IDLE` event. See wxIdleEvent.

Library: wxBase

Category: Application and Process Management

See also

wxApp Overview, wxAppTraits, wxEventLoopBase, wxSystemSettings

Public Member Functions

- wxApp ()

  Constructor.
- virtual ~wxApp ()

  Destructor.
- virtual wxVideoMode GetDisplayMode () const

  Get display mode that is used use.
- bool GetExitOnFrameDelete () const

  Returns true if the application will exit when the top-level frame is deleted.
- virtual wxLayoutDirection GetLayoutDirection () const

  Return the layout direction for the current locale or `wxLayout_Default` if it's unknown.
- bool GetUseBestVisual () const

  Returns true if the application will use the best visual on systems that support different visuals, false otherwise.
- virtual wxWindow * GetTopWindow () const

  Returns a pointer to the top window.
- virtual bool IsActive () const

  Returns true if the application is active, i.e. if one of its windows is currently in the foreground.
- virtual bool SafeYield (wxWindow *win, bool onlyIfNeeded)

  This function is similar to wxYield(), except that it disables the user input to all program windows before calling wxAppConsole::Yield and re-enables it again afterwards.
- virtual bool SafeYieldFor (wxWindow *win, long eventsToProcess)

  Works like SafeYield() with onlyIfNeeded == true except that it allows the caller to specify a mask of events to be processed.
- bool ProcessMessage (WXMSG *msg)

  Windows-only function for processing a message.
- virtual bool SetDisplayMode (const wxVideoMode &info)

  Set display mode to use.
- void SetExitOnFrameDelete (bool flag)

  Allows the programmer to specify whether the application will exit when the top-level frame is deleted.
- virtual bool SetNativeTheme (const wxString &theme)

  Allows runtime switching of the UI environment theme.
- void SetTopWindow (wxWindow *window)

  Sets the 'top' window.
- void SetUseBestVisual (bool flag, bool forceTrueColour=false)

  Allows the programmer to specify whether the application will use the best visual on systems that support several visual on the same display.

Mac-specific functions

- virtual void MacNewFile ()

  Called in response of an "open-application" Apple event.
- virtual void MacOpenFiles (const wxArrayString &fileNames)

  Called in response of an openFiles message with Cocoa, or an "open-document" Apple event with Carbon.

- virtual void MacOpenFile (const wxString &fileName)
  Called in response of an "open-document" Apple event.
- virtual void MacOpenURL (const wxString &url)
  Called in response of a "get-url" Apple event.
- virtual void MacPrintFile (const wxString &fileName)
  Called in response of a "print-document" Apple event.
- virtual void MacReopenApp ()
  Called in response of a "reopen-application" Apple event.
- virtual bool OSXIsGUIApplication ()
  May be overridden to indicate that the application is not a foreground GUI application under OS X.

Additional Inherited Members

21.24.2 Constructor & Destructor Documentation

wxApp::wxApp ( )

Constructor.

Called implicitly with a definition of a wxApp object.

virtual wxApp::~wxApp ( ) [virtual]

Destructor.

Will be called implicitly on program exit if the wxApp object is created on the stack.

21.24.3 Member Function Documentation

virtual wxVideoMode wxApp::GetDisplayMode ( ) const [virtual]

Get display mode that is used use.

This is only used in framebuffer wxWidgets ports such as wxDFB.

bool wxApp::GetExitOnFrameDelete ( ) const

Returns true if the application will exit when the top-level frame is deleted.

See also

SetExitOnFrameDelete()

virtual wxLayoutDirection wxApp::GetLayoutDirection ( ) const [virtual]

Return the layout direction for the current locale or `wxLayout_Default` if it's unknown.

virtual wxWindow* wxApp::GetTopWindow ( ) const [virtual]

Returns a pointer to the top window.

Remarks

If the top window hasn't been set using SetTopWindow(), this function will find the first top-level window (frame or dialog or instance of wxTopLevelWindow) from the internal top level window list and return that.

See also

SetTopWindow()

bool wxApp::GetUseBestVisual ( ) const

Returns true if the application will use the best visual on systems that support different visuals, false otherwise.

See also

SetUseBestVisual()

virtual bool wxApp::IsActive ( ) const `[virtual]`

Returns true if the application is active, i.e. if one of its windows is currently in the foreground.

If this function returns false and you need to attract users attention to the application, you may use wxTopLevelWindow::RequestUserAttention to do it.

virtual void wxApp::MacNewFile ( ) `[virtual]`

Called in response of an "open-application" Apple event.

Override this to create a new document in your app.

Availability: only available for the wxOSX port.

virtual void wxApp::MacOpenFile (const wxString & *fileName*) `[virtual]`

Called in response of an "open-document" Apple event.

Deprecated This function is kept mostly for backwards compatibility. Please override wxApp::MacOpenFiles method instead in any new code.

Availability: only available for the wxOSX port.

virtual void wxApp::MacOpenFiles (const wxArrayString & *fileNames*) `[virtual]`

Called in response of an openFiles message with Cocoa, or an "open-document" Apple event with Carbon.

You need to override this method in order to open one or more document files after the user double clicked on it or if the files and/or folders were dropped on either the application in the dock or the application icon in Finder.

By default this method calls MacOpenFile for each file/folder.

Availability: only available for the wxOSX port.

Since

2.9.3

virtual void wxApp::MacOpenURL (const wxString & *url*) `[virtual]`

Called in response of a "get-url" Apple event.

Availability: only available for the wxOSX port.

virtual void wxApp::MacPrintFile (const wxString & *fileName*) [virtual]

Called in response of a "print-document" Apple event.

Availability: only available for the wxOSX port.

virtual void wxApp::MacReopenApp ( ) [virtual]

Called in response of a "reopen-application" Apple event.

Availability: only available for the wxOSX port.

virtual bool wxApp::OSXIsGUIApplication ( ) [virtual]

May be overridden to indicate that the application is not a foreground GUI application under OS X.

This method is called during the application startup and returns true by default. In this case, wxWidgets ensures that the application is ran as a foreground, GUI application so that the user can interact with it normally, even if it is not bundled. If this is undesired, i.e. if the application doesn't need to be brought to the foreground, this method can be overridden to return false.

Notice that overriding it doesn't make any difference for the bundled applications which are always foreground unless `LSBackgroundOnly` key is specified in the `Info.plist` file.

Availability: only available for the wxOSX port.

Since

3.0.1

bool wxApp::ProcessMessage (WXMSG * *msg*)

Windows-only function for processing a message.

This function is called from the main message loop, checking for windows that may wish to process it.

The function returns true if the message was processed, false otherwise. If you use wxWidgets with another class library with its own message loop, you should make sure that this function is called to allow wxWidgets to receive messages. For example, to allow co-existence with the Microsoft Foundation Classes, override the PreTranslate↩ Message function:

```
// Provide wxWidgets message loop compatibility
BOOL CTheApp::PreTranslateMessage(MSG *msg)
{
    if (wxTheApp && wxTheApp->ProcessMessage((WXMSW *)msg))
        return true;
    else
        return CWinApp::PreTranslateMessage(msg);
}
```

Availability: only available for the wxMSW port.

virtual bool wxApp::SafeYield (wxWindow * *win*, bool *onlyIfNeeded*) [virtual]

This function is similar to wxYield(), except that it disables the user input to all program windows before calling wxAppConsole::Yield and re-enables it again afterwards.

If *win* is not NULL, this window will remain enabled, allowing the implementation of some limited user interaction. Returns the result of the call to wxAppConsole::Yield.

See also

wxSafeYield

virtual bool wxApp::SafeYieldFor (wxWindow * *win*, long *eventsToProcess*) [virtual]

Works like SafeYield() with *onlyIfNeeded* == true except that it allows the caller to specify a mask of events to be processed.

See wxAppConsole::YieldFor for more info.

virtual bool wxApp::SetDisplayMode (const wxVideoMode & *info*) [virtual]

Set display mode to use.

This is only used in framebuffer wxWidgets ports such as wxDFB.

void wxApp::SetExitOnFrameDelete (bool *flag*)

Allows the programmer to specify whether the application will exit when the top-level frame is deleted.

Parameters

flag	If true (the default), the application will exit when the top-level frame is deleted. If false, the application will continue to run.

See also

GetExitOnFrameDelete(), Application Shutdown

virtual bool wxApp::SetNativeTheme (const wxString & *theme*) [virtual]

Allows runtime switching of the UI environment theme.

Currently implemented for wxGTK2-only. Return true if theme was successfully changed.

Parameters

theme	The name of the new theme or an absolute path to a gtkrc-theme-file

void wxApp::SetTopWindow (wxWindow * *window*)

Sets the 'top' window.

You can call this from within OnInit() to let wxWidgets know which is the main window. You don't have to set the top window; it is only a convenience so that (for example) certain dialogs without parents can use a specific window as the top window.

If no top window is specified by the application, wxWidgets just uses the first frame or dialog (or better, any wxTopLevelWindow) in its top-level window list, when it needs to use the top window. If you previously called SetTopWindow() and now you need to restore this automatic behaviour you can call

```
wxApp::SetTopWindow(NULL)
```

.

Parameters

window	The new top window.

See also

GetTopWindow(), OnInit()

void wxApp::SetUseBestVisual (bool *flag,* bool *forceTrueColour* = `false`)

Allows the programmer to specify whether the application will use the best visual on systems that support several visual on the same display.

This is typically the case under Solaris and IRIX, where the default visual is only 8-bit whereas certain applications are supposed to run in TrueColour mode.

Note that this function has to be called in the constructor of the wxApp instance and won't have any effect when called later on. This function currently only has effect under GTK.

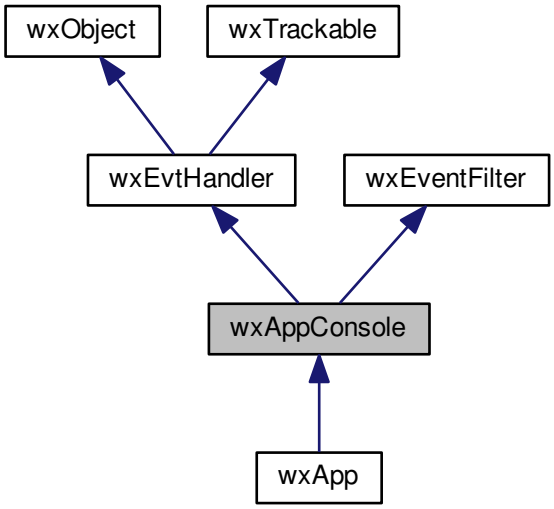
Parameters

flag	If true, the app will use the best visual.
forceTrueColour	If true then the application will try to force using a TrueColour visual and abort the app if none is found.

21.25 wxAppConsole Class Reference

`#include <wx/app.h>`

Inheritance diagram for wxAppConsole:



21.25.1 Detailed Description

This class is essential for writing console-only or hybrid apps without having to define `wxUSE_GUI=0`.

It is used to:

- set and get application-wide properties (see wxAppConsole::CreateTraits and wxAppConsole::SetXXX functions)
- implement the windowing system message or event loop: events in fact are supported even in console-mode applications (see wxAppConsole::HandleEvent and wxAppConsole::ProcessPendingEvents);
- initiate application processing via wxApp::OnInit;
- allow default processing of events not handled by other objects in the application (see wxAppConsole::FilterEvent)
- implement Apple-specific event handlers (see wxAppConsole::MacXXX functions)

You should use the macro wxIMPLEMENT_APP(appClass) in your application implementation file to tell wxWidgets how to create an instance of your application class.

Use wxDECLARE_APP(appClass) in a header file if you want the wxGetApp() function (which returns a reference to your application object) to be visible to other files.

Library: wxBase

Category: Application and Process Management

See also

wxApp Overview, wxApp, wxAppTraits, wxEventLoopBase

Public Member Functions

- virtual ~wxAppConsole ()

  Destructor.
- bool Yield (bool onlyIfNeeded=false)
- void SetCLocale ()

  Sets the C locale to the default locale for the current environment.

Event-handling

Note that you should look at wxEvtLoopBase for more event-processing documentation.

- virtual int MainLoop ()

  Called by wxWidgets on creation of the application.
- virtual void ExitMainLoop ()

  Call this to explicitly exit the main message (event) loop.
- virtual int FilterEvent (wxEvent &event)

  Overridden wxEventFilter method.
- wxEventLoopBase * GetMainLoop () const

  Returns the main event loop instance, i.e. the event loop which is started by OnRun() and which dispatches all events sent from the native toolkit to the application (except when new event loops are temporarily set-up).
- virtual void HandleEvent (wxEvtHandler *handler, wxEventFunction func, wxEvent &event) const

  This function simply invokes the given method func of the specified event handler handler with the event as parameter.
- virtual bool UsesEventLoop () const

  Returns true if the application is using an event loop.

Pending events

Pending events are handled by wxAppConsole rather than wxEventLoopBase to allow queuing of events even when there's no event loop (e.g.

in wxAppConsole::OnInit).

- virtual void ProcessPendingEvents ()

  Process all pending events; it is necessary to call this function to process events posted with wxEvtHandler::QueueEvent or wxEvtHandler::AddPendingEvent.
- void DeletePendingEvents ()

  Deletes the pending events of all wxEvtHandlers of this application.
- bool HasPendingEvents () const

  Returns true if there are pending events on the internal pending event list.
- void SuspendProcessingOfPendingEvents ()

  Temporary suspends processing of the pending events.
- void ResumeProcessingOfPendingEvents ()

  Resume processing of the pending events previously stopped because of a call to SuspendProcessingOfPendingEvents().

- void ScheduleForDestruction (wxObject *object)

  Delayed objects destruction.
- bool IsScheduledForDestruction (wxObject *object) const

  Check if the object had been scheduled for destruction with ScheduleForDestruction().

Callbacks for application-wide "events"

- virtual void OnAssertFailure (const wxChar *file, int line, const wxChar *func, const wxChar *cond, const wxChar *msg)

  This function is called when an assert failure occurs, i.e. the condition specified in wxASSERT() macro evaluated to false.
- virtual bool OnCmdLineError (wxCmdLineParser &parser)

  Called when command line parsing fails (i.e. an incorrect command line option was specified by the user).
- virtual bool OnCmdLineHelp (wxCmdLineParser &parser)

  Called when the help option (`-help`) was specified on the command line.
- virtual bool OnCmdLineParsed (wxCmdLineParser &parser)

  Called after the command line had been successfully parsed.
- virtual void OnEventLoopEnter (wxEventLoopBase *loop)

  Called by wxEventLoopBase::SetActive(): you can override this function and put here the code which needs an active event loop.
- virtual void OnEventLoopExit (wxEventLoopBase *loop)

  Called by wxEventLoopBase::OnExit() for each event loop which is exited.
- virtual int OnExit ()

  Override this member function for any processing which needs to be done as the application is about to exit.
- virtual void OnFatalException ()

  This function may be called if something fatal happens: an unhandled exception under Win32 or a fatal signal under Unix, for example.
- virtual bool OnInit ()

  This must be provided by the application, and will usually create the application's main window, optionally calling SetTopWindow().
- virtual void OnInitCmdLine (wxCmdLineParser &parser)

  Called from OnInit() and may be used to initialize the parser with the command line options for this application.
- virtual int OnRun ()

  This virtual function is where the execution of a program written in wxWidgets starts.

Exceptions support

Methods related to C++ exceptions handling.

See also

overview_exceptions

- virtual bool OnExceptionInMainLoop ()

  This function is called if an unhandled exception occurs inside the main application event loop.
- virtual void OnUnhandledException ()

  This function is called when an unhandled C++ exception occurs in user code called by wxWidgets.
- virtual bool StoreCurrentException ()

  Method to store exceptions not handled by OnExceptionInMainLoop().
- virtual void RethrowStoredException ()

  Method to rethrow exceptions stored by StoreCurrentException().

Application information

- wxString GetAppDisplayName () const

  Returns the user-readable application name.
- wxString GetAppName () const

  Returns the application name.
- wxString GetClassName () const

  Gets the class name of the application.
- wxAppTraits * GetTraits ()

  Returns a pointer to the wxAppTraits object for the application.
- const wxString & GetVendorDisplayName () const

  Returns the user-readable vendor name.
- const wxString & GetVendorName () const

  Returns the application's vendor name.
- void SetAppDisplayName (const wxString &name)

  Set the application name to be used in the user-visible places such as window titles.
- void SetAppName (const wxString &name)

  Sets the name of the application.
- void SetClassName (const wxString &name)

  Sets the class name of the application.
- void SetVendorDisplayName (const wxString &name)

  Set the vendor name to be used in the user-visible places.
- void SetVendorName (const wxString &name)

  Sets the name of application's vendor.

Static Public Member Functions

- static void SetInstance (wxAppConsole *app)

  Allows external code to modify global wxTheApp, but you should really know what you're doing if you call it.
- static wxAppConsole * GetInstance ()

  Returns the one and only global application object.
- static bool IsMainLoopRunning ()

  Returns true if the main event loop is currently running, i.e. if the application is inside OnRun().

Public Attributes

- int argc

  Number of command line arguments (after environment-specific processing).
- wxChar ** argv

  Command line arguments (after environment-specific processing).

Protected Member Functions

- virtual wxAppTraits * CreateTraits ()

  Creates the wxAppTraits object when GetTraits() needs it for the first time.

Additional Inherited Members

21.25.2 Constructor & Destructor Documentation

virtual wxAppConsole::~wxAppConsole ( ) [virtual]

Destructor.

21.25.3 Member Function Documentation

virtual wxAppTraits* wxAppConsole::CreateTraits ( ) [protected], [virtual]

Creates the wxAppTraits object when GetTraits() needs it for the first time.

See also

wxAppTraits

void wxAppConsole::DeletePendingEvents ( )

Deletes the pending events of all wxEvtHandlers of this application.

See wxEvtHandler::DeletePendingEvents() for warnings about deleting the pending events.

virtual void wxAppConsole::ExitMainLoop ( ) [virtual]

Call this to explicitly exit the main message (event) loop.

You should normally exit the main loop (and the application) by deleting the top window.

This function simply calls wxEvtLoopBase::Exit() on the active loop.

virtual int wxAppConsole::FilterEvent (wxEvent & *event*) [virtual]

Overridden wxEventFilter method.

This function is called before processing any event and allows the application to preempt the processing of some events, see wxEventFilter documentation for more information.

wxApp implementation of this method always return -1 indicating that the event should be processed normally.

Implements wxEventFilter.

wxString wxAppConsole::GetAppDisplayName ( ) const

Returns the user-readable application name.

The difference between this string and the one returned by GetAppName() is that this one is meant to be shown to the user and so should be used for the window titles, page headers and so on while the other one should be only used internally, e.g. for the file names or configuration file keys.

If the application name for display had been previously set by SetAppDisplayName(), it will be returned by this function. Otherwise, if SetAppName() had been called its value will be returned; also as is. Finally if none was called, this function returns the program name capitalized using wxString::Capitalize().

Since

2.9.0

wxString wxAppConsole::GetAppName ( ) const

Returns the application name.

If SetAppName() had been called, returns the string passed to it. Otherwise returns the program name, i.e. the value of `argv`[0] passed to the `main()` function.

See also

GetAppDisplayName()

wxString wxAppConsole::GetClassName ( ) const

Gets the class name of the application.

The class name may be used in a platform specific manner to refer to the application.

See also

SetClassName()

static wxAppConsole* wxAppConsole::GetInstance ( ) [static]

Returns the one and only global application object.

Usually wxTheApp is used instead.

See also

SetInstance()

wxEventLoopBase* wxAppConsole::GetMainLoop ( ) const

Returns the main event loop instance, i.e. the event loop which is started by OnRun() and which dispatches all events sent from the native toolkit to the application (except when new event loops are temporarily set-up).

The returned value maybe NULL. Put initialization code which needs a non-NULL main event loop into OnEvent↩LoopEnter().

wxAppTraits* wxAppConsole::GetTraits ( )

Returns a pointer to the wxAppTraits object for the application.

If you want to customize the wxAppTraits object, you must override the CreateTraits() function.

const wxString& wxAppConsole::GetVendorDisplayName ( ) const

Returns the user-readable vendor name.

The difference between this string and the one returned by GetVendorName() is that this one is meant to be shown to the user and so should be used for the window titles, page headers and so on while the other one should be only used internally, e.g. for the file names or configuration file keys.

By default, returns the same string as GetVendorName().

Since

2.9.0

const wxString& wxAppConsole::GetVendorName ( ) const

Returns the application's vendor name.

virtual void wxAppConsole::HandleEvent (wxEvtHandler * *handler,* wxEventFunction *func,* wxEvent & *event*) const `[virtual]`

This function simply invokes the given method *func* of the specified event handler *handler* with the *event* as parameter.

It exists solely to allow to catch the C++ exceptions which could be thrown by all event handlers in the application in one place: if you want to do this, override this function in your wxApp-derived class and add try/catch clause(s) to it.

bool wxAppConsole::HasPendingEvents ( ) const

Returns true if there are pending events on the internal pending event list.

Whenever wxEvtHandler::QueueEvent or wxEvtHandler::AddPendingEvent() are called (not only for wxApp itself, but for any event handler of the application!), the internal wxApp's list of handlers with pending events is updated and this function will return true.

static bool wxAppConsole::IsMainLoopRunning ( ) `[static]`

Returns true if the main event loop is currently running, i.e. if the application is inside OnRun().

This can be useful to test whether events can be dispatched. For example, if this function returns false, non-blocking sockets cannot be used because the events from them would never be processed.

bool wxAppConsole::IsScheduledForDestruction (wxObject * *object*) const

Check if the object had been scheduled for destruction with ScheduleForDestruction().

This function may be useful as an optimization to avoid doing something with an object which will be soon destroyed in any case.

virtual int wxAppConsole::MainLoop ( ) `[virtual]`

Called by wxWidgets on creation of the application.

Override this if you wish to provide your own (environment-dependent) main loop.

Returns

0 under X, and the wParam of the WM_QUIT message under Windows.

virtual void wxAppConsole::OnAssertFailure (const wxChar * *file*, int *line*, const wxChar * *func*, const wxChar * *cond*, const wxChar * *msg*) `[virtual]`

This function is called when an assert failure occurs, i.e. the condition specified in wxASSERT() macro evaluated to false.

It is only called in debug mode (when **WXDEBUG** is defined) as asserts are not left in the release code at all. The base class version shows the default assert failure dialog box proposing to the user to stop the program, continue or ignore all subsequent asserts.

Parameters

file	the name of the source file where the assert occurred
line	the line number in this file where the assert occurred
func	the name of the function where the assert occurred, may be empty if the compiler doesn't support C99 **FUNCTION**
cond	the condition of the failed assert in text form
msg	the message specified as argument to wxASSERT_MSG or wxFAIL_MSG, will be NULL if just wxASSERT or wxFAIL was used

virtual bool wxAppConsole::OnCmdLineError (wxCmdLineParser & *parser*) `[virtual]`

Called when command line parsing fails (i.e. an incorrect command line option was specified by the user).

The default behaviour is to show the program usage text and abort the program.

Return true to continue normal execution or false to return false from OnInit() thus terminating the program.

See also

OnInitCmdLine()

virtual bool wxAppConsole::OnCmdLineHelp (wxCmdLineParser & *parser*) `[virtual]`

Called when the help option (`-help`) was specified on the command line.

The default behaviour is to show the program usage text and abort the program.

Return true to continue normal execution or false to return false from OnInit() thus terminating the program.

See also

OnInitCmdLine()

virtual bool wxAppConsole::OnCmdLineParsed (wxCmdLineParser & *parser*) `[virtual]`

Called after the command line had been successfully parsed.

You may override this method to test for the values of the various parameters which could be set from the command line.

Don't forget to call the base class version unless you want to suppress processing of the standard command line options. Return true to continue normal execution or false to return false from OnInit() thus terminating the program.

See also

OnInitCmdLine()

virtual void wxAppConsole::OnEventLoopEnter (wxEventLoopBase * *loop*) [virtual]

Called by wxEventLoopBase::SetActive(): you can override this function and put here the code which needs an active event loop.

Note that this function is called whenever an event loop is activated; you may want to use wxEventLoopBase::IsMain() to perform initialization specific for the app's main event loop.

See also

OnEventLoopExit()

virtual void wxAppConsole::OnEventLoopExit (wxEventLoopBase * *loop*) [virtual]

Called by wxEventLoopBase::OnExit() for each event loop which is exited.

See also

OnEventLoopEnter()

virtual bool wxAppConsole::OnExceptionInMainLoop ( ) [virtual]

This function is called if an unhandled exception occurs inside the main application event loop.

It can return true to ignore the exception and to continue running the loop or false to exit the loop and terminate the program.

The default behaviour of this function is the latter in all ports except under Windows where a dialog is shown to the user which allows him to choose between the different options. You may override this function in your class to do something more appropriate.

If this method rethrows the exception and if the exception can't be stored for later processing using StoreCurrentException(), the program will terminate after calling OnUnhandledException().

You should consider overriding this method to perform whichever last resort exception handling that would be done in a typical C++ program in a `try/catch` block around the entire `main()` function. As this method is called during exception handling, you may use the C++ `throw` keyword to rethrow the current exception to catch it again and analyze it. For example:

```
class MyApp : public wxApp {
public:
    virtual bool OnExceptionInMainLoop()
    {
        wxString error;
        try {
            throw; // Rethrow the current exception.
        } catch (const MyException& e) {
            error = e.GetMyErrorMessage();
        } catch (const std::exception& e) {
            error = e.what();
        } catch ( ... ) {
            error = "unknown error.";
        }

        wxLogError("Unexpected exception has occurred: %s, the program will terminate.", error);

        // Exit the main loop and thus terminate the program.
        return false;
    }
};
```

virtual int wxAppConsole::OnExit ( ) [virtual]

Override this member function for any processing which needs to be done as the application is about to exit.

OnExit is called after destroying all application windows and controls, but before wxWidgets cleanup. Note that it is not called at all if OnInit() failed.

The return value of this function is currently ignored, return the same value as returned by the base class method if you override it.

virtual void wxAppConsole::OnFatalException ( ) [virtual]

This function may be called if something fatal happens: an unhandled exception under Win32 or a fatal signal under Unix, for example.

However, this will not happen by default: you have to explicitly call wxHandleFatalExceptions() to enable this.

Generally speaking, this function should only show a message to the user and return. You may attempt to save unsaved data but this is not guaranteed to work and, in fact, probably won't.

See also

wxHandleFatalExceptions()

virtual bool wxAppConsole::OnInit ( ) [virtual]

This must be provided by the application, and will usually create the application's main window, optionally calling SetTopWindow().

You may use OnExit() to clean up anything initialized here, provided that the function returns true.

Notice that if you want to use the command line processing provided by wxWidgets you have to call the base class version in the derived class OnInit().

Return true to continue processing, false to exit the application immediately.

virtual void wxAppConsole::OnInitCmdLine (wxCmdLineParser & *parser*) [virtual]

Called from OnInit() and may be used to initialize the parser with the command line options for this application.

The base class versions adds support for a few standard options only.

virtual int wxAppConsole::OnRun ( ) [virtual]

This virtual function is where the execution of a program written in wxWidgets starts.

The default implementation just enters the main loop and starts handling the events until it terminates, either because ExitMainLoop() has been explicitly called or because the last frame has been deleted and GetExitOnFrame↩ Delete() flag is true (this is the default).

The return value of this function becomes the exit code of the program, so it should return 0 in case of successful termination.

virtual void wxAppConsole::OnUnhandledException ( ) [virtual]

This function is called when an unhandled C++ exception occurs in user code called by wxWidgets.

Any unhandled exceptions thrown from (overridden versions of) OnInit() and OnExit() methods as well as any exceptions thrown from inside the main loop and re-thrown by OnUnhandledException() will result in a call to this function.

By the time this function is called, the program is already about to exit and the exception can't be handled nor ignored any more, override OnUnhandledException() or use explicit `try/catch` blocks around OnInit() body to be able to handle the exception earlier.

The default implementation dumps information about the exception using wxMessageOutputBest.

virtual void wxAppConsole::ProcessPendingEvents ( ) [virtual]

Process all pending events; it is necessary to call this function to process events posted with wxEvtHandler::QueueEvent or wxEvtHandler::AddPendingEvent.

This happens during each event loop iteration (see wxEventLoopBase) in GUI mode but it may be also called directly.

Note that this function does not only process the pending events for the wxApp object itself (which derives from wxEvtHandler) but also the pending events for *any* event handler of this application.

This function will immediately return and do nothing if SuspendProcessingOfPendingEvents() was called.

void wxAppConsole::ResumeProcessingOfPendingEvents ( )

Resume processing of the pending events previously stopped because of a call to SuspendProcessingOfPendingEvents().

virtual void wxAppConsole::RethrowStoredException ( ) [virtual]

Method to rethrow exceptions stored by StoreCurrentException().

Note

Just as with StoreCurrentException(), it is usually not necessary to override this method when using C++11.

If StoreCurrentException() is overridden, this function should be overridden as well to rethrow the exceptions stored by it when the control gets back to our code, i.e. when it's safe to do it.

See StoreCurrentException() for an example of implementing this method.

The default version does nothing when using C++98 and uses std::rethrow_exception() in C++11.

Since

3.1.0

void wxAppConsole::ScheduleForDestruction (wxObject * *object*)

Delayed objects destruction.

In applications using events it may be unsafe for an event handler to delete the object which generated the event because more events may be still pending for the same object. In this case the handler may call ScheduleForDestruction() instead. Schedule the object for destruction in the near future.

Notice that if the application is not using an event loop, i.e. if UsesEventLoop() returns false, this method will simply delete the object immediately.

Examples of using this function inside wxWidgets itself include deleting the top level windows when they are closed and sockets when they are disconnected.

void wxAppConsole::SetAppDisplayName (const wxString & *name*)

Set the application name to be used in the user-visible places such as window titles.

See GetAppDisplayName() for more about the differences between the display name and name.

Notice that if this function is called, the name is used as is, without any capitalization as done by default by GetAppDisplayName().

void wxAppConsole::SetAppName (const wxString & *name*)

Sets the name of the application.

This name should be used for file names, configuration file entries and other internal strings. For the user-visible strings, such as the window titles, the application display name set by SetAppDisplayName() is used instead.

By default the application name is set to the name of its executable file.

See also

GetAppName()

void wxAppConsole::SetClassName (const wxString & *name*)

Sets the class name of the application.

This may be used in a platform specific manner to refer to the application.

See also

GetClassName()

void wxAppConsole::SetCLocale ( )

Sets the C locale to the default locale for the current environment.

It is advised to call this to ensure that the underlying toolkit uses the locale in which the numbers and monetary amounts are shown in the format expected by user and so on.

Calling this function is roughly equivalent to calling

```
setlocale(LC_ALL, "");
```

but performs additional toolkit-specific tasks under some platforms and so should be used instead of `setlocale()` itself. Alternatively, you can use wxLocale to change the locale with more control.

Notice that this does *not* change the global C++ locale, you need to do it explicitly if you want, e.g.

```
std::locale::global(std::locale(""));
```

but be warned that locale support in C++ standard library can be poor or worse under some platforms, e.g. the above line results in an immediate crash under OS X up to the version 10.8.2.

Since

2.9.5

static void wxAppConsole::SetInstance (wxAppConsole * *app*) [static]

Allows external code to modify global wxTheApp, but you should really know what you're doing if you call it.

Parameters

app	Replacement for the global application object.

See also

GetInstance()

void wxAppConsole::SetVendorDisplayName (const wxString & *name*)

Set the vendor name to be used in the user-visible places.

See GetVendorDisplayName() for more about the differences between the display name and name.

void wxAppConsole::SetVendorName (const wxString & *name*)

Sets the name of application's vendor.

The name will be used in registry access. A default name is set by wxWidgets.

See also

GetVendorName()

virtual bool wxAppConsole::StoreCurrentException ( ) [virtual]

Method to store exceptions not handled by OnExceptionInMainLoop().

Note

The default implementation of this function when using C++98 compiler just returns false, as there is no generic way to store an arbitrary exception in C++98 and each application must do it on its own for the exceptions it uses in its overridden version. When using C++11, the default implementation uses std::current_exception() and returns true, so it's normally not necessary to override this method when using C++11.

This function can be overridden to store the current exception, in view of rethrowing it later when RethrowStoredException() is called. If the exception was stored, return true. If the exception can't be stored, i.e. if this function returns false, the program will abort after calling OnUnhandledException().

It is necessary to override this function if OnExceptionInMainLoop() doesn't catch all exceptions, but you still want to handle them using explicit `try/catch` statements. Typical use could be to allow code like the following to work:

```
void MyFrame::SomeFunction()
{
    try {
        MyDialog dlg(this);
        dlg.ShowModal();
    } catch ( const MyExpectedException& e ) {
        // Deal with the exceptions thrown from the dialog.
    }
}
```

By default, throwing an exception from an event handler called from the dialog modal event loop would terminate the application as the exception can't be safely propagated to the code in the catch clause because of the presence of the native system functions (through which C++ exceptions can't, generally speaking, propagate) in the call stack between them.

Overriding this method allows the exception to be stored when it is detected and rethrown using RethrowStoredException() when the native system function dispatching the dialog events terminates, with the result that the code above works as expected.

An example of implementing this method:

```
class MyApp : public wxApp {
public:
    virtual bool StoreCurrentException()
    {
        try {
            throw;
        } catch ( const std::runtime_exception& e ) {
            if ( !m_runtimeError.empty() ) {
                // This is not supposed to happen, only one exception,
                // at most, should be stored.
                return false;
            }

            m_runtimeError = e.what();

            // Don't terminate, let our code handle this exception later.
            return true;
        } catch ( ... ) {
            // This could be extended to store information about any
            // other exceptions too, but if we don't store them, we
            // should return false to let the program die.
        }

        return false;
    }

    virtual void RethrowStoredException()
    {
        if ( !m_runtimeError.empty() ) {
            std::runtime_exception e(m_runtimeError);
            m_runtimeError.clear();
            throw e;
        }
    }

private:
    std::string m_runtimeError;
};
```

See also

OnExceptionInMainLoop(), RethrowStoredException()

Since

3.1.0

void wxAppConsole::SuspendProcessingOfPendingEvents ( )

Temporary suspends processing of the pending events.

See also

ResumeProcessingOfPendingEvents()

virtual bool wxAppConsole::UsesEventLoop ( ) const [virtual]

Returns true if the application is using an event loop.

This function always returns true for the GUI applications which must use an event loop but by default only returns true for the console programs if an event loop is already running as it can't know whether one will be created in the future.

Thus, it only makes sense to override it in console applications which do use an event loop, to return true instead of checking if there is a currently active event loop.

bool wxAppConsole::Yield (bool *onlyIfNeeded* = `false`)

21.25.4 Member Data Documentation

int wxAppConsole::argc

Number of command line arguments (after environment-specific processing).

wxChar∗∗ wxAppConsole::argv

Command line arguments (after environment-specific processing).

Under Windows and Linux/Unix, you should parse the command line arguments and check for files to be opened when starting your application. Under OS X, you need to override MacOpenFiles() since command line arguments are used differently there.

You may use the wxCmdLineParser to parse command line arguments.

21.26 wxAppProgressIndicator Class Reference

`#include <wx/appprogress.h>`

21.26.1 Detailed Description

A helper class that can be used to update the progress bar in the taskbar button.

Library: wxCore

Category: Miscellaneous

Availability: only available for the wxMSW port.

See also

wxTaskBarButton

Since

3.1.0

Public Member Functions

- wxAppProgressIndicator (wxWindow ∗parent=NULL, int maxValue=100)

  Constructs the wxAppProgressIndicator.
- virtual ∼wxAppProgressIndicator ()

  Destructor, stops displaying progress and returns the indicator to its normal state.
- bool IsAvailable () const

  Check if the application progress display is available.
- void SetValue (int value)

  Set the progress value in taskbar button of parent window.
- void SetRange (int range)

  Set the progress range in taskbar button of parent window.

- bool Pulse ()

  Makes the progress bar run in indeterminate mode.

21.26.2 Constructor & Destructor Documentation

wxAppProgressIndicator::wxAppProgressIndicator (wxWindow * *parent* = NULL, int *maxValue* = 100)

Constructs the wxAppProgressIndicator.

Parameters

parent	The parent window of wxAppProgressIndicator. Note that the window should has taskbar button showing. If parent is NULL, the progress will reflect on the taskbar buttons of all the top level windows.
maxValue	Integer range (maximum value) of the progress indicator.

virtual wxAppProgressIndicator::~wxAppProgressIndicator ( ) [virtual]

Destructor, stops displaying progress and returns the indicator to its normal state.

21.26.3 Member Function Documentation

bool wxAppProgressIndicator::IsAvailable ( ) const

Check if the application progress display is available.

Currently this only returns true when using wxMSW and running under Vista or later system, which provide task bar button API.

If this method returns false, no other methods of this class do anything, but they may still be called without any ill effects.

bool wxAppProgressIndicator::Pulse ( )

Makes the progress bar run in indeterminate mode.

void wxAppProgressIndicator::SetRange (int *range*)

Set the progress range in taskbar button of parent window.

void wxAppProgressIndicator::SetValue (int *value*)

Set the progress value in taskbar button of parent window.

Parameters

value	The new value of the progress meter. It should be less than or equal to the range.

21.27 wxAppTraits Class Reference

`#include <wx/apptrait.h>`

21.27.1 Detailed Description

The wxAppTraits class defines various configurable aspects of a wxApp.

You can access it using wxApp::GetTraits() function and you can create your own wxAppTraits overriding the wxApp::CreateTraits() function.

Note that wxAppTraits is an abstract class since it contains many pure virtual functions. In fact, by default, wxWidgets creates a `wxConsoleAppTraits` object for console applications (i.e. those applications linked against wxBase library only - see the Library List page) and a `wxGUIAppTraits` object for GUI applications. Both these classes are derived by wxAppTraits and represent concrete implementation of the wxAppTraits interface.

Library: wxBase

Category: Application and System configuration

See also

wxApp Overview, wxApp

Public Member Functions

- virtual wxConfigBase * CreateConfig ()

  Called by wxWidgets to create the default configuration object for the application.
- virtual wxEventLoopBase * CreateEventLoop ()=0

  Used by wxWidgets to create the main event loop used by wxApp::OnRun().
- virtual wxFontMapper * CreateFontMapper ()=0

  Creates the global font mapper object used for encodings/charset mapping.
- virtual wxLog * CreateLogTarget ()=0

  Creates a wxLog class for the application to use for logging errors.
- virtual wxMessageOutput * CreateMessageOutput ()=0

  Creates the global object used for printing out messages.
- virtual wxRendererNative * CreateRenderer ()=0

  Returns the renderer to use for drawing the generic controls (return value may be NULL in which case the default renderer for the current platform is used); this is used in GUI mode only and always returns NULL in console.
- virtual wxString GetDesktopEnvironment () const =0

  This method returns the name of the desktop environment currently running in a Unix desktop.
- virtual wxStandardPaths & GetStandardPaths ()

  Returns the wxStandardPaths object for the application.
- virtual wxPortId GetToolkitVersion (int *major=NULL, int *minor=NULL) const =0

  Returns the wxWidgets port ID used by the running program and eventually fills the given pointers with the values of the major and minor digits of the native toolkit currently used.
- virtual bool HasStderr ()=0

  Returns true if `fprintf(stderr)` goes somewhere, false otherwise.
- virtual bool IsUsingUniversalWidgets () const =0

  Returns true if the library was built as wxUniversal.
- virtual bool ShowAssertDialog (const wxString &msg)=0

  Shows the assert dialog with the specified message in GUI mode or just prints the string to stderr in console mode.

21.27.2 Member Function Documentation

virtual wxConfigBase* wxAppTraits::CreateConfig ( ) [virtual]

Called by wxWidgets to create the default configuration object for the application.

The default version creates a registry-based wxRegConfig class under MSW and wxFileConfig under all other platforms.

The wxApp::GetAppName and wxApp::GetVendorName methods are used to determine the registry key or file name.

virtual wxEventLoopBase* wxAppTraits::CreateEventLoop ( ) [pure virtual]

Used by wxWidgets to create the main event loop used by wxApp::OnRun().

The default implementation of this method in wxGUIAppTraits returns the usual platform-specific GUI event loop. The version in wxConsoleAppTraits returns a console-specific event loop which can be used to handle timer and socket events in console programs under Unix and MSW or NULL under the other platforms where console event loops are not supported yet.

virtual wxFontMapper* wxAppTraits::CreateFontMapper ( ) [pure virtual]

Creates the global font mapper object used for encodings/charset mapping.

virtual wxLog* wxAppTraits::CreateLogTarget ( ) [pure virtual]

Creates a wxLog class for the application to use for logging errors.

The default implementation returns a new wxLogGui class.

See also

wxLog

virtual wxMessageOutput* wxAppTraits::CreateMessageOutput ( ) [pure virtual]

Creates the global object used for printing out messages.

virtual wxRendererNative* wxAppTraits::CreateRenderer ( ) [pure virtual]

Returns the renderer to use for drawing the generic controls (return value may be NULL in which case the default renderer for the current platform is used); this is used in GUI mode only and always returns NULL in console.

Note

the returned pointer needs to be deleted by the caller.

virtual wxString wxAppTraits::GetDesktopEnvironment ( ) const [pure virtual]

This method returns the name of the desktop environment currently running in a Unix desktop.

Currently only "KDE" or "GNOME" are supported and the code uses the X11 session protocol vendor name to figure out, which desktop environment is running. The method returns an empty string otherwise and on all other platforms.

virtual wxStandardPaths& wxAppTraits::GetStandardPaths ( ) [virtual]

Returns the wxStandardPaths object for the application.

It's normally the same for wxBase and wxGUI except in the case of wxMac and wxCocoa.

Note

The returned reference is to a `wxStandardPathsBase` class but you can consider it to be equivalent to wxStandardPaths (which is documented).

virtual wxPortId wxAppTraits::GetToolkitVersion (int * *major* = NULL, int * *minor* = NULL) const [pure virtual]

Returns the wxWidgets port ID used by the running program and eventually fills the given pointers with the values of the major and minor digits of the native toolkit currently used.

The version numbers returned are thus detected at run-time and not compile-time (except when this is not possible e.g. wxMotif).

E.g. if your program is using wxGTK port this function will return wxPORT_GTK and put in given pointers the versions of the GTK library in use. See wxPlatformInfo for more details.

virtual bool wxAppTraits::HasStderr ( ) [pure virtual]

Returns true if `fprintf(stderr)` goes somewhere, false otherwise.

virtual bool wxAppTraits::IsUsingUniversalWidgets ( ) const [pure virtual]

Returns true if the library was built as wxUniversal.

Always returns false for wxBase-only apps.

virtual bool wxAppTraits::ShowAssertDialog (const wxString & *msg*) [pure virtual]

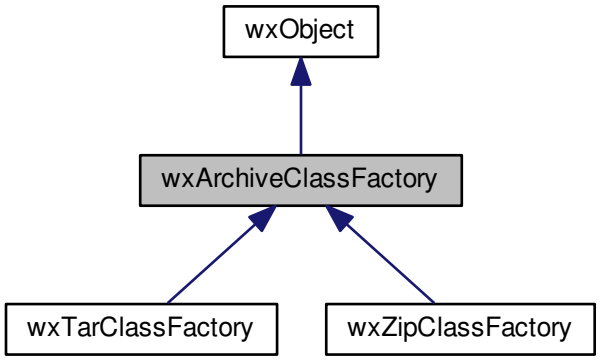
Shows the assert dialog with the specified message in GUI mode or just prints the string to stderr in console mode.

Returns true to suppress subsequent asserts, false to continue as before.

21.28 wxArchiveClassFactory Class Reference

`#include <wx/archive.h>`

Inheritance diagram for wxArchiveClassFactory:



21.28.1 Detailed Description

Allows the creation of streams to handle archive formats such as zip and tar.

For example, given a filename you can search for a factory that will handle it and create a stream to read it:

```
factory = wxArchiveClassFactory::Find(filename,
      wxSTREAM_FILEEXT);
if (factory)
    stream = factory->NewStream(new wxFFileInputStream(filename));
```

wxArchiveClassFactory::Find can also search for a factory by MIME type or wxFileSystem protocol.

The available factories can be enumerated using wxArchiveClassFactory::GetFirst() and wxArchiveClassFactory::GetNext().

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, Generic Archive Programming, wxArchiveEntry, wxArchiveInputStream, wxArchiveOutputStream, wxFilterClassFactory

- const wxArchiveClassFactory * GetNext () const

  GetFirst and GetNext can be used to enumerate the available factories.
- static const wxArchiveClassFactory * GetFirst ()

  GetFirst and GetNext can be used to enumerate the available factories.

Public Member Functions

- bool CanHandle (const wxString &protocol, wxStreamProtocolType type=wxSTREAM_PROTOCOL) const

Returns true if this factory can handle the given protocol, MIME type or file extension.

- wxMBConv & GetConv () const

  Returns the wxMBConv object that the created streams will use when translating meta-data.

- void SetConv (wxMBConv &conv)

  Sets the wxMBConv object that the created streams will use when translating meta-data.

- virtual wxString GetInternalName (const wxString &name, wxPathFormat format=wxPATH_NATIVE) const =0

  Calls the static GetInternalName() function for the archive entry type, for example wxZipEntry::GetInternalName.

- wxString GetProtocol () const

  Returns the wxFileSystem protocol supported by this factory.

- virtual const wxChar ** GetProtocols (wxStreamProtocolType type=wxSTREAM_PROTOCOL) const =0

  Returns the protocols, MIME types or file extensions supported by this factory, as an array of null terminated strings.

- wxArchiveEntry * NewEntry () const

  Create a new wxArchiveEntry object of the appropriate type.

- void PushFront ()

  Adds this class factory to the list returned by GetFirst() or GetNext().

- void Remove ()

  Removes this class factory from the list returned by GetFirst() and GetNext().

- wxArchiveInputStream * NewStream (wxInputStream &stream) const

  Create a new input or output stream to read or write an archive.

- wxArchiveOutputStream * NewStream (wxOutputStream &stream) const

  Create a new input or output stream to read or write an archive.

- wxArchiveInputStream * NewStream (wxInputStream *stream) const

  Create a new input or output stream to read or write an archive.

- wxArchiveOutputStream * NewStream (wxOutputStream *stream) const

  Create a new input or output stream to read or write an archive.

Static Public Member Functions

- static const wxArchiveClassFactory * Find (const wxString &protocol, wxStreamProtocolType type=wxSTREAM_PROTOCOL)

  A static member that finds a factory that can handle a given protocol, MIME type or file extension.

Additional Inherited Members

21.28.2 Member Function Documentation

bool wxArchiveClassFactory::CanHandle (const wxString & *protocol*, wxStreamProtocolType *type* = wxSTREAM_PROTOCOL) const

Returns true if this factory can handle the given protocol, MIME type or file extension.

When using wxSTREAM_FILEEXT for the second parameter, the first parameter can be a complete filename rather than just an extension.

static const wxArchiveClassFactory* wxArchiveClassFactory::Find (const wxString & *protocol,* wxStreamProtocolType *type* = wxSTREAM_PROTOCOL) [static]

A static member that finds a factory that can handle a given protocol, MIME type or file extension.

Returns a pointer to the class factory if found, or NULL otherwise. It does not give away ownership of the factory.

When using wxSTREAM_FILEEXT for the second parameter, the first parameter can be a complete filename rather than just an extension.

wxMBConv& wxArchiveClassFactory::GetConv ( ) const

Returns the wxMBConv object that the created streams will use when translating meta-data.

The initial default, set by the constructor, is wxConvLocal.

static const wxArchiveClassFactory* wxArchiveClassFactory::GetFirst ( ) [static]

GetFirst and GetNext can be used to enumerate the available factories.

For example, to list them:

```
wxString list;
const wxArchiveClassFactory *factory =
      wxArchiveClassFactory::GetFirst();

while (factory) {
    list << factory->GetProtocol() << wxT("\n");
    factory = factory->GetNext();
}
```

GetFirst() and GetNext() return a pointer to a factory or NULL if no more are available. They do not give away ownership of the factory.

virtual wxString wxArchiveClassFactory::GetInternalName (const wxString & *name,* wxPathFormat *format* = wxPATH_NATIVE) const [pure virtual]

Calls the static GetInternalName() function for the archive entry type, for example wxZipEntry::GetInternalName.

const wxArchiveClassFactory* wxArchiveClassFactory::GetNext ( ) const

GetFirst and GetNext can be used to enumerate the available factories.

For example, to list them:

```
wxString list;
const wxArchiveClassFactory *factory =
      wxArchiveClassFactory::GetFirst();

while (factory) {
    list << factory->GetProtocol() << wxT("\n");
    factory = factory->GetNext();
}
```

GetFirst() and GetNext() return a pointer to a factory or NULL if no more are available. They do not give away ownership of the factory.

wxString wxArchiveClassFactory::GetProtocol ( ) const

Returns the wxFileSystem protocol supported by this factory.

Equivalent to

```
wxString(*GetProtocols())
```

.

virtual const wxChar** wxArchiveClassFactory::GetProtocols (wxStreamProtocolType *type* = wxSTREAM_PROTOCOL) const [pure virtual]

Returns the protocols, MIME types or file extensions supported by this factory, as an array of null terminated strings.

It does not give away ownership of the array or strings. For example, to list the file extensions a factory supports:

```
wxString list;
const wxChar *const *p;

for (p = factory->GetProtocols(wxSTREAM_FILEEXT); *p; p++)
    list << *p << wxT("\n");
```

wxArchiveEntry* wxArchiveClassFactory::NewEntry ( ) const

Create a new wxArchiveEntry object of the appropriate type.

wxArchiveInputStream* wxArchiveClassFactory::NewStream (wxInputStream & *stream*) const

Create a new input or output stream to read or write an archive.

If the parent stream is passed as a pointer then the new archive stream takes ownership of it. If it is passed by reference then it does not.

wxArchiveOutputStream* wxArchiveClassFactory::NewStream (wxOutputStream & *stream*) const

Create a new input or output stream to read or write an archive.

If the parent stream is passed as a pointer then the new archive stream takes ownership of it. If it is passed by reference then it does not.

wxArchiveInputStream* wxArchiveClassFactory::NewStream (wxInputStream * *stream*) const

Create a new input or output stream to read or write an archive.

If the parent stream is passed as a pointer then the new archive stream takes ownership of it. If it is passed by reference then it does not.

wxArchiveOutputStream* wxArchiveClassFactory::NewStream (wxOutputStream * *stream*) const

Create a new input or output stream to read or write an archive.

If the parent stream is passed as a pointer then the new archive stream takes ownership of it. If it is passed by reference then it does not.

void wxArchiveClassFactory::PushFront ( )

Adds this class factory to the list returned by GetFirst() or GetNext().

It is not necessary to do this to use the archive streams. It is usually used when implementing streams, typically the implementation will add a static instance of its factory class.

It can also be used to change the order of a factory already in the list, bringing it to the front. This isn't a thread safe operation so can't be done when other threads are running that will be using the list. The list does not take ownership of the factory.

void wxArchiveClassFactory::Remove ( )

Removes this class factory from the list returned by GetFirst() and GetNext().

Removing from the list isn't a thread safe operation so can't be done when other threads are running that will be using the list. The list does not own the factories, so removing a factory does not delete it.

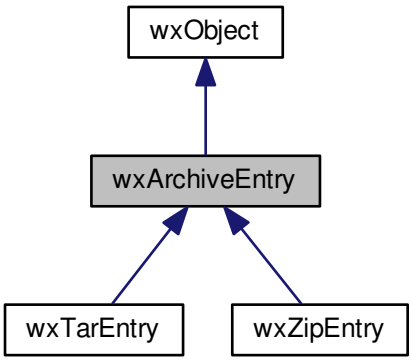
void wxArchiveClassFactory::SetConv (wxMBConv & *conv*)

Sets the wxMBConv object that the created streams will use when translating meta-data.

21.29 wxArchiveEntry Class Reference

`#include <wx/archive.h>`

Inheritance diagram for wxArchiveEntry:



21.29.1 Detailed Description

This is an abstract base class which serves as a common interface to archive entry classes such as wxZipEntry.

These hold the meta-data (filename, timestamp, etc.), for entries in archive files such as zips and tars.

21.29.2 About non-seekable streams

This information applies only when reading archives from non-seekable streams. When the stream is seekable GetNextEntry() returns a fully populated wxArchiveEntry. See Archives on Non-Seekable Streams for more information.

For generic programming, when the worst case must be assumed, you can rely on all the fields of wxArchiveEntry being fully populated when wxArchiveInputStream::GetNextEntry() returns, with the following exceptions:

- GetSize(): guaranteed to be available after the entry has been read to Eof(), or CloseEntry() has been called;
- IsReadOnly(): guaranteed to be available after the end of the archive has been reached, i.e. after GetNext↩ Entry() returns NULL and Eof() is true.

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, Generic Archive Programming, wxArchiveInputStream, wxArchiveOutputStream, wx↩ ArchiveNotifier

Public Member Functions

- wxArchiveEntry * Clone () const

  Returns a copy of this entry object.
- virtual wxDateTime GetDateTime () const =0

  Gets the entry's timestamp.
- virtual void SetDateTime (const wxDateTime &dt)=0

  Sets the entry's timestamp.
- virtual wxString GetName (wxPathFormat format=wxPATH_NATIVE) const =0

  Returns the entry's name, by default in the native format.
- virtual void SetName (const wxString &name, wxPathFormat format=wxPATH_NATIVE)=0

  Sets the entry's name.
- virtual wxFileOffset GetSize () const =0

  Returns the size of the entry's data in bytes.
- virtual void SetSize (wxFileOffset size)=0

  Sets the size of the entry's data in bytes.
- virtual wxPathFormat GetInternalFormat () const =0

  Returns the path format used internally within the archive to store filenames.
- virtual wxString GetInternalName () const =0

  Returns the entry's filename in the internal format used within the archive.
- virtual wxFileOffset GetOffset () const =0

  Returns a numeric value unique to the entry within the archive.
- virtual bool IsDir () const =0

  Returns true if this is a directory entry.
- virtual void SetIsDir (bool isDir=true)=0

  Marks this entry as a directory if isDir is true.
- virtual bool IsReadOnly () const =0

  Returns true if the entry is a read-only file.
- virtual void SetIsReadOnly (bool isReadOnly=true)=0

  Sets this entry as a read-only file.
- void SetNotifier (wxArchiveNotifier &notifier)

  Sets the notifier (see wxArchiveNotifier) for this entry.
- virtual void UnsetNotifier ()

  Unsets the notifier eventually attached to this entry.

Additional Inherited Members

21.29.3 Member Function Documentation

wxArchiveEntry* wxArchiveEntry::Clone ( ) const

Returns a copy of this entry object.

virtual wxDateTime wxArchiveEntry::GetDateTime ( ) const [pure virtual]

Gets the entry's timestamp.

virtual wxPathFormat wxArchiveEntry::GetInternalFormat ( ) const [pure virtual]

Returns the path format used internally within the archive to store filenames.

virtual wxString wxArchiveEntry::GetInternalName ( ) const [pure virtual]

Returns the entry's filename in the internal format used within the archive.

The name can include directory components, i.e. it can be a full path.

The names of directory entries are returned without any trailing path separator. This gives a canonical name that can be used in comparisons.

See also

Looking Up an Archive Entry by Name

Implemented in wxTarEntry, and wxZipEntry.

virtual wxString wxArchiveEntry::GetName (wxPathFormat *format* = wxPATH_NATIVE) const [pure virtual]

Returns the entry's name, by default in the native format.

The name can include directory components, i.e. it can be a full path.

If this is a directory entry, (i.e. if IsDir() is true) then the returned string is the name with a trailing path separator.

virtual wxFileOffset wxArchiveEntry::GetOffset ( ) const [pure virtual]

Returns a numeric value unique to the entry within the archive.

virtual wxFileOffset wxArchiveEntry::GetSize ( ) const [pure virtual]

Returns the size of the entry's data in bytes.

Implemented in wxTarEntry.

virtual bool wxArchiveEntry::IsDir ( ) const [pure virtual]

Returns true if this is a directory entry.

Directory entries are entries with no data, which are used to store the meta-data of directories. They also make it possible for completely empty directories to be stored.

Note

The names of entries within an archive can be complete paths, and unarchivers typically create whatever directories are necessary as they restore files, even if the archive contains no explicit directory entries.

virtual bool wxArchiveEntry::IsReadOnly ( ) const `[pure virtual]`

Returns true if the entry is a read-only file.

virtual void wxArchiveEntry::SetDateTime (const wxDateTime & *dt*) `[pure virtual]`

Sets the entry's timestamp.

virtual void wxArchiveEntry::SetIsDir (bool *isDir* = `true`) `[pure virtual]`

Marks this entry as a directory if *isDir* is true.

See IsDir() for more info.

virtual void wxArchiveEntry::SetIsReadOnly (bool *isReadOnly* = `true`) `[pure virtual]`

Sets this entry as a read-only file.

virtual void wxArchiveEntry::SetName (const wxString & *name*, wxPathFormat *format* = wxPATH_NATIVE) `[pure virtual]`

Sets the entry's name.

Setting a name with a trailing path separator sets IsDir().

See also

GetName()

void wxArchiveEntry::SetNotifier (wxArchiveNotifier & *notifier*)

Sets the notifier (see wxArchiveNotifier) for this entry.

Whenever the wxArchiveInputStream updates this entry, it will then invoke the associated notifier's wxArchiveNotifier::OnEntryUpdated method.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an archive in a pipeline (i.e. between non-seekable streams).

virtual void wxArchiveEntry::SetSize (wxFileOffset *size*) `[pure virtual]`

Sets the size of the entry's data in bytes.

Implemented in wxTarEntry.

virtual void wxArchiveEntry::UnsetNotifier ( ) `[virtual]`

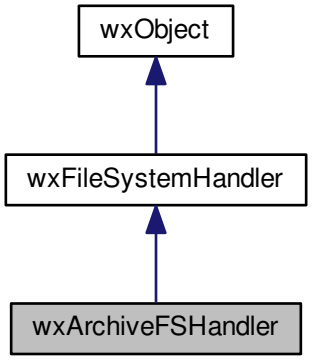
Unsets the notifier eventually attached to this entry.

Reimplemented in wxZipEntry.

21.30 wxArchiveFSHandler Class Reference

`#include <wx/fs_arc.h>`

Inheritance diagram for wxArchiveFSHandler:



21.30.1 Detailed Description

A file system handler for accessing files inside of archives.

Public Member Functions

- wxArchiveFSHandler ()
- virtual ~wxArchiveFSHandler ()
- void Cleanup ()

Additional Inherited Members

21.30.2 Constructor & Destructor Documentation

wxArchiveFSHandler::wxArchiveFSHandler ( )

virtual wxArchiveFSHandler::~wxArchiveFSHandler ( ) `[virtual]`

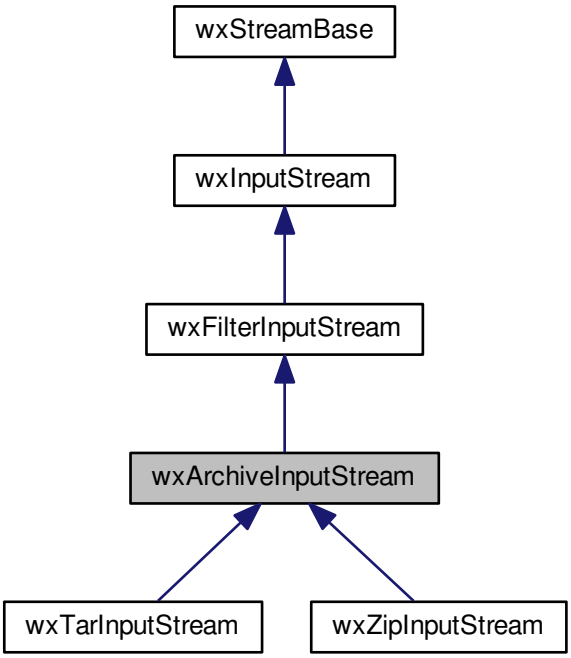
21.30.3 Member Function Documentation

void wxArchiveFSHandler::Cleanup ( )

21.31 wxArchiveInputStream Class Reference

`#include <wx/archive.h>`

Inheritance diagram for wxArchiveInputStream:



21.31.1 Detailed Description

This is an abstract base class which serves as a common interface to archive input streams such as wxZipInputStream.

wxArchiveInputStream::GetNextEntry returns an wxArchiveEntry object containing the meta-data for the next entry in the archive (and gives away ownership).

Reading from the wxArchiveInputStream then returns the entry's data. Eof() becomes true after an attempt has been made to read past the end of the entry's data.

When there are no more entries, GetNextEntry() returns NULL and sets Eof().

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, wxArchiveEntry, wxArchiveOutputStream

Public Member Functions

- virtual bool CloseEntry ()=0

  Closes the current entry.

- wxArchiveEntry * GetNextEntry ()

  Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a wxArchiveEntry object, giving away ownership.
- virtual bool OpenEntry (wxArchiveEntry &entry)=0

  Closes the current entry if one is open, then opens the entry specified by the wxArchiveEntry object.

Additional Inherited Members

21.31.2 Member Function Documentation

virtual bool wxArchiveInputStream::CloseEntry ( ) [pure virtual]

Closes the current entry.

On a non-seekable stream reads to the end of the current entry first.

Implemented in wxZipInputStream, and wxTarInputStream.

wxArchiveEntry* wxArchiveInputStream::GetNextEntry ( )

Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a wxArchiveEntry object, giving away ownership.

Reading this wxArchiveInputStream then returns the entry's data.

virtual bool wxArchiveInputStream::OpenEntry (wxArchiveEntry & *entry*) [pure virtual]

Closes the current entry if one is open, then opens the entry specified by the wxArchiveEntry object.

entry must be from the same archive file that this wxArchiveInputStream is reading, and it must be reading it from a seekable stream.

21.32 wxArchiveIterator Class Reference

`#include <wx/archive.h>`

21.32.1 Detailed Description

An input iterator template class that can be used to transfer an archive's catalogue to a container.

It is only available if wxUSE_STL is set to 1 in setup.h, and the uses for it outlined below require a compiler which supports member templates.

```
template<class Arc, class T = typename Arc::entry_type*>
class wxArchiveIterator
{
    // this constructor creates an 'end of sequence' object
    wxArchiveIterator();

    // template parameter 'Arc' should be the type of an archive input stream
    wxArchiveIterator(Arc& arc) {
        // ...
    }
};
```

The first template parameter should be the type of archive input stream (e.g. wxArchiveInputStream) and the second can either be a pointer to an entry (e.g. wxArchiveEntry*), or a string/pointer pair (e.g. std::pair<wx↩String,wxArchiveEntry*>).

The wx/archive.h header defines the following typedefs:

```
typedef wxArchiveIterator<wxArchiveInputStream> wxArchiveIter;

typedef wxArchiveIterator<wxArchiveInputStream,
        std::pair<wxString, wxArchiveEntry*> > wxArchivePairIter;
```

The header for any implementation of this interface should define similar typedefs for its types, for example in wx/zipstrm.h there is:

```
typedef wxArchiveIterator<wxZipInputStream> wxZipIter;

typedef wxArchiveIterator<wxZipInputStream,
         std::pair<wxString, wxZipEntry*> > wxZipPairIter;
```

Transferring the catalogue of an archive *arc* to a vector *cat*, can then be done something like this:

```
std::vector<wxArchiveEntry*> cat((wxArchiveIter)arc, wxArchiveIter());
```

When the iterator is dereferenced, it gives away ownership of an entry object. So in the above example, when you have finished with *cat* you must delete the pointers it contains.

If you have smart pointers with normal copy semantics (i.e. not auto_ptr or wxScopedPtr), then you can create an iterator which uses them instead.

For example, with a smart pointer class for zip entries *ZipEntryPtr:*

```
typedef std::vector<ZipEntryPtr> ZipCatalog;
typedef wxArchiveIterator<wxZipInputStream, ZipEntryPtr>
      ZipIter;
ZipCatalog cat((ZipIter)zip, ZipIter());
```

Iterators that return std::pair objects can be used to populate a std::multimap, to allow entries to be looked up by name. The string is initialised using the wxArchiveEntry object's wxArchiveEntry::GetInternalName function.

```
typedef std::multimap<wxString, wxZipEntry*> ZipCatalog;
ZipCatalog cat((wxZipPairIter)zip, wxZipPairIter());
```

Note that this iterator also gives away ownership of an entry object each time it is dereferenced. So in the above example, when you have finished with *cat* you must delete the pointers it contains.

Or if you have them, a pair containing a smart pointer can be used (again *ZipEntryPtr*), no worries about ownership:

```
typedef std::multimap<wxString, ZipEntryPtr> ZipCatalog;
typedef wxArchiveIterator<wxZipInputStream,
            std::pair<wxString, ZipEntryPtr> > ZipPairIter;
ZipCatalog cat((ZipPairIter)zip, ZipPairIter());
```

Library: wxBase

Category: Archive support, Streams

See also

wxArchiveEntry, wxArchiveInputStream, wxArchiveOutputStream

Public Member Functions

- wxArchiveIterator ()

  Default constructor.
- wxArchiveIterator (Arc &arc)

  Construct the iterator that returns all the entries in the archive input stream arc.

- const T operator∗ () const

  Returns an entry object from the archive input stream, giving away ownership.

- wxArchiveIterator operator++ ()

  Position the input iterator at the next entry in the archive input stream.

- wxArchiveIterator operator++ (int)

  Position the input iterator at the next entry in the archive input stream.

21.32.2 Constructor & Destructor Documentation

wxArchiveIterator::wxArchiveIterator ( )

Default constructor.

wxArchiveIterator::wxArchiveIterator (Arc & *arc*)

Construct the iterator that returns all the entries in the archive input stream *arc*.

21.32.3 Member Function Documentation

const T wxArchiveIterator::operator∗ ( ) const

Returns an entry object from the archive input stream, giving away ownership.

wxArchiveIterator wxArchiveIterator::operator++ ( )

Position the input iterator at the next entry in the archive input stream.

wxArchiveIterator wxArchiveIterator::operator++ (int)

Position the input iterator at the next entry in the archive input stream.

21.33 wxArchiveNotifier Class Reference

`#include <wx/archive.h>`

21.33.1 Detailed Description

If you need to know when a wxArchiveInputStream updates a wxArchiveEntry object, you can create a notifier by deriving from this abstract base class, overriding wxArchiveNotifier::OnEntryUpdated.

An instance of your notifier class can then be assigned to the wxArchiveEntry object using wxArchiveEntry::Set↩ Notifier. Your OnEntryUpdated() method will then be invoked whenever the input stream updates the entry.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an archive in a pipeline (i.e. between non-seekable streams). See Archives on Non-Seekable Streams.

Library: wxBase

Category: Archive support, Streams

See also

Archives on Non-Seekable Streams, wxArchiveEntry, wxArchiveInputStream, wxArchiveOutputStream

Public Member Functions

- virtual void OnEntryUpdated (wxArchiveEntry &entry)=0

  This method must be overridden in your derived class.

21.33.2 Member Function Documentation

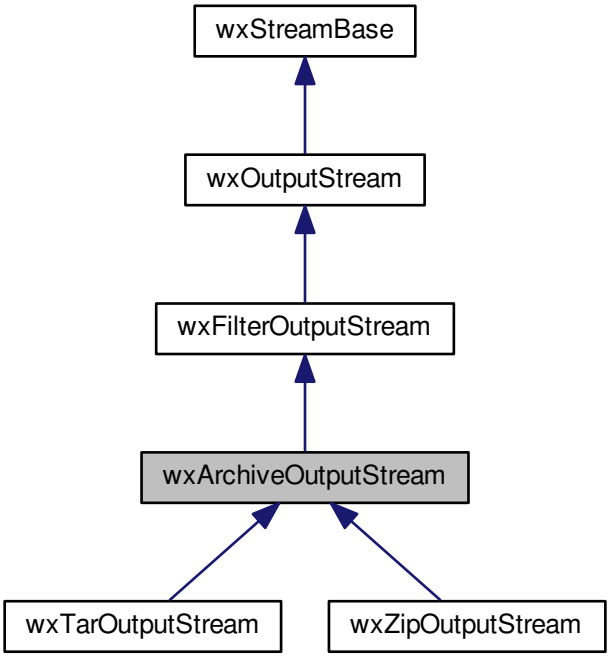
virtual void wxArchiveNotifier::OnEntryUpdated (wxArchiveEntry & *entry*) `[pure virtual]`

This method must be overridden in your derived class.

21.34 wxArchiveOutputStream Class Reference

`#include <wx/archive.h>`

Inheritance diagram for wxArchiveOutputStream:



21.34.1 Detailed Description

This is an abstract base class which serves as a common interface to archive output streams such as wxZip↩ OutputStream.

wxArchiveOutputStream::PutNextEntry is used to create a new entry in the output archive, then the entry's data is written to the wxArchiveOutputStream. Another call to PutNextEntry() closes the current entry and begins the next.

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, wxArchiveEntry, wxArchiveInputStream

Public Member Functions

- virtual ~wxArchiveOutputStream ()

  Calls Close() if it has not already been called.
- virtual bool Close ()

  Closes the archive, returning true if it was successfully written.
- virtual bool CloseEntry ()=0

  Close the current entry.
- virtual bool CopyArchiveMetaData (wxArchiveInputStream &stream)=0

  Some archive formats have additional meta-data that applies to the archive as a whole.
- virtual bool CopyEntry (wxArchiveEntry *entry, wxArchiveInputStream &stream)=0

  Takes ownership of entry and uses it to create a new entry in the archive.
- virtual bool PutNextDirEntry (const wxString &name, const wxDateTime &dt=wxDateTime::Now())=0

  Create a new directory entry (see wxArchiveEntry::IsDir) with the given name and timestamp.
- virtual bool PutNextEntry (wxArchiveEntry *entry)=0

  Takes ownership of entry and uses it to create a new entry in the archive.
- virtual bool PutNextEntry (const wxString &name, const wxDateTime &dt=wxDateTime::Now(), wxFileOffset size=wxInvalidOffset)=0

  Create a new entry with the given name, timestamp and size.

Additional Inherited Members

21.34.2 Constructor & Destructor Documentation

virtual wxArchiveOutputStream::~wxArchiveOutputStream ( ) [virtual]

Calls Close() if it has not already been called.

21.34.3 Member Function Documentation

virtual bool wxArchiveOutputStream::Close ( ) [virtual]

Closes the archive, returning true if it was successfully written.

Called by the destructor if not called explicitly.

See also

wxOutputStream::Close()

Reimplemented from wxOutputStream.

Reimplemented in wxZipOutputStream, and wxTarOutputStream.

virtual bool wxArchiveOutputStream::CloseEntry ( ) [pure virtual]

Close the current entry.

It is called implicitly whenever another new entry is created with CopyEntry() or PutNextEntry(), or when the archive is closed.

Implemented in wxZipOutputStream, and wxTarOutputStream.

virtual bool wxArchiveOutputStream::CopyArchiveMetaData (wxArchiveInputStream & *stream*) [pure virtual]

Some archive formats have additional meta-data that applies to the archive as a whole.

For example in the case of zip there is a comment, which is stored at the end of the zip file. CopyArchiveMetaData() can be used to transfer such information when writing a modified copy of an archive.

Since the position of the meta-data can vary between the various archive formats, it is best to call CopyArchive↩MetaData() before transferring the entries. The wxArchiveOutputStream will then hold on to the meta-data and write it at the correct point in the output file.

When the input archive is being read from a non-seekable stream, the meta-data may not be available when Copy↩ArchiveMetaData() is called, in which case the two streams set up a link and transfer the data when it becomes available.

virtual bool wxArchiveOutputStream::CopyEntry (wxArchiveEntry * *entry,* wxArchiveInputStream & *stream*) [pure virtual]

Takes ownership of *entry* and uses it to create a new entry in the archive.

entry is then opened in the input stream *stream* and its contents copied to this stream.

For archive types which compress entry data, CopyEntry() is likely to be much more efficient than transferring the data using Read() and Write() since it will copy them without decompressing and recompressing them.

entry must be from the same archive file that *stream* is accessing. For non-seekable streams, *entry* must also be the last thing read from *stream*.

virtual bool wxArchiveOutputStream::PutNextDirEntry (const wxString & *name,* const wxDateTime & *dt* = wxDateTime::Now()) [pure virtual]

Create a new directory entry (see wxArchiveEntry::IsDir) with the given name and timestamp.

PutNextEntry() can also be used to create directory entries, by supplying a name with a trailing path separator.

Implemented in wxZipOutputStream, and wxTarOutputStream.

virtual bool wxArchiveOutputStream::PutNextEntry (wxArchiveEntry * *entry*) [pure virtual]

Takes ownership of entry and uses it to create a new entry in the archive.

The entry's data can then be written by writing to this wxArchiveOutputStream.

virtual bool wxArchiveOutputStream::PutNextEntry (const wxString & *name,* const wxDateTime & *dt* = wxDateTime::Now (), wxFileOffset *size* = wxInvalidOffset) [pure virtual]

Create a new entry with the given name, timestamp and size.

The entry's data can then be written by writing to this wxArchiveOutputStream.

Implemented in wxZipOutputStream, and wxTarOutputStream.

21.35 wxArray< T > Class Template Reference

`#include <wx/dynarray.h>`

21.35.1 Detailed Description

template<typename T>class wxArray< T >

This section describes the so called *"dynamic arrays"*.

This is a C array-like type safe data structure i.e. the member access time is constant (and not linear according to the number of container elements as for linked lists). However, these arrays are dynamic in the sense that they will automatically allocate more memory if there is not enough of it for adding a new element. They also perform range checking on the index values but in debug mode only, so please be sure to compile your application in debug mode to use it (see Debugging for details). So, unlike the arrays in some other languages, attempt to access an element beyond the arrays bound doesn't automatically expand the array but provokes an assertion failure instead in debug build and does nothing (except possibly crashing your program) in the release build.

The array classes were designed to be reasonably efficient, both in terms of run-time speed and memory consumption and the executable size. The speed of array item access is, of course, constant (independent of the number of elements) making them much more efficient than linked lists (wxList). Adding items to the arrays is also implemented in more or less constant time, but the price is preallocating the memory in advance. In the "memory management" function section, you may find some useful hints about optimizing wxArray memory usage. As for executable size, all wxArray functions are inline, so they do not take *any* space at all.

wxWidgets has three different kinds of array. All of them derive from wxBaseArray class which works with untyped data and cannot be used directly. The standard macros WX_DEFINE_ARRAY(), WX_DEFINE_SORTED_ARRAY() and WX_DEFINE_OBJARRAY() are used to define a new class deriving from it. The classes declared will be called in this documentation wxArray, wxSortedArray and wxObjArray but you should keep in mind that no classes with such names actually exist, each time you use one of the WX_DEFINE_XXXARRAY() macros, you define a class with a new name. In fact, these names are "template" names and each usage of one of the macros mentioned above creates a template specialization for the given element type.

wxArray is suitable for storing integer types and pointers which it does not treat as objects in any way, i.e. the element pointed to by the pointer is not deleted when the element is removed from the array. It should be noted that all of wxArray's functions are inline, so it costs strictly nothing to define as many array types as you want (either in terms of the executable size or the speed) as long as at least one of them is defined and this is always the case because wxArrays are used by wxWidgets internally. This class has one serious limitation: it can only be used for storing integral types (bool, char, short, int, long and their unsigned variants) or pointers (of any kind). An attempt to use with objects of `sizeof()` greater than `sizeof(long)` will provoke a runtime assertion failure, however declaring a wxArray of floats will not (on the machines where "`sizeof(float) <= sizeof(long)`"), yet it will **not** work, please use wxObjArray for storing floats and doubles.

wxSortedArray is a wxArray variant which should be used when searching in the array is a frequently used operation. It requires you to define an additional function for comparing two elements of the array element type and always stores its items in the sorted order (according to this function). Thus, its Index() function execution time is "`O(log(N))`" instead of "`O(N)`" for the usual arrays but the Add() method is slower: it is "`O(log(N))`" instead of constant time (neglecting time spent in memory allocation routine). However, in a usual situation elements are added to an array much less often than searched inside it, so wxSortedArray may lead to huge performance improvements compared to wxArray. Finally, it should be noticed that, as wxArray, wxSortedArray can be only used

for storing integral types or pointers.

wxObjArray class treats its elements like "objects". It may delete them when they are removed from the array (invoking the correct destructor) and copies them using the objects copy constructor. In order to implement this behaviour the definition of the wxObjArray arrays is split in two parts: first, you should declare the new wxObjArray class using the WX_DECLARE_OBJARRAY() macro and then you must include the file defining the implementation of template type: <wx/arrimpl.cpp> and define the array class with the WX_DEFINE_OBJARRAY() macro from a point where the full (as opposed to 'forward') declaration of the array elements class is in scope. As it probably sounds very complicated here is an example:

```
#include <wx/dynarray.h>

// We must forward declare the array because it is used
// inside the class declaration.
class MyDirectory;
class MyFile;

// This defines two new types: ArrayOfDirectories and ArrayOfFiles which
// can be now used as shown below.
WX_DECLARE_OBJARRAY(MyDirectory, ArrayOfDirectories);
WX_DECLARE_OBJARRAY(MyFile,      ArrayOfFiles);

class MyDirectory
{
    // ...
    ArrayOfDirectories m_subdirectories; // All subdirectories
    ArrayOfFiles       m_files;          // All files in this directory
};

// ...

// Now that we have MyDirectory declaration in scope we may finish the
// definition of ArrayOfDirectories -- note that this expands into some C++
// code and so should only be compiled once (i.e., don't put this in the
// header, but into a source file or you will get linking errors)
#include <wx/arrimpl.cpp> // This is a magic incantation which must be done!
WX_DEFINE_OBJARRAY(ArrayOfDirectories);

// that's all!
```

It is not as elegant as writing this:

```
typedef std::vector<MyDirectory> ArrayOfDirectories;
```

But is not that complicated and allows the code to be compiled with any, however dumb, C++ compiler in the world.

Remember to include <wx/arrimpl.cpp> just before each WX_DEFINE_OBJARRAY() occurrence in your code, even if you have several in the same file.

Things are much simpler for wxArray and wxSortedArray however: it is enough just to write:

```
WX_DEFINE_ARRAY_INT(int, ArrayOfInts);
WX_DEFINE_SORTED_ARRAY_INT(int, ArrayOfSortedInts);
```

There is only one DEFINE macro and no need for separate DECLARE one. For the arrays of the primitive types, the macros WX_DEFINE_ARRAY_CHAR/SHORT/INT/SIZE_T/LONG/DOUBLE should be used depending on the sizeof of the values (notice that storing values of smaller type, e.g. shorts, in an array of larger one, e.g. ARRAY_INT, does not work on all architectures!).

21.35.2 Macros for Template Array Definition

To use an array you must first define the array class. This is done with the help of the macros in this section. The class of array elements must be (at least) forward declared for WX_DEFINE_ARRAY(), WX_DEFINE_SORTED_ARRAY() and WX_DECLARE_OBJARRAY() macros and must be fully declared before you use WX_DEFINE_OBJARRAY() macro.

- WX_DEFINE_ARRAY()
- WX_DEFINE_EXPORTED_ARRAY()

- WX_DEFINE_USER_EXPORTED_ARRAY()
- WX_DEFINE_SORTED_ARRAY()
- WX_DEFINE_SORTED_EXPORTED_ARRAY()
- WX_DEFINE_SORTED_USER_EXPORTED_ARRAY()
- WX_DECLARE_EXPORTED_OBJARRAY()
- WX_DECLARE_USER_EXPORTED_OBJARRAY()
- WX_DEFINE_OBJARRAY()
- WX_DEFINE_EXPORTED_OBJARRAY()
- WX_DEFINE_USER_EXPORTED_OBJARRAY()

To slightly complicate the matters even further, the operator "->" defined by default for the array iterators by these macros only makes sense if the array element type is not a pointer itself and, although it still works, this provokes warnings from some compilers and to avoid them you should use the _PTR versions of the macros above. For example, to define an array of pointers to double you should use:

```
WX_DEFINE_ARRAY_PTR(double *, MyArrayOfDoublePointers);
```

Note that the above macros are generally only useful for wxObject types. There are separate macros for declaring an array of a simple type, such as an int.

The following simple types are supported:

- int
- long
- size_t
- double

To create an array of a simple type, simply append the type you want in CAPS to the array definition.

For example, you'd use one of the following variants for an integer array:

- WX_DEFINE_ARRAY_INT()
- WX_DEFINE_EXPORTED_ARRAY_INT()
- WX_DEFINE_USER_EXPORTED_ARRAY_INT()
- WX_DEFINE_SORTED_ARRAY_INT()
- WX_DEFINE_SORTED_EXPORTED_ARRAY_INT()
- WX_DEFINE_SORTED_USER_EXPORTED_ARRAY_INT()

21.35.3 Predefined array types

wxWidgets defines the following dynamic array types:

- wxArrayShort
- wxArrayInt
- wxArrayDouble
- wxArrayLong
- wxArrayPtrVoid

To use them you don't need any macro; you just need to include dynarray.h.

Library: wxBase

Category: Containers

See also

Container Classes, wxList<T>, wxVector<T>

Public Member Functions

Constructors and Destructors

Array classes are 100% C++ objects and as such they have the appropriate copy constructors and assignment operators.

Copying wxArray just copies the elements but copying wxObjArray copies the arrays items. However, for memory-efficiency sake, neither of these classes has virtual destructor. It is not very important for wxArray which has trivial destructor anyhow, but it does mean that you should avoid deleting wxObjArray through a wx↩ BaseArray pointer (as you would never use wxBaseArray anyhow it shouldn't be a problem) and that you should not derive your own classes from the array classes.

- wxArray ()
  Default constructor.
- wxObjArray ()
  Default constructor initializes an empty array object.
- wxSortedArray (int(∗)(T first, T second) compareFunction)
  There is no default constructor for wxSortedArray classes - you must initialize it with a function to use for item comparison.
- wxArray (const wxArray &array)
  Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).
- wxSortedArray (const wxSortedArray &array)
  Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).
- wxObjArray (const wxObjArray &array)
  Performs a deep copy (i.e. the array element are copied too).
- wxArray & operator= (const wxArray &array)
  Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).
- wxSortedArray & operator= (const wxSortedArray &array)
  Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).
- wxObjArray & operator= (const wxObjArray &array)
  Performs a deep copy (i.e. the array element are copied too).
- ∼wxArray ()
  This destructor does not delete all the items owned by the array, you may use the WX_CLEAR_ARRAY() macro for this.
- ∼wxSortedArray ()
  This destructor does not delete all the items owned by the array, you may use the WX_CLEAR_ARRAY() macro for this.
- ∼wxObjArray ()
  This destructor deletes all the items owned by the array.

Memory Management

Automatic array memory management is quite trivial: the array starts by preallocating some minimal amount of memory (defined by `WX_ARRAY_DEFAULT_INITIAL_SIZE`) and when further new items exhaust already allocated memory it reallocates it adding 50% of the currently allocated amount, but no more than some maximal number which is defined by the `ARRAY_MAXSIZE_INCREMENT` constant.

Of course, this may lead to some memory being wasted (`ARRAY_MAXSIZE_INCREMENT` in the worst case, i.e. 4Kb in the current implementation), so the Shrink() function is provided to deallocate the extra memory. The Alloc() function can also be quite useful if you know in advance how many items you are going to put in the array and will prevent the array code from reallocating the memory more times than needed.

- void Alloc (size_t count)

  Preallocates memory for a given number of array elements.
- void Shrink ()

  Frees all memory unused by the array.

Number of Elements and Simple Item Access

Functions in this section return the total number of array elements and allow to retrieve them - possibly using just the C array indexing [] operator which does exactly the same as the Item() method.

- size_t GetCount () const

  Return the number of items in the array.
- bool IsEmpty () const

  Returns true if the array is empty, false otherwise.
- T & Item (size_t index) const

  Returns the item at the given position in the array.
- T & Last () const

  Returns the last element in the array, i.e. is the same as calling "Item(GetCount() - 1)".

Adding Items

- void Add (T item, size_t copies=1)

  Appends the given number of copies of the item to the array consisting of the elements of type `T`.
- size_t Add (T item)

  Appends the item to the array consisting of the elements of type `T`.
- void Add (T *item)

  Appends the item to the array consisting of the elements of type `T`.
- void Add (T &item, size_t copies=1)

  Appends the given number of copies of the item to the array consisting of the elements of type `T`.
- void AddAt (T item, size_t index)

  Inserts the given item into the array in the specified index position.
- void Insert (T item, size_t n, size_t copies=1)

  Insert the given number of copies of the item into the array before the existing item n - thus, Insert(something, 0u) will insert an item in such way that it will become the first array element.
- void Insert (T *item, size_t n)

  Insert the item into the array before the existing item n - thus, Insert(something, 0u) will insert an item in such way that it will become the first array element.
- void Insert (T &item, size_t n, size_t copies=1)

  Insert the given number of copies of the item into the array before the existing item n - thus, Insert(something, 0u) will insert an item in such way that it will become the first array element.
- void SetCount (size_t count, T defval=T(0))

  This function ensures that the number of array elements is at least count.

Removing Items

- void Clear ()

  This function does the same as Empty() and additionally frees the memory allocated to the array.
- T * Detach (size_t index)

  Removes the element from the array, but unlike Remove(), it doesn't delete it.
- void Empty ()

  Empties the array.
- void Remove (T item)

  Removes an element from the array by value: the first item of the array equal to item is removed, an assert failure will result from an attempt to remove an item which doesn't exist in the array.
- void RemoveAt (size_t index, size_t count=1)

Removes count elements starting at index from the array.

Searching and Sorting

- int Index (T &item, bool searchFromEnd=false) const
  This version of Index() is for wxArray and wxObjArray only.
- int Index (T &item) const
  This version of Index() is for wxSortedArray only.
- size_t IndexForInsert (T item) const
  Search for a place to insert item into the sorted array (binary search).
- void Sort (CMPFUNC< T > compareFunction)
  The notation "CMPFUNC<T>" should be read as if we had the following declaration:

21.35.4 Constructor & Destructor Documentation

template<typename T > wxArray< T >::wxArray ( )

Default constructor.

template<typename T > wxArray< T >::wxArray (const wxArray< T > & *array*)

Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).

template<typename T > wxArray< T >::~wxArray ( )

This destructor does not delete all the items owned by the array, you may use the WX_CLEAR_ARRAY() macro for this.

template<typename T > wxArray< T >::~wxSortedArray ( )

This destructor does not delete all the items owned by the array, you may use the WX_CLEAR_ARRAY() macro for this.

template<typename T > wxArray< T >::~wxObjArray ( )

This destructor deletes all the items owned by the array.

21.35.5 Member Function Documentation

template<typename T > void wxArray< T >::Add (T *item*, size_t *copies* = 1)

Appends the given number of *copies* of the *item* to the array consisting of the elements of type T.

This version is used with wxArray.

You may also use WX_APPEND_ARRAY() macro to append all elements of one array to another one but it is more efficient to use the *copies* parameter and modify the elements in place later if you plan to append a lot of items.

template<typename T > size_t wxArray< T >::Add (T *item*)

Appends the *item* to the array consisting of the elements of type T.

This version is used with wxSortedArray, returning the index where *item* is stored.

template<typename T > void wxArray< T >::Add (T * *item*)

Appends the *item* to the array consisting of the elements of type `T`.

This version is used with wxObjArray. The array will take ownership of the *item*, deleting it when the item is deleted from the array. Note that you cannot append more than one pointer as reusing it would lead to deleting it twice (or more) resulting in a crash.

You may also use WX_APPEND_ARRAY() macro to append all elements of one array to another one but it is more efficient to use the *copies* parameter and modify the elements in place later if you plan to append a lot of items.

template<typename T > void wxArray< T >::Add (T & *item*, size_t *copies* = `1`)

Appends the given number of *copies* of the *item* to the array consisting of the elements of type `T`.

This version is used with wxObjArray. The array will make a copy of the item and will not take ownership of the original item.

You may also use WX_APPEND_ARRAY() macro to append all elements of one array to another one but it is more efficient to use the *copies* parameter and modify the elements in place later if you plan to append a lot of items.

template<typename T > void wxArray< T >::AddAt (T *item*, size_t *index*)

Inserts the given *item* into the array in the specified *index* position.

Be aware that you will set out the order of the array if you give a wrong position.

This function is useful in conjunction with IndexForInsert() for a common operation of "insert only if not found".

template<typename T > void wxArray< T >::Alloc (size_t *count*)

Preallocates memory for a given number of array elements.

It is worth calling when the number of items which are going to be added to the array is known in advance because it will save unneeded memory reallocation. If the array already has enough memory for the given number of items, nothing happens. In any case, the existing contents of the array is not modified.

template<typename T > void wxArray< T >::Clear ( )

This function does the same as Empty() and additionally frees the memory allocated to the array.

template<typename T > T* wxArray< T >::Detach (size_t *index*)

Removes the element from the array, but unlike Remove(), it doesn't delete it.

The function returns the pointer to the removed element.

template<typename T > void wxArray< T >::Empty ( )

Empties the array.

For wxObjArray classes, this destroys all of the array elements. For wxArray and wxSortedArray this does nothing except marking the array of being empty - this function does not free the allocated memory, use Clear() for this.

template<typename T > size_t wxArray< T >::GetCount ( ) const

Return the number of items in the array.

template<typename T > int wxArray< T >::Index (T & *item*, bool *searchFromEnd* = `false`) const

This version of Index() is for wxArray and wxObjArray only.

Searches the element in the array, starting from either beginning or the end depending on the value of *searchFromEnd* parameter. `wxNOT_FOUND` is returned if the element is not found, otherwise the index of the element is returned.

Note

Even for wxObjArray classes, the operator "==" of the elements in the array is **not** used by this function. It searches exactly the given element in the array and so will only succeed if this element had been previously added to the array, but fail even if another, identical, element is in the array.

template<typename T > int wxArray< T >::Index (T & *item*) const

This version of Index() is for wxSortedArray only.

Searches for the element in the array, using binary search.

`wxNOT_FOUND` is returned if the element is not found, otherwise the index of the element is returned.

template<typename T > size_t wxArray< T >::IndexForInsert (T *item*) const

Search for a place to insert *item* into the sorted array (binary search).

The index returned is just before the first existing item that is greater or equal (according to the compare function) to the given *item*.

You have to do extra work to know if the *item* already exists in array.

This function is useful in conjunction with AddAt() for a common operation of "insert only if not found".

template<typename T > void wxArray< T >::Insert (T *item*, size_t *n*, size_t *copies* = `1`)

Insert the given number of *copies* of the *item* into the array before the existing item *n* - thus, *Insert(something, 0u)* will insert an item in such way that it will become the first array element.

wxSortedArray doesn't have this function because inserting in wrong place would break its sorted condition.

Please see Add() for an explanation of the differences between the overloaded versions of this function.

template<typename T > void wxArray< T >::Insert (T * *item*, size_t *n*)

Insert the *item* into the array before the existing item *n* - thus, *Insert(something, 0u)* will insert an item in such way that it will become the first array element.

wxSortedArray doesn't have this function because inserting in wrong place would break its sorted condition.

Please see Add() for an explanation of the differences between the overloaded versions of this function.

template<typename T > void wxArray< T >::Insert (T & *item*, size_t *n*, size_t *copies* = `1`)

Insert the given number of *copies* of the *item* into the array before the existing item *n* - thus, *Insert(something, 0u)* will insert an item in such way that it will become the first array element.

wxSortedArray doesn't have this function because inserting in wrong place would break its sorted condition.

Please see Add() for an explanation of the differences between the overloaded versions of this function.

template<typename T > bool wxArray< T >::IsEmpty ( ) const

Returns true if the array is empty, false otherwise.

template<typename T > T& wxArray< T >::Item (size_t *index*) const

Returns the item at the given position in the array.

If *index* is out of bounds, an assert failure is raised in the debug builds but nothing special is done in the release build.

The returned value is of type "reference to the array element type" for all of the array classes.

template<typename T > T& wxArray< T >::Last ( ) const

Returns the last element in the array, i.e. is the same as calling "Item(GetCount() - 1)".

An assert failure is raised in the debug mode if the array is empty.

The returned value is of type "reference to the array element type" for all of the array classes.

template<typename T > wxArray& wxArray< T >::operator= (const wxArray< T > & *array*)

Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).

template<typename T > wxSortedArray& wxArray< T >::operator= (const wxSortedArray & *array*)

Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).

template<typename T > wxObjArray& wxArray< T >::operator= (const wxObjArray & *array*)

Performs a deep copy (i.e. the array element are copied too).

template<typename T > void wxArray< T >::Remove (T *item*)

Removes an element from the array by value: the first item of the array equal to *item* is removed, an assert failure will result from an attempt to remove an item which doesn't exist in the array.

When an element is removed from wxObjArray it is deleted by the array - use Detach() if you don't want this to happen. On the other hand, when an object is removed from a wxArray nothing happens - you should delete it manually if required:

```
T *item = array[n];
array.Remove(item);
delete item;
```

See also WX_CLEAR_ARRAY() macro which deletes all elements of a wxArray (supposed to contain pointers).

Notice that for sorted arrays this method uses binary search to find the item so it doesn't necessarily remove the first matching item, but the first one found by the binary search.

See also

RemoveAt()

template<typename T > void wxArray< T >::RemoveAt (size_t *index*, size_t *count* = 1)

Removes *count* elements starting at *index* from the array.

When an element is removed from wxObjArray it is deleted by the array - use Detach() if you don't want this to happen. On the other hand, when an object is removed from a wxArray nothing happens - you should delete it manually if required:

```
T *item = array[n];
delete item;
array.RemoveAt(n);
```

See also WX_CLEAR_ARRAY() macro which deletes all elements of a wxArray (supposed to contain pointers).

template<typename T > void wxArray< T >::SetCount (size_t *count*, T *defval* = T(0))

This function ensures that the number of array elements is at least *count*.

If the array has already *count* or more items, nothing is done. Otherwise, *count* - GetCount() elements are added and initialized to the value *defval*.

See also

GetCount()

template<typename T > void wxArray< T >::Shrink ( )

Frees all memory unused by the array.

If the program knows that no new items will be added to the array it may call Shrink() to reduce its memory usage. However, if a new item is added to the array, some extra memory will be allocated again.

template<typename T > void wxArray< T >::Sort (CMPFUNC< T > *compareFunction*)

The notation "CMPFUNCT<T>" should be read as if we had the following declaration:

```
template int CMPFUNC(T *first, T *second);
```

Where *T* is the type of the array elements. I.e. it is a function returning *int* which is passed two arguments of type *T**.

Sorts the array using the specified compare function: this function should return a negative, zero or positive value according to whether the first element passed to it is less than, equal to or greater than the second one.

wxSortedArray doesn't have this function because it is always sorted.

template<typename T > wxArray< T >::wxObjArray ( )

Default constructor initializes an empty array object.

template<typename T > wxArray< T >::wxObjArray (const wxObjArray & *array*)

Performs a deep copy (i.e. the array element are copied too).

template<typename T > wxArray< T >::wxSortedArray (int(*)(T first, T second) *compareFunction*)

There is no default constructor for wxSortedArray classes - you must initialize it with a function to use for item comparison.

It is a function which is passed two arguments of type `T` where `T` is the array element type and which should return a negative, zero or positive value according to whether the first element passed to it is less than, equal to or greater than the second one.

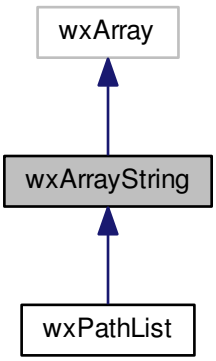
template<typename T > wxArray< T >::wxSortedArray (const wxSortedArray & *array*)

Performs a shallow array copy (i.e. doesn't copy the objects pointed to even if the source array contains the items of pointer type).

21.36 wxArrayString Class Reference

`#include <wx/arrstr.h>`

Inheritance diagram for wxArrayString:



21.36.1 Detailed Description

wxArrayString is an efficient container for storing wxString objects.

It has the same features as all wxArray classes, i.e. it dynamically expands when new items are added to it (so it is as easy to use as a linked list), but the access time to the elements is constant, instead of being linear in number of elements as in the case of linked lists. It is also very size efficient and doesn't take more space than a C array *wxString*[] type (wxArrayString uses its knowledge of internals of wxString class to achieve this).

This class is used in the same way as other dynamic arrays(), except that no WX_DEFINE_ARRAY declaration is needed for it. When a string is added or inserted in the array, a copy of the string is created, so the original string may be safely deleted (e.g. if it was a *wxChar* * pointer the memory it was using can be freed immediately after this). In general, there is no need to worry about string memory deallocation when using this class - it will always free the memory it uses itself.

The references returned by wxArrayString::Item, wxArrayString::Last or wxArrayString::operator[] are not constant, so the array elements may be modified in place like this:

```
array.Last().MakeUpper();
```

Note

none of the methods of wxArrayString is virtual including its destructor, so this class should not be used as a base class.

Although this is not true strictly speaking, this class may be considered as a specialization of wxArray class for the wxString member data: it is not implemented like this, but it does have all of the wxArray functions.

It also has the full set of `std::vector<wxString>` compatible methods, including nested `iterator` and `const_iterator` classes which should be used in the new code for forward compatibility with the future wxWidgets versions.

Library: wxBase

Category: Containers

See also

wxArray<T>, wxString, wxString Overview

Public Types

- typedef int(* CompareFunction)(const wxString &first, const wxString &second)

  The function type used with wxArrayString::Sort().

Public Member Functions

- wxArrayString ()

  Default constructor.
- wxArrayString (const wxArrayString &array)

  Copy constructor.
- wxArrayString (size_t sz, const wxString *arr)

  Constructor from a wxString array.
- ~wxArrayString ()

  Destructor frees memory occupied by the array strings.
- size_t Add (const wxString &str, size_t copies=1)

  Appends the given number of copies of the new item str to the array and returns the index of the first new item in the array.
- void Alloc (size_t nCount)

  Preallocates enough memory to store nCount items.
- void Clear ()

  Clears the array contents and frees memory.
- void Empty ()

  Empties the array: after a call to this function GetCount() will return 0.
- size_t GetCount () const

  Returns the number of items in the array.
- int Index (const wxString &sz, bool bCase=true, bool bFromEnd=false) const

  Search the element in the array, starting from the beginning if bFromEnd is false or from end otherwise.
- void Insert (wxString lItem, size_t nIndex, size_t copies=1)

Insert the given number of copies of the new element in the array before the position nIndex.

- bool IsEmpty () const

  Returns true if the array is empty, false otherwise.

- void Remove (const wxString &sz)

  Removes the first item matching this value.

- void RemoveAt (size_t nIndex, size_t count=1)

  Removes count items starting at position nIndex from the array.

- void Shrink ()

  Releases the extra memory allocated by the array.

- void Sort (bool reverseOrder=false)

  Sorts the array in alphabetical order or in reverse alphabetical order if reverseOrder is true.

- void Sort (CompareFunction compareFunction)

  Sorts the array using the specified compareFunction for item comparison.

- bool operator!= (const wxArrayString &array) const

  Compares 2 arrays respecting the case.

- wxArrayString & operator= (const wxArrayString &)

  Assignment operator.

- bool operator== (const wxArrayString &array) const

  Compares 2 arrays respecting the case.

- wxString & operator[] (size_t nIndex) const

  Return the array element at position nIndex.

- wxArrayString (size_t sz, const char **arr)

  Constructor from a C string array.

- wxArrayString (size_t sz, const wchar_t **arr)

  Constructor from a C string array.

- wxString & Item (size_t nIndex)

  Return the array element at position nIndex.

- const wxString & Item (size_t nIndex) const

  Return the array element at position nIndex.

- wxString & Last ()

  Returns the last element of the array.

- const wxString & Last () const

  Returns the last element of the array.

21.36.2 Member Typedef Documentation

typedef int(* wxArrayString::CompareFunction)(const wxString &first, const wxString &second)

The function type used with wxArrayString::Sort().

This function uses the same conventions as the standard `qsort()` comparison function, that is it should return a negative value if the first argument is less than the second one, a positive value if the first argument is greater than the second one and 0 if the arguments are equal.

Since

3.1.0

21.36.3 Constructor & Destructor Documentation

wxArrayString::wxArrayString ( )

Default constructor.

wxArrayString::wxArrayString (const wxArrayString & *array*)

Copy constructor.

wxArrayString::wxArrayString (size_t *sz,* const char ** *arr*)

Constructor from a C string array.

Pass a size *sz* and an array *arr*.

wxArrayString::wxArrayString (size_t *sz,* const wchar_t ** *arr*)

Constructor from a C string array.

Pass a size *sz* and an array *arr*.

wxArrayString::wxArrayString (size_t *sz,* const wxString * *arr*)

Constructor from a wxString array.

Pass a size *sz* and array *arr*.

wxArrayString::~wxArrayString ( )

Destructor frees memory occupied by the array strings.

For performance reasons it is not virtual, so this class should not be derived from.

21.36.4 Member Function Documentation

size_t wxArrayString::Add (const wxString & *str,* size_t *copies* = 1)

Appends the given number of *copies* of the new item *str* to the array and returns the index of the first new item in the array.

See also

Insert()

void wxArrayString::Alloc (size_t *nCount*)

Preallocates enough memory to store *nCount* items.

This function may be used to improve array class performance before adding a known number of items consecutively.

void wxArrayString::Clear ( )

Clears the array contents and frees memory.

See also

Empty()

void wxArrayString::Empty ( )

Empties the array: after a call to this function GetCount() will return 0.

However, this function does not free the memory used by the array and so should be used when the array is going to be reused for storing other strings. Otherwise, you should use Clear() to empty the array and free memory.

size_t wxArrayString::GetCount ( ) const

Returns the number of items in the array.

int wxArrayString::Index (const wxString & *sz*, bool *bCase* = `true`, bool *bFromEnd* = `false`) const

Search the element in the array, starting from the beginning if *bFromEnd* is false or from end otherwise.

If *bCase*, comparison is case sensitive (default), otherwise the case is ignored.

This function uses linear search for wxArrayString. Returns index of the first item matched or `wxNOT_FOUND` if there is no match.

void wxArrayString::Insert (wxString *lItem*, size_t *nIndex*, size_t *copies* = `1`)

Insert the given number of *copies* of the new element in the array before the position *nIndex*.

Thus, for example, to insert the string in the beginning of the array you would write:

```
Insert("foo", 0);
```

If *nIndex* is equal to GetCount() this function behaves as Add().

bool wxArrayString::IsEmpty ( ) const

Returns true if the array is empty, false otherwise.

This function returns the same result as GetCount() == 0 but is probably easier to read.

wxString& wxArrayString::Item (size_t *nIndex*)

Return the array element at position *nIndex*.

An assert failure will result from an attempt to access an element beyond the end of array in debug mode, but no check is done in release mode.

See also

operator[] for the operator version.

const wxString& wxArrayString::Item (size_t *nIndex*) const

Return the array element at position *nIndex*.

An assert failure will result from an attempt to access an element beyond the end of array in debug mode, but no check is done in release mode.

See also

operator[] for the operator version.

wxString& wxArrayString::Last ( )

Returns the last element of the array.

Attempt to access the last element of an empty array will result in assert failure in debug build, however no checks are done in release mode.

const wxString& wxArrayString::Last ( ) const

Returns the last element of the array.

Attempt to access the last element of an empty array will result in assert failure in debug build, however no checks are done in release mode.

bool wxArrayString::operator!= (const wxArrayString & *array*) const

Compares 2 arrays respecting the case.

Returns true if the arrays have different number of elements or if the elements don't match pairwise.

wxArrayString& wxArrayString::operator= (const wxArrayString &)

Assignment operator.

bool wxArrayString::operator== (const wxArrayString & *array*) const

Compares 2 arrays respecting the case.

Returns true only if the arrays have the same number of elements and the same strings in the same order.

wxString& wxArrayString::operator[] (size_t *nIndex*) const

Return the array element at position *nIndex*.

An assert failure will result from an attempt to access an element beyond the end of array in debug mode, but no check is done in release mode.

This is the operator version of the Item() method.

void wxArrayString::Remove (const wxString & *sz*)

Removes the first item matching this value.

An assert failure is provoked by an attempt to remove an element which does not exist in debug build.

See also

Index()

void wxArrayString::RemoveAt (size_t *nIndex*, size_t *count* = `1`)

Removes *count* items starting at position *nIndex* from the array.

void wxArrayString::Shrink ( )

Releases the extra memory allocated by the array.

This function is useful to minimize the array memory consumption.

See also

Alloc()

void wxArrayString::Sort (bool *reverseOrder* = `false`)

Sorts the array in alphabetical order or in reverse alphabetical order if *reverseOrder* is true.

The sort is case-sensitive.

void wxArrayString::Sort (CompareFunction *compareFunction*)

Sorts the array using the specified *compareFunction* for item comparison.

CompareFunction is defined as a function taking two *const wxString&* parameters and returning an *int* value less than, equal to or greater than 0 if the first string is less than, equal to or greater than the second one.

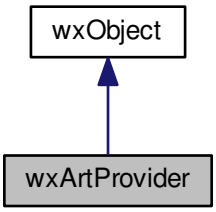
Example: The following example sorts strings by their length.

```
static int CompareStringLen(const wxString& first, const wxString& second)
{
    return first.length() - second.length();
}

...

wxArrayString array;

array.Add("one");
array.Add("two");
array.Add("three");
array.Add("four");

array.Sort(CompareStringLen);
```

21.37 wxArtProvider Class Reference

`#include <wx/artprov.h>`

Inheritance diagram for wxArtProvider:



21.37.1 Detailed Description

wxArtProvider class is used to customize the look of wxWidgets application.

When wxWidgets needs to display an icon or a bitmap (e.g. in the standard file dialog), it does not use a hard-coded resource but asks wxArtProvider for it instead. This way users can plug in their own wxArtProvider class and easily replace standard art with their own version.

All that is needed is to derive a class from wxArtProvider, override either its wxArtProvider::CreateBitmap() and/or its wxArtProvider::CreateIconBundle() methods and register the provider with wxArtProvider::Push():

```
class MyProvider : public wxArtProvider
{
protected:
  wxBitmap CreateBitmap(const wxArtID& id,
                        const wxArtClient& client,
                        const wxSize size)

  // optionally override this one as well
  wxIconBundle CreateIconBundle(const wxArtID& id,
                                const wxArtClient& client)
  { ... }
};
...
wxArtProvider::Push(new MyProvider);
```

If you need bitmap images (of the same artwork) that should be displayed at different sizes you should probably consider overriding wxArtProvider::CreateIconBundle and supplying icon bundles that contain different bitmap sizes.

There's another way of taking advantage of this class: you can use it in your code and use platform native icons as provided by wxArtProvider::GetBitmap or wxArtProvider::GetIcon.

21.37.2 Identifying art resources

Every bitmap and icon bundle are known to wxArtProvider under an unique ID that is used when requesting a resource from it. The ID is represented by the wxArtID type and can have one of these predefined values (you can see bitmaps represented by these constants in the Art Provider Sample):

- wxART_ERROR
- wxART_QUESTION
- wxART_WARNING
- wxART_INFORMATION
- wxART_ADD_BOOKMARK
- wxART_DEL_BOOKMARK
- wxART_HELP_SIDE_PANEL
- wxART_HELP_SETTINGS
- wxART_HELP_BOOK
- wxART_HELP_FOLDER
- wxART_HELP_PAGE
- wxART_GO_BACK
- wxART_GO_FORWARD
- wxART_GO_UP
- wxART_GO_DOWN
- wxART_GO_TO_PARENT
- wxART_GO_HOME
- wxART_GOTO_FIRST (since 2.9.2)
- wxART_GOTO_LAST (since 2.9.2)
- wxART_PRINT
- wxART_HELP
- wxART_TIP
- wxART_REPORT_VIEW
- wxART_LIST_VIEW
- wxART_NEW_DIR
- wxART_FOLDER
- wxART_FOLDER_OPEN
- wxART_GO_DIR_UP
- wxART_EXECUTABLE_FILE
- wxART_NORMAL_FILE
- wxART_TICK_MARK
- wxART_CROSS_MARK
- wxART_MISSING_IMAGE
- wxART_NEW
- wxART_FILE_OPEN
- wxART_FILE_SAVE
- wxART_FILE_SAVE_AS
- wxART_DELETE
- wxART_COPY
- wxART_CUT
- wxART_PASTE
- wxART_UNDO
- wxART_REDO
- wxART_PLUS (since 2.9.2)
- wxART_MINUS (since 2.9.2)
- wxART_CLOSE
- wxART_QUIT
- wxART_FIND
- wxART_FIND_AND_REPLACE
- wxART_FULL_SCREEN (since 3.1.0)
- wxART_HARDDISK
- wxART_FLOPPY
- wxART_CDROM
- wxART_REMOVABLE

Additionally, any string recognized by custom art providers registered using wxArtProvider::Push may be used.

Note

When running under GTK+ 2, GTK+ stock item IDs (e.g. "gtk-cdrom") may be used as well:

```
#ifdef __WXGTK__
    wxBitmap bmp = wxArtProvider::GetBitmap("gtk-cdrom",
      wxART_MENU);
#endif
```

For a list of the GTK+ stock items please refer to the GTK+ documentation page. It is also possible to load icons from the current icon theme by specifying their name (without extension and directory components). Icon themes recognized by GTK+ follow the freedesktop.org Icon Themes specification. Note that themes are not guaranteed to contain all icons, so wxArtProvider may return wxNullBitmap or wxNullIcon. The default theme is typically installed in /usr/share/icons/hicolor.

21.37.3 Clients

The *client* is the entity that calls wxArtProvider's GetBitmap() or GetIcon() function. It is represented by wxClientID type and can have one of these values:

- wxART_TOOLBAR
- wxART_MENU

- wxART_BUTTON
- wxART_FRAME_ICON
- wxART_CMN_DIALOG
- wxART_HELP_BROWSER
- wxART_MESSAGE_BOX
- wxART_OTHER (used for all requests that don't fit into any of the categories above)

Client ID serve as a hint to wxArtProvider that is supposed to help it to choose the best looking bitmap. For example it is often desirable to use slightly different icons in menus and toolbars even though they represent the same action (e.g. wxART_FILE_OPEN). Remember that this is really only a hint for wxArtProvider – it is common that wxArtProvider::GetBitmap returns identical bitmap for different client values!

Library: wxCore

Category: Miscellaneous

See also

Art Provider Sample for an example of wxArtProvider usage; stock ID list

Public Member Functions

- virtual ~wxArtProvider ()

  The destructor automatically removes the provider from the provider stack used by GetBitmap().

Static Public Member Functions

- static bool Delete (wxArtProvider *provider)

  Delete the given provider.
- static wxBitmap GetBitmap (const wxArtID &id, const wxArtClient &client=wxART_OTHER, const wxSize &size=wxDefaultSize)

  Query registered providers for bitmap with given ID.
- static wxIcon GetIcon (const wxArtID &id, const wxArtClient &client=wxART_OTHER, const wxSize &size=wxDefaultSize)

  Same as wxArtProvider::GetBitmap, but return a wxIcon object (or wxNullIcon on failure).
- static wxSize GetNativeSizeHint (const wxArtClient &client)

  Returns native icon size for use specified by client hint.
- static wxSize GetSizeHint (const wxArtClient &client, bool platform_default=false)

  Returns a suitable size hint for the given wxArtClient.
- static wxIconBundle GetIconBundle (const wxArtID &id, const wxArtClient &client=wxART_OTHER)

  Query registered providers for icon bundle with given ID.
- static bool HasNativeProvider ()

  Returns true if the platform uses native icons provider that should take precedence over any customizations.
- static void Insert (wxArtProvider *provider)
- static bool Pop ()

  Remove latest added provider and delete it.
- static void Push (wxArtProvider *provider)

  Register new art provider and add it to the top of providers stack (i.e.

- static void PushBack (wxArtProvider *provider)

  Register new art provider and add it to the bottom of providers stack.
- static bool Remove (wxArtProvider *provider)

  Remove a provider from the stack if it is on it.
- static wxArtID GetMessageBoxIconId (int flags)

  Helper used by GetMessageBoxIcon(): return the art id corresponding to the standard wxICON_INFORMATION/↩ WARNING/ERROR/QUESTION flags (only one can be set)
- static wxIcon GetMessageBoxIcon (int flags)

  Helper used by several generic classes: return the icon corresponding to the standard wxICON_INFORMATION/↩ WARNING/ERROR/QUESTION flags (only one can be set)

Protected Member Functions

- virtual wxBitmap CreateBitmap (const wxArtID &id, const wxArtClient &client, const wxSize &size)

  Derived art provider classes must override this method to create requested art resource.
- virtual wxIconBundle CreateIconBundle (const wxArtID &id, const wxArtClient &client)

  This method is similar to CreateBitmap() but can be used when a bitmap (or an icon) exists in several sizes.

Additional Inherited Members

21.37.4 Constructor & Destructor Documentation

virtual wxArtProvider::~wxArtProvider ( ) `[virtual]`

The destructor automatically removes the provider from the provider stack used by GetBitmap().

21.37.5 Member Function Documentation

virtual wxBitmap wxArtProvider::CreateBitmap (const wxArtID & *id*, const wxArtClient & *client*, const wxSize & *size*) `[protected], [virtual]`

Derived art provider classes must override this method to create requested art resource.

Note that returned bitmaps are cached by wxArtProvider and it is therefore not necessary to optimize CreateBitmap() for speed (e.g. you may create wxBitmap objects from XPMs here).

Parameters

id	wxArtID unique identifier of the bitmap.
client	wxArtClient identifier of the client (i.e. who is asking for the bitmap). This only servers as a hint.
size	Preferred size of the bitmap. The function may return a bitmap of different dimensions, it will be automatically rescaled to meet client's request.

Note

This is not part of wxArtProvider's public API, use wxArtProvider::GetBitmap or wxArtProvider::GetIconBundle or wxArtProvider::GetIcon to query wxArtProvider for a resource.

See also

CreateIconBundle()

virtual **wxIconBundle** wxArtProvider::CreateIconBundle (const **wxArtID** & *id*, const **wxArtClient** & *client*) [protected], [virtual]

This method is similar to CreateBitmap() but can be used when a bitmap (or an icon) exists in several sizes.

static bool wxArtProvider::Delete (**wxArtProvider** * *provider*) [static]

Delete the given *provider*.

static **wxBitmap** wxArtProvider::GetBitmap (const **wxArtID** & *id*, const **wxArtClient** & *client* = **wxART_OTHER**, const **wxSize** & *size* = **wxDefaultSize**) [static]

Query registered providers for bitmap with given ID.

Parameters

id	wxArtID unique identifier of the bitmap.
client	wxArtClient identifier of the client (i.e. who is asking for the bitmap).
size	Size of the returned bitmap or wxDefaultSize if size doesn't matter.

Returns

The bitmap if one of registered providers recognizes the ID or wxNullBitmap otherwise.

static **wxIcon** wxArtProvider::GetIcon (const **wxArtID** & *id*, const **wxArtClient** & *client* = **wxART_OTHER**, const **wxSize** & *size* = **wxDefaultSize**) [static]

Same as wxArtProvider::GetBitmap, but return a wxIcon object (or wxNullIcon on failure).

static **wxIconBundle** wxArtProvider::GetIconBundle (const **wxArtID** & *id*, const **wxArtClient** & *client* = **wxART_OTHER**) [static]

Query registered providers for icon bundle with given ID.

Parameters

id	wxArtID unique identifier of the icon bundle.
client	wxArtClient identifier of the client (i.e. who is asking for the icon bundle).

Returns

The icon bundle if one of registered providers recognizes the ID or wxNullIconBundle otherwise.

static **wxIcon** wxArtProvider::GetMessageBoxIcon (int *flags*) [static]

Helper used by several generic classes: return the icon corresponding to the standard wxICON_INFORMATION/↩ WARNING/ERROR/QUESTION flags (only one can be set)

static **wxArtID** wxArtProvider::GetMessageBoxIconId (int *flags*) [static]

Helper used by GetMessageBoxIcon(): return the art id corresponding to the standard wxICON_INFORMATION/↩ WARNING/ERROR/QUESTION flags (only one can be set)

static wxSize wxArtProvider::GetNativeSizeHint (const wxArtClient & *client*) `[static]`

Returns native icon size for use specified by *client* hint.

If the platform has no commonly used default for this use or if *client* is not recognized, returns wxDefaultSize.

Note

In some cases, a platform may have *several* appropriate native sizes (for example, wxART_FRAME_ICON for frame icons). In that case, this method returns only one of them, picked reasonably.

Since

2.9.0

static wxSize wxArtProvider::GetSizeHint (const wxArtClient & *client*, bool *platform_default* = `false`) `[static]`

Returns a suitable size hint for the given *wxArtClient*.

If *platform_default* is true, return a size based on the current platform using GetNativeSizeHint(), otherwise return the size from the topmost wxArtProvider. *wxDefaultSize* may be returned if the client doesn't have a specified size, like wxART_OTHER for example.

See also

GetNativeSizeHint()

static bool wxArtProvider::HasNativeProvider ( ) `[static]`

Returns true if the platform uses native icons provider that should take precedence over any customizations.

This is true for any platform that has user-customizable icon themes, currently only wxGTK.

A typical use for this method is to decide whether a custom art provider should be plugged in using Push() or PushBack().

Since

2.9.0

static void wxArtProvider::Insert (wxArtProvider * *provider*) `[static]`

Deprecated Use PushBack() instead.

static bool wxArtProvider::Pop ( ) `[static]`

Remove latest added provider and delete it.

static void wxArtProvider::Push (wxArtProvider * *provider*) `[static]`

Register new art provider and add it to the top of providers stack (i.e.

it will be queried as the first provider).

See also

PushBack()

static void wxArtProvider::PushBack (wxArtProvider * *provider*) [static]

Register new art provider and add it to the bottom of providers stack.

In other words, it will be queried as the last one, after all others, including the default provider.

See also

Push()

Since

2.9.0

static bool wxArtProvider::Remove (wxArtProvider * *provider*) [static]

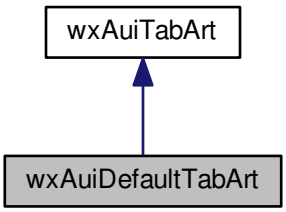
Remove a provider from the stack if it is on it.

The provider is not deleted, unlike when using Delete().

21.38 wxAuiDefaultTabArt Class Reference

`#include <wx/aui/auibook.h>`

Inheritance diagram for wxAuiDefaultTabArt:



21.38.1 Detailed Description

Default art provider for wxAuiNotebook.

See also

wxAuiTabArt

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiDefaultTabArt ()
- virtual ~wxAuiDefaultTabArt ()
- wxAuiTabArt * Clone ()

  Clones the art object.
- void SetFlags (unsigned int flags)

  Sets flags.
- void SetSizingInfo (const wxSize &tabCtrlSize, size_t tabCount)

  Sets sizing information.
- void SetNormalFont (const wxFont &font)

  Sets the normal font for drawing labels.
- void SetSelectedFont (const wxFont &font)

  Sets the font for drawing text for selected UI elements.
- void SetMeasuringFont (const wxFont &font)

  Sets the font used for calculating measurements.
- void SetColour (const wxColour &colour)

  Sets the colour of the inactive tabs.
- void SetActiveColour (const wxColour &colour)

  Sets the colour of the selected tab.
- void DrawBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)

  Draws a background on the given area.
- void DrawTab (wxDC &dc, wxWindow *wnd, const wxAuiNotebookPage &pane, const wxRect &inRect, int closeButtonState, wxRect *outTabRect, wxRect *outButtonRect, int *xExtent)

  Draws a tab.
- void DrawButton (wxDC &dc, wxWindow *wnd, const wxRect &inRect, int bitmapId, int buttonState, int orientation, wxRect *outRect)

  Draws a button.
- int GetIndentSize ()

  Returns the indent size.
- wxSize GetTabSize (wxDC &dc, wxWindow *wnd, const wxString &caption, const wxBitmap &bitmap, bool active, int closeButtonState, int *xExtent)

  Returns the tab size for the given caption, bitmap and state.
- int ShowDropDown (wxWindow *wnd, const wxAuiNotebookPageArray &items, int activeIdx)
- int GetBestTabCtrlSize (wxWindow *wnd, const wxAuiNotebookPageArray &pages, const wxSize &requiredBmpSize)

  Returns the tab control size.

Protected Attributes

- wxFont m_normalFont

  The font used for all tabs.
- wxFont m_selectedFont
- wxFont m_measuringFont

  The font used on the selected tab.
- wxColour m_baseColour
- wxPen m_baseColourPen
- wxPen m_borderPen
- wxBrush m_baseColourBrush
- wxColour m_activeColour
- wxBitmap m_activeCloseBmp
- wxBitmap m_disabledCloseBmp

- wxBitmap m_activeLeftBmp
- wxBitmap m_disabledLeftBmp
- wxBitmap m_activeRightBmp
- wxBitmap m_disabledRightBmp
- wxBitmap m_activeWindowListBmp
- wxBitmap m_disabledWindowListBmp
- int m_fixedTabWidth
- int m_tabCtrlHeight
- unsigned int m_flags

21.38.2 Constructor & Destructor Documentation

wxAuiDefaultTabArt::wxAuiDefaultTabArt ( )

virtual wxAuiDefaultTabArt::~wxAuiDefaultTabArt ( ) [virtual]

21.38.3 Member Function Documentation

wxAuiTabArt* wxAuiDefaultTabArt::Clone ( ) [virtual]

Clones the art object.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::DrawBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) [virtual]

Draws a background on the given area.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::DrawButton (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *in_rect*, int *bitmap_id*, int *button_state*, int *orientation*, wxRect * *out_rect*) [virtual]

Draws a button.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::DrawTab (wxDC & *dc*, wxWindow * *wnd*, const wxAuiNotebookPage & *page*, const wxRect & *rect*, int *close_button_state*, wxRect * *out_tab_rect*, wxRect * *out_button_rect*, int * *x_extent*) [virtual]

Draws a tab.

Implements wxAuiTabArt.

int wxAuiDefaultTabArt::GetBestTabCtrlSize (wxWindow * , const wxAuiNotebookPageArray & , const wxSize &) [virtual]

Returns the tab control size.

Implements wxAuiTabArt.

int wxAuiDefaultTabArt::GetIndentSize ( ) [virtual]

Returns the indent size.

Implements wxAuiTabArt.

wxSize wxAuiDefaultTabArt::GetTabSize (wxDC & *dc,* wxWindow * *wnd,* const wxString & *caption,* const wxBitmap & *bitmap,* bool *active,* int *close_button_state,* int * *x_extent*) [virtual]

Returns the tab size for the given caption, bitmap and state.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetActiveColour (const wxColour & *colour*) [virtual]

Sets the colour of the selected tab.

Since

2.9.2

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetColour (const wxColour & *colour*) [virtual]

Sets the colour of the inactive tabs.

Since

2.9.2

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetFlags (unsigned int *flags*) [virtual]

Sets flags.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetMeasuringFont (const wxFont & *font*) [virtual]

Sets the font used for calculating measurements.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetNormalFont (const wxFont & *font*) [virtual]

Sets the normal font for drawing labels.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetSelectedFont (const wxFont & *font*) [virtual]

Sets the font for drawing text for selected UI elements.

Implements wxAuiTabArt.

void wxAuiDefaultTabArt::SetSizingInfo (const wxSize & *tab_ctrl_size,* size_t *tab_count*) [virtual]

Sets sizing information.

Implements wxAuiTabArt.

int wxAuiDefaultTabArt::ShowDropDown (wxWindow * *wnd,* const wxAuiNotebookPageArray & *items,* int *activeIdx*)

21.38.4 Member Data Documentation

wxBitmap wxAuiDefaultTabArt::m_activeCloseBmp [protected]

wxColour wxAuiDefaultTabArt::m_activeColour [protected]

wxBitmap wxAuiDefaultTabArt::m_activeLeftBmp [protected]

wxBitmap wxAuiDefaultTabArt::m_activeRightBmp [protected]

wxBitmap wxAuiDefaultTabArt::m_activeWindowListBmp [protected]

wxColour wxAuiDefaultTabArt::m_baseColour [protected]

wxBrush wxAuiDefaultTabArt::m_baseColourBrush [protected]

wxPen wxAuiDefaultTabArt::m_baseColourPen [protected]

wxPen wxAuiDefaultTabArt::m_borderPen [protected]

wxBitmap wxAuiDefaultTabArt::m_disabledCloseBmp [protected]

wxBitmap wxAuiDefaultTabArt::m_disabledLeftBmp [protected]

wxBitmap wxAuiDefaultTabArt::m_disabledRightBmp [protected]

wxBitmap wxAuiDefaultTabArt::m_disabledWindowListBmp [protected]

int wxAuiDefaultTabArt::m_fixedTabWidth [protected]

unsigned int wxAuiDefaultTabArt::m_flags [protected]

wxFont wxAuiDefaultTabArt::m_measuringFont [protected]

The font used on the selected tab.

wxFont wxAuiDefaultTabArt::m_normalFont [protected]

The font used for all tabs.

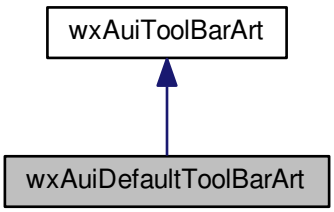
wxFont wxAuiDefaultTabArt::m_selectedFont [protected]

int wxAuiDefaultTabArt::m_tabCtrlHeight [protected]

21.39 wxAuiDefaultToolBarArt Class Reference

`#include <wx/aui/auibar.h>`

Inheritance diagram for wxAuiDefaultToolBarArt:



21.39.1 Detailed Description

wxAuiDefaultToolBarArt is part of the wxAUI class framework.

See also wxAuiToolBarArt , wxAuiToolBar and wxAUI Overview.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiDefaultToolBarArt ()
- virtual ~wxAuiDefaultToolBarArt ()
- virtual wxAuiToolBarArt * Clone ()
- virtual void SetFlags (unsigned int flags)
- virtual unsigned int GetFlags ()
- virtual void SetFont (const wxFont &font)
- virtual wxFont GetFont ()
- virtual void SetTextOrientation (int orientation)
- virtual int GetTextOrientation ()
- virtual void DrawBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)
- virtual void DrawPlainBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)
- virtual void DrawLabel (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)
- virtual void DrawButton (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)
- virtual void DrawDropDownButton (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)
- virtual void DrawControlLabel (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)
- virtual void DrawSeparator (wxDC &dc, wxWindow *wnd, const wxRect &rect)
- virtual void DrawGripper (wxDC &dc, wxWindow *wnd, const wxRect &rect)
- virtual void DrawOverflowButton (wxDC &dc, wxWindow *wnd, const wxRect &rect, int state)
- virtual wxSize GetLabelSize (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item)
- virtual wxSize GetToolSize (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item)
- virtual int GetElementSize (int element)
- virtual void SetElementSize (int element_id, int size)
- virtual int ShowDropDown (wxWindow *wnd, const wxAuiToolBarItemArray &items)

21.39.2 Constructor & Destructor Documentation

wxAuiDefaultToolBarArt::wxAuiDefaultToolBarArt ( )

virtual wxAuiDefaultToolBarArt::~wxAuiDefaultToolBarArt ( ) `[virtual]`

21.39.3 Member Function Documentation

virtual wxAuiToolBarArt* wxAuiDefaultToolBarArt::Clone ( ) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawButton (wxDC & *dc*, wxWindow * *wnd*, const wxAuiToolBarItem & *item*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawControlLabel (wxDC & *dc*, wxWindow * *wnd*, const wxAuiToolBarItem & *item*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawDropDownButton (wxDC & *dc*, wxWindow * *wnd*, const wxAuiToolBarItem & *item*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawGripper (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawLabel (wxDC & *dc*, wxWindow * *wnd*, const wxAuiToolBarItem & *item*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawOverflowButton (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*, int *state*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawPlainBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) `[virtual]`

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::DrawSeparator (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) [virtual]

Implements wxAuiToolBarArt.

virtual int wxAuiDefaultToolBarArt::GetElementSize (int *element*) [virtual]

Implements wxAuiToolBarArt.

virtual unsigned int wxAuiDefaultToolBarArt::GetFlags ( ) [virtual]

Implements wxAuiToolBarArt.

virtual wxFont wxAuiDefaultToolBarArt::GetFont ( ) [virtual]

Implements wxAuiToolBarArt.

virtual wxSize wxAuiDefaultToolBarArt::GetLabelSize (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item*) [virtual]

Implements wxAuiToolBarArt.

virtual int wxAuiDefaultToolBarArt::GetTextOrientation ( ) [virtual]

Implements wxAuiToolBarArt.

virtual wxSize wxAuiDefaultToolBarArt::GetToolSize (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item*) [virtual]

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::SetElementSize (int *element_id,* int *size*) [virtual]

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::SetFlags (unsigned int *flags*) [virtual]

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::SetFont (const wxFont & *font*) [virtual]

Implements wxAuiToolBarArt.

virtual void wxAuiDefaultToolBarArt::SetTextOrientation (int *orientation*) [virtual]

Implements wxAuiToolBarArt.

virtual int wxAuiDefaultToolBarArt::ShowDropDown (wxWindow * *wnd*, const wxAuiToolBarItemArray & *items*) [virtual]

Implements wxAuiToolBarArt.

21.40 wxAuiDockArt Class Reference

`#include <wx/aui/dockart.h>`

21.40.1 Detailed Description

wxAuiDockArt is part of the wxAUI class framework.

See also wxAUI Overview.

wxAuiDockArt is the art provider: provides all drawing functionality to the wxAui dock manager. This allows the dock manager to have a plugable look-and-feel.

By default, a wxAuiManager uses an instance of this class called wxAuiDefaultDockArt which provides bitmap art and a colour scheme that is adapted to the major platforms' look. You can either derive from that class to alter its behaviour or write a completely new dock art class. Call wxAuiManager::SetArtProvider to force wxAUI to use your new dock art provider.

Library: wxAui

Category: Window Docking (wxAUI)

See also

wxAuiManager, wxAuiPaneInfo

Public Member Functions

- wxAuiDockArt ()

  Constructor.
- virtual ~wxAuiDockArt ()

  Destructor.
- virtual void DrawBackground (wxDC &dc, wxWindow *window, int orientation, const wxRect &rect)=0

  Draws a background.
- virtual void DrawBorder (wxDC &dc, wxWindow *window, const wxRect &rect, wxAuiPaneInfo &pane)=0

  Draws a border.
- virtual void DrawCaption (wxDC &dc, wxWindow *window, const wxString &text, const wxRect &rect, wxAuiPaneInfo &pane)=0

  Draws a caption.
- virtual void DrawGripper (wxDC &dc, wxWindow *window, const wxRect &rect, wxAuiPaneInfo &pane)=0

  Draws a gripper.
- virtual void DrawPaneButton (wxDC &dc, wxWindow *window, int button, int button_state, const wxRect &rect, wxAuiPaneInfo &pane)=0

  Draws a button in the pane's title bar.
- virtual void DrawSash (wxDC &dc, wxWindow *window, int orientation, const wxRect &rect)=0

  Draws a sash between two windows.
- virtual wxColour GetColour (int id)=0

Get the colour of a certain setting.

- virtual wxFont GetFont (int id)=0

  Get a font setting.

- virtual int GetMetric (int id)=0

  Get the value of a certain setting.

- virtual void SetColour (int id, const wxColour &colour)=0

  Set a certain setting with the value colour.

- virtual void SetFont (int id, const wxFont &font)=0

  Set a font setting.

- virtual void SetMetric (int id, int new_val)=0

  Set a certain setting with the value new_val.

21.40.2 Constructor & Destructor Documentation

wxAuiDockArt::wxAuiDockArt ( )

Constructor.

virtual wxAuiDockArt::~wxAuiDockArt ( ) [virtual]

Destructor.

21.40.3 Member Function Documentation

virtual void wxAuiDockArt::DrawBackground (wxDC & *dc*, wxWindow * *window*, int *orientation*, const wxRect & *rect*) [pure virtual]

Draws a background.

virtual void wxAuiDockArt::DrawBorder (wxDC & *dc*, wxWindow * *window*, const wxRect & *rect*, wxAuiPaneInfo & *pane*) [pure virtual]

Draws a border.

virtual void wxAuiDockArt::DrawCaption (wxDC & *dc*, wxWindow * *window*, const wxString & *text*, const wxRect & *rect*, wxAuiPaneInfo & *pane*) [pure virtual]

Draws a caption.

virtual void wxAuiDockArt::DrawGripper (wxDC & *dc*, wxWindow * *window*, const wxRect & *rect*, wxAuiPaneInfo & *pane*) [pure virtual]

Draws a gripper.

virtual void wxAuiDockArt::DrawPaneButton (wxDC & *dc*, wxWindow * *window*, int *button*, int *button_state*, const wxRect & *rect*, wxAuiPaneInfo & *pane*) [pure virtual]

Draws a button in the pane's title bar.

button can be one of the values of **wxAuiButtonId**. *button_state* can be one of the values of **wxAuiPaneButtonState**.

virtual void wxAuiDockArt::DrawSash (wxDC & *dc,* wxWindow * *window,* int *orientation,* const wxRect & *rect*) [pure virtual]

Draws a sash between two windows.

virtual wxColour wxAuiDockArt::GetColour (int *id*) [pure virtual]

Get the colour of a certain setting.

id can be one of the colour values of **wxAuiPaneDockArtSetting**.

virtual wxFont wxAuiDockArt::GetFont (int *id*) [pure virtual]

Get a font setting.

virtual int wxAuiDockArt::GetMetric (int *id*) [pure virtual]

Get the value of a certain setting.

id can be one of the size values of **wxAuiPaneDockArtSetting**.

virtual void wxAuiDockArt::SetColour (int *id,* const wxColour & *colour*) [pure virtual]

Set a certain setting with the value *colour*.

id can be one of the colour values of **wxAuiPaneDockArtSetting**.

virtual void wxAuiDockArt::SetFont (int *id,* const wxFont & *font*) [pure virtual]

Set a font setting.

virtual void wxAuiDockArt::SetMetric (int *id,* int *new_val*) [pure virtual]

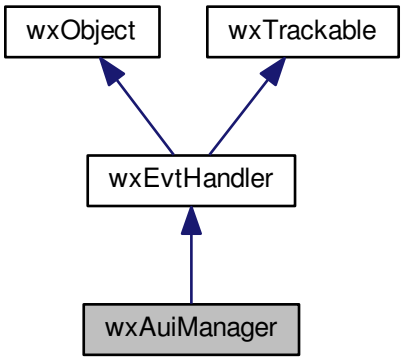
Set a certain setting with the value *new_val*.

id can be one of the size values of **wxAuiPaneDockArtSetting**.

21.41 wxAuiManager Class Reference

```
#include <wx/aui/framemanager.h>
```

Inheritance diagram for wxAuiManager:



21.41.1 Detailed Description

wxAuiManager is the central class of the wxAUI class framework.

wxAuiManager manages the panes associated with it for a particular wxFrame, using a pane's wxAuiPaneInfo information to determine each pane's docking and floating behaviour.

wxAuiManager uses wxWidgets' sizer mechanism to plan the layout of each frame. It uses a replaceable dock art class to do all drawing, so all drawing is localized in one area, and may be customized depending on an application's specific needs.

wxAuiManager works as follows: the programmer adds panes to the class, or makes changes to existing pane properties (dock position, floating state, show state, etc.). To apply these changes, wxAuiManager's Update() function is called. This batch processing can be used to avoid flicker, by modifying more than one pane at a time, and then "committing" all of the changes at once by calling Update().

Panes can be added quite easily:

```
wxTextCtrl* text1 = new wxTextCtrl(this, -1);
wxTextCtrl* text2 = new wxTextCtrl(this, -1);
m_mgr.AddPane(text1, wxLEFT, "Pane Caption");
m_mgr.AddPane(text2, wxBOTTOM, "Pane Caption");
m_mgr.Update();
```

Later on, the positions can be modified easily. The following will float an existing pane in a tool window:

```
m_mgr.GetPane(text1).Float();
```

21.41.2 Layers, Rows and Directions, Positions

Inside wxAUI, the docking layout is figured out by checking several pane parameters. Four of these are important for determining where a pane will end up:

- Direction: Each docked pane has a direction, Top, Bottom, Left, Right, or Center. This is fairly self-explanatory. The pane will be placed in the location specified by this variable.
- Position: More than one pane can be placed inside of a dock. Imagine two panes being docked on the left side of a window. One pane can be placed over another. In proportionally managed docks, the pane position

indicates its sequential position, starting with zero. So, in our scenario with two panes docked on the left side, the top pane in the dock would have position 0, and the second one would occupy position 1.

- Row: A row can allow for two docks to be placed next to each other. One of the most common places for this to happen is in the toolbar. Multiple toolbar rows are allowed, the first row being row 0, and the second row 1. Rows can also be used on vertically docked panes.
- Layer: A layer is akin to an onion. Layer 0 is the very center of the managed pane. Thus, if a pane is in layer 0, it will be closest to the center window (also sometimes known as the "content window"). Increasing layers "swallow up" all layers of a lower value. This can look very similar to multiple rows, but is different because all panes in a lower level yield to panes in higher levels. The best way to understand layers is by running the wxAUI sample.

Styles

This class supports the following styles:

- wxAUI_MGR_ALLOW_FLOATING: Allow a pane to be undocked to take the form of a wxMiniFrame.
- wxAUI_MGR_ALLOW_ACTIVE_PANE: Change the color of the title bar of the pane when it is activated.
- wxAUI_MGR_TRANSPARENT_DRAG: Make the pane transparent during its movement.
- wxAUI_MGR_TRANSPARENT_HINT: The possible location for docking is indicated by a translucent area.
- wxAUI_MGR_VENETIAN_BLINDS_HINT: The possible location for docking is indicated by gradually appearing partially transparent hint.
- wxAUI_MGR_RECTANGLE_HINT: The possible location for docking is indicated by a rectangular outline.
- wxAUI_MGR_HINT_FADE: The translucent area where the pane could be docked appears gradually.
- wxAUI_MGR_NO_VENETIAN_BLINDS_FADE: Used in complement of wxAUI_MGR_VENETIAN_BLIND↩ S_HINT to show the docking hint immediately.
- wxAUI_MGR_LIVE_RESIZE: When a docked pane is resized, its content is refreshed in live (instead of moving the border alone and refreshing the content at the end).
- wxAUI_MGR_DEFAULT: Default behavior, combines: wxAUI_MGR_ALLOW_FLOATING | wxAUI_MGR_↩ TRANSPARENT_HINT | wxAUI_MGR_HINT_FADE | wxAUI_MGR_NO_VENETIAN_BLINDS_FADE.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxAuiManagerEvent& event)

Event macros for events emitted by this class:

- EVT_AUI_PANE_BUTTON(func): Triggered when any button is pressed for any docked panes.
- EVT_AUI_PANE_CLOSE(func): Triggered when a docked or floating pane is closed.
- EVT_AUI_PANE_MAXIMIZE(func): Triggered when a pane is maximized.
- EVT_AUI_PANE_RESTORE(func): Triggered when a pane is restored.
- EVT_AUI_PANE_ACTIVATED(func): Triggered when a pane is made 'active'. This event is new since wx↩ Widgets 2.9.4.
- EVT_AUI_RENDER(func): This event can be caught to override the default renderer in order to custom draw your wxAuiManager window (not recommended).

Library: wxAui

Category: Window Docking (wxAUI)

See also

wxAUI Overview, wxAuiNotebook, wxAuiDockArt, wxAuiPaneInfo

Public Member Functions

- wxAuiManager (wxWindow *managed_wnd=NULL, unsigned int flags=wxAUI_MGR_DEFAULT)

  Constructor.
- virtual ~wxAuiManager ()

  Dtor.
- bool DetachPane (wxWindow *window)

  Tells the wxAuiManager to stop managing the pane specified by window.
- wxAuiPaneInfoArray & GetAllPanes ()

  Returns an array of all panes managed by the frame manager.
- wxAuiDockArt * GetArtProvider () const

  Returns the current art provider being used.
- void GetDockSizeConstraint (double *widthpct, double *heightpct) const

  Returns the current dock constraint values.
- unsigned int GetFlags () const

  Returns the current wxAuiManagerOption's flags.
- wxWindow * GetManagedWindow () const

  Returns the frame currently being managed by wxAuiManager.
- virtual void HideHint ()

  HideHint() hides any docking hint that may be visible.
- bool InsertPane (wxWindow *window, const wxAuiPaneInfo &insert_location, int insert_level=wxAUI_INSE↩ RT_PANE)

  This method is used to insert either a previously unmanaged pane window into the frame manager, or to insert a currently managed pane somewhere else.
- void LoadPaneInfo (wxString pane_part, wxAuiPaneInfo &pane)

  LoadPaneInfo() is similar to LoadPerspective, with the exception that it only loads information about a single pane.
- bool LoadPerspective (const wxString &perspective, bool update=true)

  Loads a saved perspective.
- wxString SavePaneInfo (wxAuiPaneInfo &pane)

  SavePaneInfo() is similar to SavePerspective, with the exception that it only saves information about a single pane.
- wxString SavePerspective ()

  Saves the entire user interface layout into an encoded wxString, which can then be stored by the application (probably using wxConfig).
- void SetArtProvider (wxAuiDockArt *art_provider)

  Instructs wxAuiManager to use art provider specified by parameter art_provider for all drawing calls.
- void SetDockSizeConstraint (double widthpct, double heightpct)

  When a user creates a new dock by dragging a window into a docked position, often times the large size of the window will create a dock that is unwieldly large.
- void SetFlags (unsigned int flags)

  This method is used to specify wxAuiManagerOption's flags.
- void SetManagedWindow (wxWindow *managed_wnd)

  Called to specify the frame or window which is to be managed by wxAuiManager.
- virtual void ShowHint (const wxRect &rect)

This function is used by controls to explicitly show a hint window at the specified rectangle.

- void UnInit ()

  Uninitializes the framework and should be called before a managed frame or window is destroyed.

- void Update ()

  This method is called after any number of changes are made to any of the managed panes.

- bool AddPane (wxWindow *window, const wxAuiPaneInfo &pane_info)

  AddPane() tells the frame manager to start managing a child window.

- bool AddPane (wxWindow *window, int direction=wxLEFT, const wxString &caption=wxEmptyString)

  AddPane() tells the frame manager to start managing a child window.

- bool AddPane (wxWindow *window, const wxAuiPaneInfo &pane_info, const wxPoint &drop_pos)

  AddPane() tells the frame manager to start managing a child window.

- wxAuiPaneInfo & GetPane (wxWindow *window)

  GetPane() is used to lookup a wxAuiPaneInfo object either by window pointer or by pane name, which acts as a unique id for a window pane.

- wxAuiPaneInfo & GetPane (const wxString &name)

  GetPane() is used to lookup a wxAuiPaneInfo object either by window pointer or by pane name, which acts as a unique id for a window pane.

Static Public Member Functions

- static wxAuiManager * GetManager (wxWindow *window)

  Calling this method will return the wxAuiManager for a given window.

Protected Member Functions

- virtual bool ProcessDockResult (wxAuiPaneInfo &target, const wxAuiPaneInfo &new_pos)

  ProcessDockResult() is a protected member of the wxAUI layout manager.

Additional Inherited Members

21.41.3 Constructor & Destructor Documentation

wxAuiManager::wxAuiManager (wxWindow * *managed_wnd* = NULL, unsigned int *flags* = wxAUI_MGR_DEFAULT)

Constructor.

Parameters

managed_wnd	Specifies the wxFrame which should be managed.
flags	Specifies the frame management behaviour and visual effects with the wxAuiManager↩ Option's style flags.

virtual wxAuiManager::~wxAuiManager ( ) [virtual]

Dtor.

21.41.4 Member Function Documentation

bool wxAuiManager::AddPane (wxWindow * *window,* const wxAuiPaneInfo & *pane_info*)

AddPane() tells the frame manager to start managing a child window.

There are several versions of this function. The first version allows the full spectrum of pane parameter possibilities. The second version is used for simpler user interfaces which do not require as much configuration. The last version allows a drop position to be specified, which will determine where the pane will be added.

bool wxAuiManager::AddPane (wxWindow * *window,* int *direction* = wxLEFT, const wxString & *caption* = wxEmptyString)

AddPane() tells the frame manager to start managing a child window.

There are several versions of this function. The first version allows the full spectrum of pane parameter possibilities. The second version is used for simpler user interfaces which do not require as much configuration. The last version allows a drop position to be specified, which will determine where the pane will be added.

bool wxAuiManager::AddPane (wxWindow * *window,* const wxAuiPaneInfo & *pane_info,* const wxPoint & *drop_pos*)

AddPane() tells the frame manager to start managing a child window.

There are several versions of this function. The first version allows the full spectrum of pane parameter possibilities. The second version is used for simpler user interfaces which do not require as much configuration. The last version allows a drop position to be specified, which will determine where the pane will be added.

bool wxAuiManager::DetachPane (wxWindow * *window*)

Tells the wxAuiManager to stop managing the pane specified by window.

The window, if in a floated frame, is reparented to the frame managed by wxAuiManager.

wxAuiPaneInfoArray& wxAuiManager::GetAllPanes ( )

Returns an array of all panes managed by the frame manager.

wxAuiDockArt* wxAuiManager::GetArtProvider ( ) const

Returns the current art provider being used.

See also

wxAuiDockArt.

void wxAuiManager::GetDockSizeConstraint (double * *widthpct,* double * *heightpct*) const

Returns the current dock constraint values.

See SetDockSizeConstraint() for more information.

unsigned int wxAuiManager::GetFlags ( ) const

Returns the current wxAuiManagerOption's flags.

wxWindow* wxAuiManager::GetManagedWindow ( ) const

Returns the frame currently being managed by wxAuiManager.

static wxAuiManager* wxAuiManager::GetManager (wxWindow * *window*) [static]

Calling this method will return the wxAuiManager for a given window.

The *window* parameter should specify any child window or sub-child window of the frame or window managed by wxAuiManager.

The *window* parameter need not be managed by the manager itself, nor does it even need to be a child or sub-child of a managed window. It must however be inside the window hierarchy underneath the managed window.

wxAuiPaneInfo& wxAuiManager::GetPane (wxWindow * *window*)

GetPane() is used to lookup a wxAuiPaneInfo object either by window pointer or by pane name, which acts as a unique id for a window pane.

The returned wxAuiPaneInfo object may then be modified to change a pane's look, state or position. After one or more modifications to wxAuiPaneInfo, wxAuiManager::Update() should be called to commit the changes to the user interface. If the lookup failed (meaning the pane could not be found in the manager), a call to the returned wxAuiPaneInfo's IsOk() method will return false.

wxAuiPaneInfo& wxAuiManager::GetPane (const wxString & *name*)

GetPane() is used to lookup a wxAuiPaneInfo object either by window pointer or by pane name, which acts as a unique id for a window pane.

The returned wxAuiPaneInfo object may then be modified to change a pane's look, state or position. After one or more modifications to wxAuiPaneInfo, wxAuiManager::Update() should be called to commit the changes to the user interface. If the lookup failed (meaning the pane could not be found in the manager), a call to the returned wxAuiPaneInfo's IsOk() method will return false.

virtual void wxAuiManager::HideHint ( ) [virtual]

HideHint() hides any docking hint that may be visible.

bool wxAuiManager::InsertPane (wxWindow * *window,* const wxAuiPaneInfo & *insert_location,* int *insert_level* = wxAUI_INSERT_PANE)

This method is used to insert either a previously unmanaged pane window into the frame manager, or to insert a currently managed pane somewhere else.

InsertPane() will push all panes, rows, or docks aside and insert the window into the position specified by *insert_location*.

Because *insert_location* can specify either a pane, dock row, or dock layer, the *insert_level* parameter is used to disambiguate this. The parameter *insert_level* can take a value of wxAUI_INSERT_PANE, wxAUI_INSERT_ROW or wxAUI_INSERT_DOCK.

void wxAuiManager::LoadPaneInfo (wxString *pane_part,* wxAuiPaneInfo & *pane*)

LoadPaneInfo() is similar to LoadPerspective, with the exception that it only loads information about a single pane.

It is used in combination with SavePaneInfo().

bool wxAuiManager::LoadPerspective (const wxString & *perspective,* bool *update* = `true`)

Loads a saved perspective.

If update is true, wxAuiManager::Update() is automatically invoked, thus realizing the saved perspective on screen.

virtual bool wxAuiManager::ProcessDockResult (wxAuiPaneInfo & *target,* const wxAuiPaneInfo & *new_pos*) `[protected], [virtual]`

ProcessDockResult() is a protected member of the wxAUI layout manager.

It can be overridden by derived classes to provide custom docking calculations.

wxString wxAuiManager::SavePaneInfo (wxAuiPaneInfo & *pane*)

SavePaneInfo() is similar to SavePerspective, with the exception that it only saves information about a single pane.

It is used in combination with LoadPaneInfo().

wxString wxAuiManager::SavePerspective ( )

Saves the entire user interface layout into an encoded wxString, which can then be stored by the application (probably using wxConfig).

When a perspective is restored using LoadPerspective(), the entire user interface will return to the state it was when the perspective was saved.

void wxAuiManager::SetArtProvider (wxAuiDockArt * *art_provider*)

Instructs wxAuiManager to use art provider specified by parameter *art_provider* for all drawing calls.

This allows plugable look-and-feel features. The previous art provider object, if any, will be deleted by wxAui↩ Manager.

See also

wxAuiDockArt.

void wxAuiManager::SetDockSizeConstraint (double *widthpct,* double *heightpct*)

When a user creates a new dock by dragging a window into a docked position, often times the large size of the window will create a dock that is unwieldly large.

wxAuiManager by default limits the size of any new dock to 1/3 of the window size. For horizontal docks, this would be 1/3 of the window height. For vertical docks, 1/3 of the width.

Calling this function will adjust this constraint value. The numbers must be between 0.0 and 1.0. For instance, calling SetDockSizeContraint with 0.5, 0.5 will cause new docks to be limited to half of the size of the entire managed window.

void wxAuiManager::SetFlags (unsigned int *flags*)

This method is used to specify wxAuiManagerOption's flags.

flags specifies options which allow the frame management behaviour to be modified.

void wxAuiManager::SetManagedWindow (wxWindow * *managed_wnd*)

Called to specify the frame or window which is to be managed by wxAuiManager.

Frame management is not restricted to just frames. Child windows or custom controls are also allowed.

virtual void wxAuiManager::ShowHint (const wxRect & *rect*) [virtual]

This function is used by controls to explicitly show a hint window at the specified rectangle.

It is rarely called, and is mostly used by controls implementing custom pane drag/drop behaviour. The specified rectangle should be in screen coordinates.

void wxAuiManager::UnInit ( )

Uninitializes the framework and should be called before a managed frame or window is destroyed.

UnInit() is usually called in the managed wxFrame's destructor. It is necessary to call this function before the managed frame or window is destroyed, otherwise the manager cannot remove its custom event handlers from a window.

void wxAuiManager::Update ( )

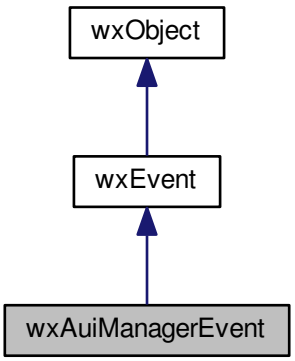
This method is called after any number of changes are made to any of the managed panes.

Update() must be invoked after AddPane() or InsertPane() are called in order to "realize" or "commit" the changes. In addition, any number of changes may be made to wxAuiPaneInfo structures (retrieved with wxAuiManager::GetPane), but to realize the changes, Update() must be called. This construction allows pane flicker to be avoided by updating the whole layout at one time.

21.42 wxAuiManagerEvent Class Reference

`#include <wx/aui/framemanager.h>`

Inheritance diagram for wxAuiManagerEvent:



21.42.1 Detailed Description

Event used to indicate various actions taken with wxAuiManager.

See wxAuiManager for available event types.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxAuiManagerEvent& event)

Event macros:

- EVT_AUI_PANE_BUTTON(func): Triggered when any button is pressed for any docked panes.
- EVT_AUI_PANE_CLOSE(func): Triggered when a docked or floating pane is closed.
- EVT_AUI_PANE_MAXIMIZE(func): Triggered when a pane is maximized.
- EVT_AUI_PANE_RESTORE(func): Triggered when a pane is restored.
- EVT_AUI_PANE_ACTIVATED(func): Triggered when a pane is made 'active'. This event is new since wx↩ Widgets 2.9.4.
- EVT_AUI_RENDER(func): This event can be caught to override the default renderer in order to custom draw your wxAuiManager window (not recommended).

Library: wxAui

Category: Events, Window Docking (wxAUI)

See also

wxAuiManager, wxAuiPaneInfo

Public Member Functions

- wxAuiManagerEvent (wxEventType type=wxEVT_NULL)

  Constructor.
- bool CanVeto ()
- int GetButton ()
- wxDC * GetDC ()
- bool GetVeto ()
- wxAuiManager * GetManager ()
- wxAuiPaneInfo * GetPane ()
- void SetButton (int button)

  Sets the ID of the button clicked that triggered this event.
- void SetCanVeto (bool can_veto)

  Sets whether or not this event can be vetoed.
- void SetDC (wxDC *pdc)
- void SetManager (wxAuiManager *manager)

  Sets the wxAuiManager this event is associated with.
- void SetPane (wxAuiPaneInfo *pane)

  Sets the pane this event is associated with.
- void Veto (bool veto=true)

  Cancels the action indicated by this event if CanVeto() is true.

Additional Inherited Members

21.42.2 Constructor & Destructor Documentation

wxAuiManagerEvent::wxAuiManagerEvent (wxEventType *type* = wxEVT_NULL)

Constructor.

21.42.3 Member Function Documentation

bool wxAuiManagerEvent::CanVeto ( )

Returns

true if this event can be vetoed.

See also

Veto()

int wxAuiManagerEvent::GetButton ( )

Returns

The ID of the button that was clicked.

wxDC* wxAuiManagerEvent::GetDC ( )

Todo What is this?

wxAuiManager* wxAuiManagerEvent::GetManager ( )

Returns

The wxAuiManager this event is associated with.

wxAuiPaneInfo* wxAuiManagerEvent::GetPane ( )

Returns

The pane this event is associated with.

bool wxAuiManagerEvent::GetVeto ( )

Returns

true if this event was vetoed.

See also

Veto()

void wxAuiManagerEvent::SetButton (int *button*)

Sets the ID of the button clicked that triggered this event.

void wxAuiManagerEvent::SetCanVeto (bool *can_veto*)

Sets whether or not this event can be vetoed.

void wxAuiManagerEvent::SetDC (wxDC * *pdc*)

Todo What is this?

void wxAuiManagerEvent::SetManager (wxAuiManager * *manager*)

Sets the wxAuiManager this event is associated with.

void wxAuiManagerEvent::SetPane (wxAuiPaneInfo * *pane*)

Sets the pane this event is associated with.

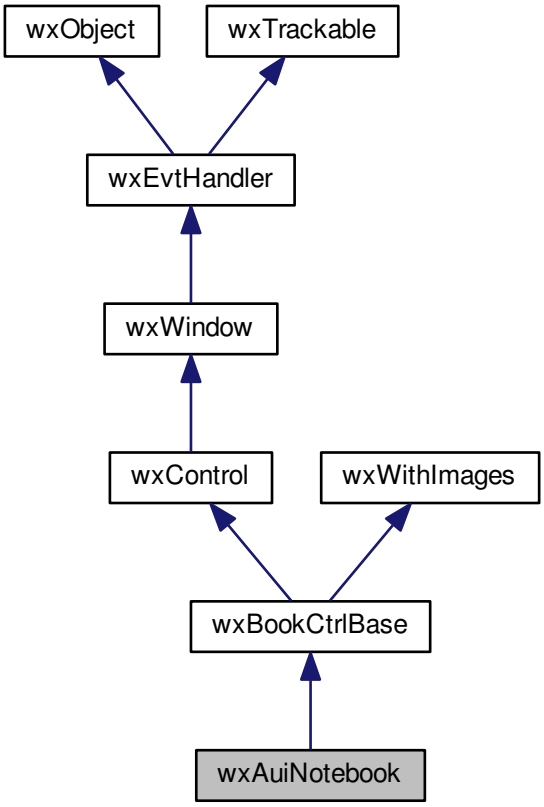
void wxAuiManagerEvent::Veto (bool *veto* = `true`)

Cancels the action indicated by this event if CanVeto() is true.

21.43 wxAuiNotebook Class Reference

`#include <wx/aui/auibook.h>`

Inheritance diagram for wxAuiNotebook:



21.43.1 Detailed Description

wxAuiNotebook is part of the wxAUI class framework, which represents a notebook control, managing multiple windows with associated tabs.

See also wxAUI Overview.

wxAuiNotebook is a notebook control which implements many features common in applications with dockable panes. Specifically, wxAuiNotebook implements functionality which allows the user to rearrange tab order via drag-and-drop, split the tab window into many different splitter configurations, and toggle through different themes to customize the control's look and feel.

The appearance of this class is configurable and can be changed by calling wxAuiNotebook::SetArtProvider(). By default, native art provider is used if available (currently only in wxGTK) and wxAuiGenericTabArt otherwise.

Styles

This class supports the following styles:

- wxAUI_NB_DEFAULT_STYLE: Defined as wxAUI_NB_TOP | wxAUI_NB_TAB_SPLIT | wxAUI_NB_TAB_↩ MOVE | wxAUI_NB_SCROLL_BUTTONS | wxAUI_NB_CLOSE_ON_ACTIVE_TAB | wxAUI_NB_MIDDLE↩ _CLICK_CLOSE.

- wxAUI_NB_TAB_SPLIT: Allows the tab control to be split by dragging a tab.
- wxAUI_NB_TAB_MOVE: Allows a tab to be moved horizontally by dragging.
- wxAUI_NB_TAB_EXTERNAL_MOVE: Allows a tab to be moved to another tab control.
- wxAUI_NB_TAB_FIXED_WIDTH: With this style, all tabs have the same width.
- wxAUI_NB_SCROLL_BUTTONS: With this style, left and right scroll buttons are displayed.
- wxAUI_NB_WINDOWLIST_BUTTON: With this style, a drop-down list of windows is available.
- wxAUI_NB_CLOSE_BUTTON: With this style, a close button is available on the tab bar.
- wxAUI_NB_CLOSE_ON_ACTIVE_TAB: With this style, the close button is visible on the active tab.
- wxAUI_NB_CLOSE_ON_ALL_TABS: With this style, the close button is visible on all tabs.
- wxAUI_NB_MIDDLE_CLICK_CLOSE: With this style, middle click on a tab closes the tab.
- wxAUI_NB_TOP: With this style, tabs are drawn along the top of the notebook.
- wxAUI_NB_BOTTOM: With this style, tabs are drawn along the bottom of the notebook.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxAuiNotebookEvent& event)

Event macros for events emitted by this class:

- EVT_AUINOTEBOOK_PAGE_CLOSE(id, func): A page is about to be closed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CLOSE` event.
- EVT_AUINOTEBOOK_PAGE_CLOSED(winid, fn): A page has been closed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CLOSED` event.
- EVT_AUINOTEBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CHANGED` event.
- EVT_AUINOTEBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CHANGING` event. This event can be vetoed.
- EVT_AUINOTEBOOK_BUTTON(id, func): The window list button has been pressed. Processes a `wxEVT_AUINOTEBOOK_BUTTON` event.
- EVT_AUINOTEBOOK_BEGIN_DRAG(id, func): Dragging is about to begin. Processes a `wxEVT_AUINOTEBOOK_BEGIN_DRAG` event.
- EVT_AUINOTEBOOK_END_DRAG(id, func): Dragging has ended. Processes a `wxEVT_AUINOTEBOOK_END_DRAG` event.
- EVT_AUINOTEBOOK_DRAG_MOTION(id, func): Emitted during a drag and drop operation. Processes a `wxEVT_AUINOTEBOOK_DRAG_MOTION` event.
- EVT_AUINOTEBOOK_ALLOW_DND(id, func): Whether to allow a tab to be dropped. Processes a `wxEVT_AUINOTEBOOK_ALLOW_DND` event. This event must be specially allowed.
- EVT_AUINOTEBOOK_DRAG_DONE(winid, fn): Notify that the tab has been dragged. Processes a `wxEVT_AUINOTEBOOK_DRAG_DONE` event.
- EVT_AUINOTEBOOK_TAB_MIDDLE_DOWN(winid, fn): The middle mouse button is pressed on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_MIDDLE_DOWN` event.
- EVT_AUINOTEBOOK_TAB_MIDDLE_UP(winid, fn): The middle mouse button is released on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_MIDDLE_UP` event.

- EVT_AUINOTEBOOK_TAB_RIGHT_DOWN(winid, fn): The right mouse button is pressed on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_RIGHT_DOWN` event.
- EVT_AUINOTEBOOK_TAB_RIGHT_UP(winid, fn): The right mouse button is released on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_RIGHT_UP` event.
- EVT_AUINOTEBOOK_BG_DCLICK(winid, fn): Double clicked on the tabs background area. Processes a `wxEVT_AUINOTEBOOK_BG_DCLICK` event.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiNotebook ()

  Default ctor.
- wxAuiNotebook (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxAUI_NB_DEFAULT_STYLE)

  Constructor.
- bool AddPage (wxWindow *page, const wxString &caption, bool select=false, const wxBitmap &bitmap=wxNullBitmap)

  Adds a page.
- virtual bool AddPage (wxWindow *page, const wxString &text, bool select, int imageId)

  Adds a new page.
- void AdvanceSelection (bool forward=true)

  Sets the selection to the next or previous page.
- virtual int ChangeSelection (size_t n)

  Changes the selection for the given page, returning the previous selection.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Creates the notebook window.
- virtual bool DeleteAllPages ()

  Deletes all pages.
- bool DeletePage (size_t page)

  Deletes a page at the given index.
- wxAuiTabArt * GetArtProvider () const

  Returns the associated art provider.
- wxWindow * GetCurrentPage () const

  Returns the currently selected page or NULL.
- int GetHeightForPageHeight (int pageHeight)

  Returns the desired height of the notebook for the given page height.
- wxWindow * GetPage (size_t page_idx) const

  Returns the page specified by the given index.
- wxBitmap GetPageBitmap (size_t page) const

  Returns the tab bitmap for the page.
- size_t GetPageCount () const

  Returns the number of pages in the notebook.
- int GetPageIndex (wxWindow *page_wnd) const

  Returns the page index for the specified window.
- wxString GetPageText (size_t page) const

Returns the tab label for the page.

- wxString GetPageToolTip (size_t pageIdx) const

  Returns the tooltip for the tab label of the page.
- int GetSelection () const

  Returns the currently selected page.
- int GetTabCtrlHeight () const

  Returns the height of the tab control.
- bool InsertPage (size_t page_idx, wxWindow *page, const wxString &caption, bool select=false, const wxBitmap &bitmap=wxNullBitmap)

  InsertPage() is similar to AddPage, but allows the ability to specify the insert location.
- virtual bool InsertPage (size_t index, wxWindow *page, const wxString &text, bool select=false, int imageId=NO_IMAGE)

  Inserts a new page at the specified position.
- bool RemovePage (size_t page)

  Removes a page, without deleting the window pointer.
- void SetArtProvider (wxAuiTabArt *art)

  Sets the art provider to be used by the notebook.
- virtual bool SetFont (const wxFont &font)

  Sets the font for drawing the tab labels, using a bold version of the font for selected tab labels.
- void SetMeasuringFont (const wxFont &font)

  Sets the font for measuring tab labels.
- void SetNormalFont (const wxFont &font)

  Sets the font for drawing unselected tab labels.
- bool SetPageBitmap (size_t page, const wxBitmap &bitmap)

  Sets the bitmap for the page.
- virtual bool SetPageImage (size_t n, int imageId)

  Sets the image index for the given page.
- bool SetPageText (size_t page, const wxString &text)

  Sets the tab label for the page.
- bool SetPageToolTip (size_t page, const wxString &text)

  Sets the tooltip displayed when hovering over the tab label of the page.
- void SetSelectedFont (const wxFont &font)

  Sets the font for drawing selected tab labels.
- size_t SetSelection (size_t new_page)

  Sets the page selection.
- virtual void SetTabCtrlHeight (int height)

  Sets the tab height.
- bool ShowWindowMenu ()

  Shows the window menu for the active tab control associated with this notebook, and returns true if a selection was made.

- virtual void SetUniformBitmapSize (const wxSize &size)

  Split performs a split operation programmatically.
- virtual void Split (size_t page, int direction)

  Split performs a split operation programmatically.

Additional Inherited Members

21.43.2 Constructor & Destructor Documentation

wxAuiNotebook::wxAuiNotebook ( )

Default ctor.

wxAuiNotebook::wxAuiNotebook (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `wxAUI_NB_DEFAULT_STYLE`)

Constructor.

Creates a wxAuiNotebok control.

21.43.3 Member Function Documentation

bool wxAuiNotebook::AddPage (wxWindow * *page*, const wxString & *caption*, bool *select* = `false`, const wxBitmap & *bitmap* = wxNullBitmap)

Adds a page.

If the *select* parameter is true, calling this will generate a page change event.

virtual bool wxAuiNotebook::AddPage (wxWindow * *page*, const wxString & *text*, bool *select*, int *imageId*) `[virtual]`

Adds a new page.

The page must have the book control itself as the parent and must not have been added to this control previously.

The call to this function may generate the page changing events.

Parameters

page	Specifies the new page.
text	Specifies the text for the new page.
select	Specifies whether the page should be selected.
imageId	Specifies the optional image index for the new page.

Returns

true if successful, false otherwise.

Remarks

Do not delete the page, it will be deleted by the book control.

See also

InsertPage()

Since

2.9.3

Reimplemented from wxBookCtrlBase.

void wxAuiNotebook::AdvanceSelection (bool *forward* = `true`)

Sets the selection to the next or previous page.

virtual int wxAuiNotebook::ChangeSelection (size_t *n*) `[virtual]`

Changes the selection for the given page, returning the previous selection.

This function behaves as SetSelection() but does *not* generate the page changing events.

See User Generated Events vs Programmatically Generated Events for more information.

Since

2.9.3

Implements wxBookCtrlBase.

bool wxAuiNotebook::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Creates the notebook window.

virtual bool wxAuiNotebook::DeleteAllPages ( ) `[virtual]`

Deletes all pages.

Since

2.9.3

Reimplemented from wxBookCtrlBase.

bool wxAuiNotebook::DeletePage (size_t *page*) `[virtual]`

Deletes a page at the given index.

Calling this method will generate a page change event.

Reimplemented from wxBookCtrlBase.

wxAuiTabArt* wxAuiNotebook::GetArtProvider ( ) const

Returns the associated art provider.

wxWindow* wxAuiNotebook::GetCurrentPage ( ) const

Returns the currently selected page or NULL.

Since

2.9.3

int wxAuiNotebook::GetHeightForPageHeight (int *pageHeight*)

Returns the desired height of the notebook for the given page height.

Use this to fit the notebook to a given page size.

wxWindow* wxAuiNotebook::GetPage (size_t *page_idx*) const

Returns the page specified by the given index.

wxBitmap wxAuiNotebook::GetPageBitmap (size_t *page*) const

Returns the tab bitmap for the page.

size_t wxAuiNotebook::GetPageCount ( ) const `[virtual]`

Returns the number of pages in the notebook.

Reimplemented from wxBookCtrlBase.

int wxAuiNotebook::GetPageIndex (wxWindow * *page_wnd*) const

Returns the page index for the specified window.

If the window is not found in the notebook, wxNOT_FOUND is returned.

wxString wxAuiNotebook::GetPageText (size_t *page*) const `[virtual]`

Returns the tab label for the page.

Implements wxBookCtrlBase.

wxString wxAuiNotebook::GetPageToolTip (size_t *pageIdx*) const

Returns the tooltip for the tab label of the page.

Since

2.9.4

int wxAuiNotebook::GetSelection ( ) const `[virtual]`

Returns the currently selected page.

Implements wxBookCtrlBase.

int wxAuiNotebook::GetTabCtrlHeight ( ) const

Returns the height of the tab control.

bool wxAuiNotebook::InsertPage (size_t *page_idx*, wxWindow * *page*, const wxString & *caption*, bool *select* = `false`, const wxBitmap & *bitmap* = wxNullBitmap)

InsertPage() is similar to AddPage, but allows the ability to specify the insert location.

If the *select* parameter is true, calling this will generate a page change event.

virtual bool wxAuiNotebook::InsertPage (size_t *index,* wxWindow * *page,* const wxString & *text,* bool *select* = `false`, int *imageId* = NO_IMAGE) [virtual]

Inserts a new page at the specified position.

Parameters

index	Specifies the position for the new page.
page	Specifies the new page.
text	Specifies the text for the new page.
select	Specifies whether the page should be selected.
imageId	Specifies the optional image index for the new page.

Returns

true if successful, false otherwise.

Remarks

Do not delete the page, it will be deleted by the book control.

See also

AddPage()

Since

2.9.3

Implements wxBookCtrlBase.

bool wxAuiNotebook::RemovePage (size_t *page*) [virtual]

Removes a page, without deleting the window pointer.

Reimplemented from wxBookCtrlBase.

void wxAuiNotebook::SetArtProvider (wxAuiTabArt * *art*)

Sets the art provider to be used by the notebook.

virtual bool wxAuiNotebook::SetFont (const wxFont & *font*) [virtual]

Sets the font for drawing the tab labels, using a bold version of the font for selected tab labels.

Reimplemented from wxWindow.

void wxAuiNotebook::SetMeasuringFont (const wxFont & *font*)

Sets the font for measuring tab labels.

void wxAuiNotebook::SetNormalFont (const wxFont & *font*)

Sets the font for drawing unselected tab labels.

bool wxAuiNotebook::SetPageBitmap (size_t *page*, const wxBitmap & *bitmap*)

Sets the bitmap for the page.

To remove a bitmap from the tab caption, pass wxNullBitmap.

virtual bool wxAuiNotebook::SetPageImage (size_t *n,* int *imageId*) `[virtual]`

Sets the image index for the given page.

image is an index into the image list which was set with SetImageList().

Since

2.9.3

Implements wxBookCtrlBase.

bool wxAuiNotebook::SetPageText (size_t *page,* const wxString & *text*) `[virtual]`

Sets the tab label for the page.

Implements wxBookCtrlBase.

bool wxAuiNotebook::SetPageToolTip (size_t *page,* const wxString & *text*)

Sets the tooltip displayed when hovering over the tab label of the page.

Returns

true if tooltip was updated, false if it failed, e.g. because the page index is invalid.

Since

2.9.4

void wxAuiNotebook::SetSelectedFont (const wxFont & *font*)

Sets the font for drawing selected tab labels.

size_t wxAuiNotebook::SetSelection (size_t *new_page*) `[virtual]`

Sets the page selection.

Calling this method will generate a page change event.

Implements wxBookCtrlBase.

virtual void wxAuiNotebook::SetTabCtrlHeight (int *height*) `[virtual]`

Sets the tab height.

By default, the tab control height is calculated by measuring the text height and bitmap sizes on the tab captions. Calling this method will override that calculation and set the tab control to the specified height parameter. A call to this method will override any call to SetUniformBitmapSize().

Specifying -1 as the height will return the control to its default auto-sizing behaviour.

virtual void wxAuiNotebook::SetUniformBitmapSize (const wxSize & *size*) `[virtual]`

Split performs a split operation programmatically.

The argument *page* indicates the page that will be split off. This page will also become the active page after the split.

The *direction* argument specifies where the pane should go, it should be one of the following: wxTOP, wxBOTTOM, wxLEFT, or wxRIGHT.

bool wxAuiNotebook::ShowWindowMenu ( )

Shows the window menu for the active tab control associated with this notebook, and returns true if a selection was made.

virtual void wxAuiNotebook::Split (size_t *page*, int *direction*) `[virtual]`

Split performs a split operation programmatically.

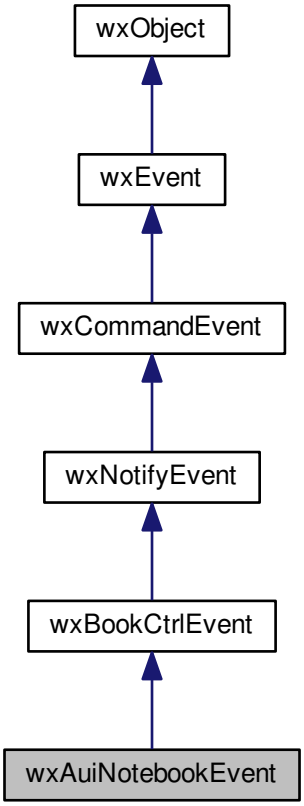
The argument *page* indicates the page that will be split off. This page will also become the active page after the split.

The *direction* argument specifies where the pane should go, it should be one of the following: wxTOP, wxBOTTOM, wxLEFT, or wxRIGHT.

21.44 wxAuiNotebookEvent Class Reference

```
#include <wx/aui/auibook.h>
```

Inheritance diagram for wxAuiNotebookEvent:



21.44.1 Detailed Description

This class is used by the events generated by wxAuiNotebook.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxAuiNotebookEvent& event)

Event macros for events emitted by this class:

- EVT_AUINOTEBOOK_PAGE_CLOSE(id, func): A page is about to be closed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CLOSE` event.
- EVT_AUINOTEBOOK_PAGE_CLOSED(winid, fn): A page has been closed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CLOSED` event.
- EVT_AUINOTEBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CHANGED` event.
- EVT_AUINOTEBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_AUINOTEBOOK_PAGE_CHANGING` event. This event can be vetoed.

- EVT_AUINOTEBOOK_BUTTON(id, func): The window list button has been pressed. Processes a `wxEVT_AUINOTEBOOK_BUTTON` event.
- EVT_AUINOTEBOOK_BEGIN_DRAG(id, func): Dragging is about to begin. Processes a `wxEVT_AUINOTEBOOK_BEGIN_DRAG` event.
- EVT_AUINOTEBOOK_END_DRAG(id, func): Dragging has ended. Processes a `wxEVT_AUINOTEBOOK_END_DRAG` event.
- EVT_AUINOTEBOOK_DRAG_MOTION(id, func): Emitted during a drag and drop operation. Processes a `wxEVT_AUINOTEBOOK_DRAG_MOTION` event.
- EVT_AUINOTEBOOK_ALLOW_DND(id, func): Whether to allow a tab to be dropped. Processes a `wxEVT_AUINOTEBOOK_ALLOW_DND` event. This event must be specially allowed.
- EVT_AUINOTEBOOK_DRAG_DONE(winid, fn): Notify that the tab has been dragged. Processes a `wxEVT_AUINOTEBOOK_DRAG_DONE` event.
- EVT_AUINOTEBOOK_TAB_MIDDLE_DOWN(winid, fn): The middle mouse button is pressed on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_MIDDLE_DOWN` event.
- EVT_AUINOTEBOOK_TAB_MIDDLE_UP(winid, fn): The middle mouse button is released on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_MIDDLE_UP` event.
- EVT_AUINOTEBOOK_TAB_RIGHT_DOWN(winid, fn): The right mouse button is pressed on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_RIGHT_DOWN` event.
- EVT_AUINOTEBOOK_TAB_RIGHT_UP(winid, fn): The right mouse button is released on a tab. Processes a `wxEVT_AUINOTEBOOK_TAB_RIGHT_UP` event.
- EVT_AUINOTEBOOK_BG_DCLICK(winid, fn): Double clicked on the tabs background area. Processes a `wxEVT_AUINOTEBOOK_BG_DCLICK` event.

Library: wxAui

Category: Events, Book Controls

See also

wxAuiNotebook, wxBookCtrlEvent

Public Member Functions

- wxAuiNotebookEvent (wxEventType command_type=wxEVT_NULL, int win_id=0)

  Constructor.
- wxEvent * Clone ()

Additional Inherited Members

21.44.2 Constructor & Destructor Documentation

wxAuiNotebookEvent::wxAuiNotebookEvent (wxEventType *command_type* = wxEVT_NULL, int *win_id* = 0)

Constructor.

21.44.3 Member Function Documentation

wxEvent* wxAuiNotebookEvent::Clone ( )

21.45 wxAuiPaneInfo Class Reference

`#include <wx/aui/framemanager.h>`

21.45.1 Detailed Description

wxAuiPaneInfo is part of the wxAUI class framework.

See also wxAUI Overview.

wxAuiPaneInfo specifies all the parameters for a pane. These parameters specify where the pane is on the screen, whether it is docked or floating, or hidden. In addition, these parameters specify the pane's docked position, floating position, preferred size, minimum size, caption text among many other parameters.

Library: wxAui

Category: Window Docking (wxAUI)

See also

wxAuiManager, wxAuiDockArt

Public Member Functions

- wxAuiPaneInfo ()
- wxAuiPaneInfo (const wxAuiPaneInfo &c)

  Copy constructor.
- wxAuiPaneInfo & Bottom ()

  Bottom() sets the pane dock position to the bottom side of the frame.
- wxAuiPaneInfo & BottomDockable (bool b=true)

  BottomDockable() indicates whether a pane can be docked at the bottom of the frame.
- wxAuiPaneInfo & Caption (const wxString &c)

  Caption() sets the caption of the pane.
- wxAuiPaneInfo & CaptionVisible (bool visible=true)

  CaptionVisible indicates that a pane caption should be visible.
- wxAuiPaneInfo & CloseButton (bool visible=true)

  CloseButton() indicates that a close button should be drawn for the pane.
- wxAuiPaneInfo & DefaultPane ()

  DefaultPane() specifies that the pane should adopt the default pane settings.
- wxAuiPaneInfo & DestroyOnClose (bool b=true)

  DestroyOnClose() indicates whether a pane should be destroyed when it is closed.
- wxAuiPaneInfo & Direction (int direction)

  Direction() determines the direction of the docked pane.
- wxAuiPaneInfo & Dock ()

  Dock() indicates that a pane should be docked.
- wxAuiPaneInfo & DockFixed (bool b=true)

  DockFixed() causes the containing dock to have no resize sash.

- wxAuiPaneInfo & Dockable (bool b=true)

  Dockable() specifies whether a frame can be docked or not.
- wxAuiPaneInfo & Fixed ()

  Fixed() forces a pane to be fixed size so that it cannot be resized.
- wxAuiPaneInfo & Float ()

  Float() indicates that a pane should be floated.
- wxAuiPaneInfo & Floatable (bool b=true)

  Floatable() sets whether the user will be able to undock a pane and turn it into a floating window.
- wxAuiPaneInfo & Gripper (bool visible=true)

  Gripper() indicates that a gripper should be drawn for the pane.
- wxAuiPaneInfo & GripperTop (bool attop=true)

  GripperTop() indicates that a gripper should be drawn at the top of the pane.
- bool HasBorder () const

  HasBorder() returns true if the pane displays a border.
- bool HasCaption () const

  HasCaption() returns true if the pane displays a caption.
- bool HasCloseButton () const

  HasCloseButton() returns true if the pane displays a button to close the pane.
- bool HasFlag (int flag) const

  HasFlag() returns true if the property specified by flag is active for the pane.
- bool HasGripper () const

  HasGripper() returns true if the pane displays a gripper.
- bool HasGripperTop () const

  HasGripper() returns true if the pane displays a gripper at the top.
- bool HasMaximizeButton () const

  HasMaximizeButton() returns true if the pane displays a button to maximize the pane.
- bool HasMinimizeButton () const

  HasMinimizeButton() returns true if the pane displays a button to minimize the pane.
- bool HasPinButton () const

  HasPinButton() returns true if the pane displays a button to float the pane.
- wxAuiPaneInfo & Hide ()

  Hide() indicates that a pane should be hidden.
- wxAuiPaneInfo & Icon (const wxBitmap &b)

  Icon() sets the icon of the pane.
- bool IsBottomDockable () const

  IsBottomDockable() returns true if the pane can be docked at the bottom of the managed frame.
- bool IsDockable () const

  Returns true if the pane can be docked at any side.
- bool IsDocked () const

  IsDocked() returns true if the pane is currently docked.
- bool IsFixed () const

  IsFixed() returns true if the pane cannot be resized.
- bool IsFloatable () const

  IsFloatable() returns true if the pane can be undocked and displayed as a floating window.
- bool IsFloating () const

  IsFloating() returns true if the pane is floating.
- bool IsLeftDockable () const

  IsLeftDockable() returns true if the pane can be docked on the left of the managed frame.
- bool IsMovable () const

  IsMoveable() returns true if the docked frame can be undocked or moved to another dock position.
- bool IsOk () const

IsOk() returns true if the wxAuiPaneInfo structure is valid.

- bool IsResizable () const

  IsResizable() returns true if the pane can be resized.
- bool IsRightDockable () const

  IsRightDockable() returns true if the pane can be docked on the right of the managed frame.
- bool IsShown () const

  IsShown() returns true if the pane is currently shown.
- bool IsToolbar () const

  IsToolbar() returns true if the pane contains a toolbar.
- bool IsTopDockable () const

  IsTopDockable() returns true if the pane can be docked at the top of the managed frame.
- wxAuiPaneInfo & Layer (int layer)

  Layer() determines the layer of the docked pane.
- wxAuiPaneInfo & Left ()

  Left() sets the pane dock position to the left side of the frame.
- wxAuiPaneInfo & LeftDockable (bool b=true)

  LeftDockable() indicates whether a pane can be docked on the left of the frame.
- wxAuiPaneInfo & MaximizeButton (bool visible=true)

  MaximizeButton() indicates that a maximize button should be drawn for the pane.
- wxAuiPaneInfo & MinimizeButton (bool visible=true)

  MinimizeButton() indicates that a minimize button should be drawn for the pane.
- wxAuiPaneInfo & Movable (bool b=true)

  Movable indicates whether a frame can be moved.
- wxAuiPaneInfo & Name (const wxString &n)

  Name() sets the name of the pane so it can be referenced in lookup functions.
- wxAuiPaneInfo & PaneBorder (bool visible=true)

  PaneBorder indicates that a border should be drawn for the pane.
- wxAuiPaneInfo & PinButton (bool visible=true)

  PinButton() indicates that a pin button should be drawn for the pane.
- wxAuiPaneInfo & Position (int pos)

  Position() determines the position of the docked pane.
- wxAuiPaneInfo & Resizable (bool resizable=true)

  Resizable() allows a pane to be resized if the parameter is true, and forces it to be a fixed size if the parameter is false.
- wxAuiPaneInfo & Right ()

  Right() sets the pane dock position to the right side of the frame.
- wxAuiPaneInfo & RightDockable (bool b=true)

  RightDockable() indicates whether a pane can be docked on the right of the frame.
- wxAuiPaneInfo & Row (int row)

  Row() determines the row of the docked pane.
- void SafeSet (wxAuiPaneInfo source)

  Write the safe parts of a newly loaded PaneInfo structure "source" into "this" used on loading perspectives etc.
- wxAuiPaneInfo & SetFlag (int flag, bool option_state)

  SetFlag() turns the property given by flag on or off with the option_state parameter.
- wxAuiPaneInfo & Show (bool show=true)

  Show() indicates that a pane should be shown.
- wxAuiPaneInfo & ToolbarPane ()

  ToolbarPane() specifies that the pane should adopt the default toolbar pane settings.
- wxAuiPaneInfo & Top ()

  Top() sets the pane dock position to the top of the frame.
- wxAuiPaneInfo & TopDockable (bool b=true)

TopDockable() indicates whether a pane can be docked at the top of the frame.

- wxAuiPaneInfo & Window (wxWindow *w)

  Window() assigns the window pointer that the wxAuiPaneInfo should use.
- wxAuiPaneInfo & operator= (const wxAuiPaneInfo &c)

  Makes a copy of the wxAuiPaneInfo object.

- wxAuiPaneInfo & BestSize (const wxSize &size)

  BestSize() sets the ideal size for the pane.
- wxAuiPaneInfo & BestSize (int x, int y)

  BestSize() sets the ideal size for the pane.

- wxAuiPaneInfo & Centre ()

  Center() sets the pane dock position to the left side of the frame.
- wxAuiPaneInfo & Center ()

  Center() sets the pane dock position to the left side of the frame.

- wxAuiPaneInfo & CentrePane ()

  CentrePane() specifies that the pane should adopt the default center pane settings.
- wxAuiPaneInfo & CenterPane ()

  CentrePane() specifies that the pane should adopt the default center pane settings.

- wxAuiPaneInfo & FloatingPosition (const wxPoint &pos)

  FloatingPosition() sets the position of the floating pane.
- wxAuiPaneInfo & FloatingPosition (int x, int y)

  FloatingPosition() sets the position of the floating pane.

- wxAuiPaneInfo & FloatingSize (const wxSize &size)

  FloatingSize() sets the size of the floating pane.
- wxAuiPaneInfo & FloatingSize (int x, int y)

  FloatingSize() sets the size of the floating pane.

- wxAuiPaneInfo & MaxSize (const wxSize &size)

  MaxSize() sets the maximum size of the pane.
- wxAuiPaneInfo & MaxSize (int x, int y)

  MaxSize() sets the maximum size of the pane.

- wxAuiPaneInfo & MinSize (const wxSize &size)

  MinSize() sets the minimum size of the pane.
- wxAuiPaneInfo & MinSize (int x, int y)

  MinSize() sets the minimum size of the pane.

21.45.2 Constructor & Destructor Documentation

wxAuiPaneInfo::wxAuiPaneInfo ( )

wxAuiPaneInfo::wxAuiPaneInfo (const wxAuiPaneInfo & *c*)

Copy constructor.

21.45.3 Member Function Documentation

wxAuiPaneInfo& wxAuiPaneInfo::BestSize (const wxSize & *size*)

BestSize() sets the ideal size for the pane.

The docking manager will attempt to use this size as much as possible when docking or floating the pane.

wxAuiPaneInfo& wxAuiPaneInfo::BestSize (int *x*, int *y*)

BestSize() sets the ideal size for the pane.

The docking manager will attempt to use this size as much as possible when docking or floating the pane.

wxAuiPaneInfo& wxAuiPaneInfo::Bottom ( )

Bottom() sets the pane dock position to the bottom side of the frame.

This is the same thing as calling Direction(wxAUI_DOCK_BOTTOM).

wxAuiPaneInfo& wxAuiPaneInfo::BottomDockable (bool *b* = `true`)

BottomDockable() indicates whether a pane can be docked at the bottom of the frame.

wxAuiPaneInfo& wxAuiPaneInfo::Caption (const wxString & *c*)

Caption() sets the caption of the pane.

wxAuiPaneInfo& wxAuiPaneInfo::CaptionVisible (bool *visible* = `true`)

CaptionVisible indicates that a pane caption should be visible.

If false, no pane caption is drawn.

wxAuiPaneInfo& wxAuiPaneInfo::Center ( )

Center() sets the pane dock position to the left side of the frame.

The centre pane is the space in the middle after all border panes (left, top, right, bottom) are subtracted from the layout. This is the same thing as calling Direction(wxAUI_DOCK_CENTRE).

wxAuiPaneInfo& wxAuiPaneInfo::CenterPane ( )

CentrePane() specifies that the pane should adopt the default center pane settings.

Centre panes usually do not have caption bars. This function provides an easy way of preparing a pane to be displayed in the center dock position.

wxAuiPaneInfo& wxAuiPaneInfo::Centre ( )

Center() sets the pane dock position to the left side of the frame.

The centre pane is the space in the middle after all border panes (left, top, right, bottom) are subtracted from the layout. This is the same thing as calling Direction(wxAUI_DOCK_CENTRE).

wxAuiPaneInfo& wxAuiPaneInfo::CentrePane ( )

CentrePane() specifies that the pane should adopt the default center pane settings.

Centre panes usually do not have caption bars. This function provides an easy way of preparing a pane to be displayed in the center dock position.

wxAuiPaneInfo& wxAuiPaneInfo::CloseButton (bool *visible* = `true`)

CloseButton() indicates that a close button should be drawn for the pane.

wxAuiPaneInfo& wxAuiPaneInfo::DefaultPane ( )

DefaultPane() specifies that the pane should adopt the default pane settings.

wxAuiPaneInfo& wxAuiPaneInfo::DestroyOnClose (bool *b* = `true`)

DestroyOnClose() indicates whether a pane should be destroyed when it is closed.

Normally a pane is simply hidden when the close button is clicked. Setting DestroyOnClose to true will cause the window to be destroyed when the user clicks the pane's close button.

wxAuiPaneInfo& wxAuiPaneInfo::Direction (int *direction*)

Direction() determines the direction of the docked pane.

It is functionally the same as calling Left(), Right(), Top() or Bottom(), except that docking direction may be specified programmatically via the parameter.

wxAuiPaneInfo& wxAuiPaneInfo::Dock ( )

Dock() indicates that a pane should be docked.

It is the opposite of Float().

wxAuiPaneInfo& wxAuiPaneInfo::Dockable (bool *b* = `true`)

Dockable() specifies whether a frame can be docked or not.

It is the same as specifying TopDockable(b).BottomDockable(b).LeftDockable(b).RightDockable(b).

wxAuiPaneInfo& wxAuiPaneInfo::DockFixed (bool *b* = `true`)

DockFixed() causes the containing dock to have no resize sash.

This is useful for creating panes that span the entire width or height of a dock, but should not be resizable in the other direction.

wxAuiPaneInfo& wxAuiPaneInfo::Fixed ( )

Fixed() forces a pane to be fixed size so that it cannot be resized.

After calling Fixed(), IsFixed() will return true.

wxAuiPaneInfo& wxAuiPaneInfo::Float ( )

Float() indicates that a pane should be floated.

It is the opposite of Dock().

wxAuiPaneInfo& wxAuiPaneInfo::Floatable (bool *b* = `true`)

Floatable() sets whether the user will be able to undock a pane and turn it into a floating window.

wxAuiPaneInfo& wxAuiPaneInfo::FloatingPosition (const wxPoint & *pos*)

FloatingPosition() sets the position of the floating pane.

wxAuiPaneInfo& wxAuiPaneInfo::FloatingPosition (int *x*, int *y*)

FloatingPosition() sets the position of the floating pane.

wxAuiPaneInfo& wxAuiPaneInfo::FloatingSize (const wxSize & *size*)

FloatingSize() sets the size of the floating pane.

wxAuiPaneInfo& wxAuiPaneInfo::FloatingSize (int *x*, int *y*)

FloatingSize() sets the size of the floating pane.

wxAuiPaneInfo& wxAuiPaneInfo::Gripper (bool *visible* = `true`)

Gripper() indicates that a gripper should be drawn for the pane.

wxAuiPaneInfo& wxAuiPaneInfo::GripperTop (bool *attop* = `true`)

GripperTop() indicates that a gripper should be drawn at the top of the pane.

bool wxAuiPaneInfo::HasBorder ( ) const

HasBorder() returns true if the pane displays a border.

bool wxAuiPaneInfo::HasCaption ( ) const

HasCaption() returns true if the pane displays a caption.

bool wxAuiPaneInfo::HasCloseButton ( ) const

HasCloseButton() returns true if the pane displays a button to close the pane.

bool wxAuiPaneInfo::HasFlag (int *flag*) const

HasFlag() returns true if the property specified by flag is active for the pane.

bool wxAuiPaneInfo::HasGripper ( ) const

HasGripper() returns true if the pane displays a gripper.

bool wxAuiPaneInfo::HasGripperTop ( ) const

HasGripper() returns true if the pane displays a gripper at the top.

bool wxAuiPaneInfo::HasMaximizeButton ( ) const

HasMaximizeButton() returns true if the pane displays a button to maximize the pane.

bool wxAuiPaneInfo::HasMinimizeButton ( ) const

HasMinimizeButton() returns true if the pane displays a button to minimize the pane.

bool wxAuiPaneInfo::HasPinButton ( ) const

HasPinButton() returns true if the pane displays a button to float the pane.

wxAuiPaneInfo& wxAuiPaneInfo::Hide ( )

Hide() indicates that a pane should be hidden.

wxAuiPaneInfo& wxAuiPaneInfo::Icon (const wxBitmap & *b*)

Icon() sets the icon of the pane.

Notice that the height of the icon should be smaller than the value returned by wxAuiDockArt::GetMetric(wxAUI_↩ DOCKART_CAPTION_SIZE) to ensure that it appears correctly.

Since

2.9.2

bool wxAuiPaneInfo::IsBottomDockable ( ) const

IsBottomDockable() returns true if the pane can be docked at the bottom of the managed frame.

See also

IsDockable()

bool wxAuiPaneInfo::IsDockable ( ) const

Returns true if the pane can be docked at any side.

See also

IsTopDockable(), IsBottomDockable(), IsLeftDockable(), IsRightDockable()

Since

2.9.2

bool wxAuiPaneInfo::IsDocked ( ) const

IsDocked() returns true if the pane is currently docked.

bool wxAuiPaneInfo::IsFixed ( ) const

IsFixed() returns true if the pane cannot be resized.

bool wxAuiPaneInfo::IsFloatable ( ) const

IsFloatable() returns true if the pane can be undocked and displayed as a floating window.

bool wxAuiPaneInfo::IsFloating ( ) const

IsFloating() returns true if the pane is floating.

bool wxAuiPaneInfo::IsLeftDockable ( ) const

IsLeftDockable() returns true if the pane can be docked on the left of the managed frame.

See also

IsDockable()

bool wxAuiPaneInfo::IsMovable ( ) const

IsMoveable() returns true if the docked frame can be undocked or moved to another dock position.

bool wxAuiPaneInfo::IsOk ( ) const

IsOk() returns true if the wxAuiPaneInfo structure is valid.

A pane structure is valid if it has an associated window.

bool wxAuiPaneInfo::IsResizable ( ) const

IsResizable() returns true if the pane can be resized.

bool wxAuiPaneInfo::IsRightDockable ( ) const

IsRightDockable() returns true if the pane can be docked on the right of the managed frame.

See also

IsDockable()

bool wxAuiPaneInfo::IsShown ( ) const

IsShown() returns true if the pane is currently shown.

bool wxAuiPaneInfo::IsToolbar ( ) const

IsToolbar() returns true if the pane contains a toolbar.

bool wxAuiPaneInfo::IsTopDockable ( ) const

IsTopDockable() returns true if the pane can be docked at the top of the managed frame.

See also

IsDockable()

wxAuiPaneInfo& wxAuiPaneInfo::Layer (int *layer*)

Layer() determines the layer of the docked pane.

The dock layer is similar to an onion, the inner-most layer being layer 0. Each shell moving in the outward direction has a higher layer number. This allows for more complex docking layout formation.

wxAuiPaneInfo& wxAuiPaneInfo::Left ( )

Left() sets the pane dock position to the left side of the frame.

This is the same thing as calling Direction(wxAUI_DOCK_LEFT).

wxAuiPaneInfo& wxAuiPaneInfo::LeftDockable (bool *b* = `true`)

LeftDockable() indicates whether a pane can be docked on the left of the frame.

wxAuiPaneInfo& wxAuiPaneInfo::MaximizeButton (bool *visible* = `true`)

MaximizeButton() indicates that a maximize button should be drawn for the pane.

wxAuiPaneInfo& wxAuiPaneInfo::MaxSize (const wxSize & *size*)

MaxSize() sets the maximum size of the pane.

wxAuiPaneInfo& wxAuiPaneInfo::MaxSize (int *x*, int *y*)

MaxSize() sets the maximum size of the pane.

wxAuiPaneInfo& wxAuiPaneInfo::MinimizeButton (bool *visible* = `true`)

MinimizeButton() indicates that a minimize button should be drawn for the pane.

wxAuiPaneInfo& wxAuiPaneInfo::MinSize (const wxSize & *size*)

MinSize() sets the minimum size of the pane.

Please note that this is only partially supported as of this writing.

wxAuiPaneInfo& wxAuiPaneInfo::MinSize (int *x,* int *y*)

MinSize() sets the minimum size of the pane.

Please note that this is only partially supported as of this writing.

wxAuiPaneInfo& wxAuiPaneInfo::Movable (bool *b* = `true`)

Movable indicates whether a frame can be moved.

wxAuiPaneInfo& wxAuiPaneInfo::Name (const wxString & *n*)

Name() sets the name of the pane so it can be referenced in lookup functions.

If a name is not specified by the user, a random name is assigned to the pane when it is added to the manager.

wxAuiPaneInfo& wxAuiPaneInfo::operator= (const wxAuiPaneInfo & *c*)

Makes a copy of the wxAuiPaneInfo object.

wxAuiPaneInfo& wxAuiPaneInfo::PaneBorder (bool *visible* = `true`)

PaneBorder indicates that a border should be drawn for the pane.

wxAuiPaneInfo& wxAuiPaneInfo::PinButton (bool *visible* = `true`)

PinButton() indicates that a pin button should be drawn for the pane.

wxAuiPaneInfo& wxAuiPaneInfo::Position (int *pos*)

Position() determines the position of the docked pane.

wxAuiPaneInfo& wxAuiPaneInfo::Resizable (bool *resizable* = `true`)

Resizable() allows a pane to be resized if the parameter is true, and forces it to be a fixed size if the parameter is false.

This is simply an antonym for Fixed().

wxAuiPaneInfo& wxAuiPaneInfo::Right ( )

Right() sets the pane dock position to the right side of the frame.

wxAuiPaneInfo& wxAuiPaneInfo::RightDockable (bool *b* = `true`)

RightDockable() indicates whether a pane can be docked on the right of the frame.

wxAuiPaneInfo& wxAuiPaneInfo::Row (int *row*)

Row() determines the row of the docked pane.

void wxAuiPaneInfo::SafeSet (wxAuiPaneInfo *source*)

Write the safe parts of a newly loaded PaneInfo structure "source" into "this" used on loading perspectives etc.

wxAuiPaneInfo& wxAuiPaneInfo::SetFlag (int *flag*, bool *option_state*)

SetFlag() turns the property given by flag on or off with the option_state parameter.

wxAuiPaneInfo& wxAuiPaneInfo::Show (bool *show* = `true`)

Show() indicates that a pane should be shown.

wxAuiPaneInfo& wxAuiPaneInfo::ToolbarPane ( )

ToolbarPane() specifies that the pane should adopt the default toolbar pane settings.

wxAuiPaneInfo& wxAuiPaneInfo::Top ( )

Top() sets the pane dock position to the top of the frame.

wxAuiPaneInfo& wxAuiPaneInfo::TopDockable (bool *b* = `true`)

TopDockable() indicates whether a pane can be docked at the top of the frame.

wxAuiPaneInfo& wxAuiPaneInfo::Window (wxWindow * *w*)

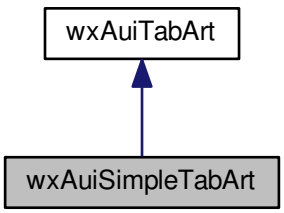
Window() assigns the window pointer that the wxAuiPaneInfo should use.

This normally does not need to be specified, as the window pointer is automatically assigned to the wxAuiPaneInfo structure as soon as it is added to the manager.

21.46 wxAuiSimpleTabArt Class Reference

`#include <wx/aui/auibook.h>`

Inheritance diagram for wxAuiSimpleTabArt:



21.46.1 Detailed Description

Another standard tab art provider for wxAuiNotebook.

wxAuiSimpleTabArt is derived from wxAuiTabArt demonstrating how to write a completely new tab art class. It can also be used as alternative to wxAuiDefaultTabArt.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiSimpleTabArt ()
- virtual ~wxAuiSimpleTabArt ()
- wxAuiTabArt * Clone ()

  Clones the art object.
- void SetFlags (unsigned int flags)

  Sets flags.
- void SetSizingInfo (const wxSize &tabCtrlSize, size_t tabCount)

  Sets sizing information.
- void SetNormalFont (const wxFont &font)

  Sets the normal font for drawing labels.
- void SetSelectedFont (const wxFont &font)

  Sets the font for drawing text for selected UI elements.
- void SetMeasuringFont (const wxFont &font)

  Sets the font used for calculating measurements.
- void SetColour (const wxColour &colour)

  Sets the colour of the inactive tabs.
- void SetActiveColour (const wxColour &colour)

  Sets the colour of the selected tab.
- void DrawBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)

  Draws a background on the given area.
- void DrawTab (wxDC &dc, wxWindow *wnd, const wxAuiNotebookPage &pane, const wxRect &inRect, int closeButtonState, wxRect *outTabRect, wxRect *outButtonRect, int *xExtent)

  Draws a tab.
- void DrawButton (wxDC &dc, wxWindow *wnd, const wxRect &inRect, int bitmapId, int buttonState, int orientation, wxRect *outRect)

  Draws a button.
- int GetIndentSize ()

  Returns the indent size.
- wxSize GetTabSize (wxDC &dc, wxWindow *wnd, const wxString &caption, const wxBitmap &bitmap, bool active, int closeButtonState, int *xExtent)

  Returns the tab size for the given caption, bitmap and state.
- int ShowDropDown (wxWindow *wnd, const wxAuiNotebookPageArray &items, int activeIdx)
- int GetBestTabCtrlSize (wxWindow *wnd, const wxAuiNotebookPageArray &pages, const wxSize &requiredBmpSize)

  Returns the tab control size.

Protected Attributes

- wxFont m_normalFont
- wxFont m_selectedFont
- wxFont m_measuringFont
- wxPen m_normalBkPen
- wxPen m_selectedBkPen
- wxBrush m_normalBkBrush
- wxBrush m_selectedBkBrush
- wxBrush m_bkBrush
- wxBitmap m_activeCloseBmp
- wxBitmap m_disabledCloseBmp
- wxBitmap m_activeLeftBmp
- wxBitmap m_disabledLeftBmp
- wxBitmap m_activeRightBmp
- wxBitmap m_disabledRightBmp
- wxBitmap m_activeWindowListBmp
- wxBitmap m_disabledWindowListBmp
- int m_fixedTabWidth
- unsigned int m_flags

21.46.2 Constructor & Destructor Documentation

wxAuiSimpleTabArt::wxAuiSimpleTabArt ( )

virtual wxAuiSimpleTabArt::~wxAuiSimpleTabArt ( ) [virtual]

21.46.3 Member Function Documentation

wxAuiTabArt* wxAuiSimpleTabArt::Clone ( ) [virtual]

Clones the art object.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::DrawBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) [virtual]

Draws a background on the given area.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::DrawButton (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *in_rect*, int *bitmap_id*, int *button_state*, int *orientation*, wxRect * *out_rect*) [virtual]

Draws a button.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::DrawTab (wxDC & *dc*, wxWindow * *wnd*, const wxAuiNotebookPage & *page*, const wxRect & *rect*, int *close_button_state*, wxRect * *out_tab_rect*, wxRect * *out_button_rect*, int * *x_extent*) [virtual]

Draws a tab.

Implements wxAuiTabArt.

int wxAuiSimpleTabArt::GetBestTabCtrlSize (wxWindow * , const wxAuiNotebookPageArray & , const wxSize &) [virtual]

Returns the tab control size.

Implements wxAuiTabArt.

int wxAuiSimpleTabArt::GetIndentSize ( ) [virtual]

Returns the indent size.

Implements wxAuiTabArt.

wxSize wxAuiSimpleTabArt::GetTabSize (wxDC & *dc,* wxWindow * *wnd,* const wxString & *caption,* const wxBitmap & *bitmap,* bool *active,* int *close_button_state,* int * *x_extent*) [virtual]

Returns the tab size for the given caption, bitmap and state.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetActiveColour (const wxColour & *colour*) [virtual]

Sets the colour of the selected tab.

Since

2.9.2

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetColour (const wxColour & *colour*) [virtual]

Sets the colour of the inactive tabs.

Since

2.9.2

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetFlags (unsigned int *flags*) [virtual]

Sets flags.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetMeasuringFont (const wxFont & *font*) [virtual]

Sets the font used for calculating measurements.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetNormalFont (const wxFont & *font*) [virtual]

Sets the normal font for drawing labels.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetSelectedFont (const wxFont & *font*) `[virtual]`

Sets the font for drawing text for selected UI elements.

Implements wxAuiTabArt.

void wxAuiSimpleTabArt::SetSizingInfo (const wxSize & *tab_ctrl_size,* size_t *tab_count*) `[virtual]`

Sets sizing information.

Implements wxAuiTabArt.

int wxAuiSimpleTabArt::ShowDropDown (wxWindow * *wnd,* const wxAuiNotebookPageArray & *items,* int *activeIdx*)

21.46.4 Member Data Documentation

wxBitmap wxAuiSimpleTabArt::m_activeCloseBmp `[protected]`

wxBitmap wxAuiSimpleTabArt::m_activeLeftBmp `[protected]`

wxBitmap wxAuiSimpleTabArt::m_activeRightBmp `[protected]`

wxBitmap wxAuiSimpleTabArt::m_activeWindowListBmp `[protected]`

wxBrush wxAuiSimpleTabArt::m_bkBrush `[protected]`

wxBitmap wxAuiSimpleTabArt::m_disabledCloseBmp `[protected]`

wxBitmap wxAuiSimpleTabArt::m_disabledLeftBmp `[protected]`

wxBitmap wxAuiSimpleTabArt::m_disabledRightBmp `[protected]`

wxBitmap wxAuiSimpleTabArt::m_disabledWindowListBmp `[protected]`

int wxAuiSimpleTabArt::m_fixedTabWidth `[protected]`

unsigned int wxAuiSimpleTabArt::m_flags `[protected]`

wxFont wxAuiSimpleTabArt::m_measuringFont `[protected]`

wxBrush wxAuiSimpleTabArt::m_normalBkBrush `[protected]`

wxPen wxAuiSimpleTabArt::m_normalBkPen `[protected]`

wxFont wxAuiSimpleTabArt::m_normalFont `[protected]`

wxBrush wxAuiSimpleTabArt::m_selectedBkBrush `[protected]`

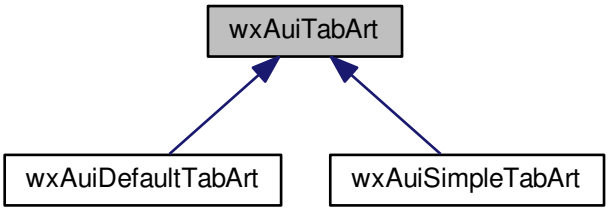
wxPen wxAuiSimpleTabArt::m_selectedBkPen `[protected]`

wxFont wxAuiSimpleTabArt::m_selectedFont `[protected]`

21.47 wxAuiTabArt Class Reference

`#include <wx/aui/auibook.h>`

Inheritance diagram for wxAuiTabArt:



21.47.1 Detailed Description

Tab art provider defines all the drawing functions used by wxAuiNotebook.

This allows the wxAuiNotebook to have a pluggable look-and-feel.

By default, a wxAuiNotebook uses an instance of this class called wxAuiDefaultTabArt which provides bitmap art and a colour scheme that is adapted to the major platforms' look. You can either derive from that class to alter its behaviour or write a completely new tab art class.

Another example of creating a new wxAuiNotebook tab bar is wxAuiSimpleTabArt.

Call wxAuiNotebook::SetArtProvider() to make use of this new tab art.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiTabArt ()

  Constructor.
- virtual wxAuiTabArt * Clone ()=0

  Clones the art object.
- virtual void DrawBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0

  Draws a background on the given area.
- virtual void DrawButton (wxDC &dc, wxWindow *wnd, const wxRect &in_rect, int bitmap_id, int button_state, int orientation, wxRect *out_rect)=0

  Draws a button.
- virtual void DrawTab (wxDC &dc, wxWindow *wnd, const wxAuiNotebookPage &page, const wxRect &rect, int close_button_state, wxRect *out_tab_rect, wxRect *out_button_rect, int *x_extent)=0

  Draws a tab.
- virtual int GetBestTabCtrlSize (wxWindow *, const wxAuiNotebookPageArray &, const wxSize &)=0

  Returns the tab control size.
- virtual int GetIndentSize ()=0

  Returns the indent size.

- virtual wxSize GetTabSize (wxDC &dc, wxWindow *wnd, const wxString &caption, const wxBitmap &bitmap, bool active, int close_button_state, int *x_extent)=0

  Returns the tab size for the given caption, bitmap and state.
- virtual void SetFlags (unsigned int flags)=0

  Sets flags.
- virtual void SetMeasuringFont (const wxFont &font)=0

  Sets the font used for calculating measurements.
- virtual void SetNormalFont (const wxFont &font)=0

  Sets the normal font for drawing labels.
- virtual void SetSelectedFont (const wxFont &font)=0

  Sets the font for drawing text for selected UI elements.
- virtual void SetColour (const wxColour &colour)=0

  Sets the colour of the inactive tabs.
- virtual void SetActiveColour (const wxColour &colour)=0

  Sets the colour of the selected tab.
- virtual void SetSizingInfo (const wxSize &tab_ctrl_size, size_t tab_count)=0

  Sets sizing information.

21.47.2 Constructor & Destructor Documentation

wxAuiTabArt::wxAuiTabArt ( )

Constructor.

21.47.3 Member Function Documentation

virtual wxAuiTabArt* wxAuiTabArt::Clone ( ) `[pure virtual]`

Clones the art object.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::DrawBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) `[pure virtual]`

Draws a background on the given area.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::DrawButton (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *in_rect*, int *bitmap_id*, int *button_state*, int *orientation*, wxRect * *out_rect*) `[pure virtual]`

Draws a button.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::DrawTab (wxDC & *dc*, wxWindow * *wnd*, const wxAuiNotebookPage & *page*, const wxRect & *rect*, int *close_button_state*, wxRect * *out_tab_rect*, wxRect * *out_button_rect*, int * *x_extent*) `[pure virtual]`

Draws a tab.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual int wxAuiTabArt::GetBestTabCtrlSize (wxWindow * , const wxAuiNotebookPageArray & , const wxSize &) `[pure virtual]`

Returns the tab control size.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual int wxAuiTabArt::GetIndentSize ( ) `[pure virtual]`

Returns the indent size.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual wxSize wxAuiTabArt::GetTabSize (wxDC & *dc,* wxWindow * *wnd,* const wxString & *caption,* const wxBitmap & *bitmap,* bool *active,* int *close_button_state,* int * *x_extent*) `[pure virtual]`

Returns the tab size for the given caption, bitmap and state.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetActiveColour (const wxColour & *colour*) `[pure virtual]`

Sets the colour of the selected tab.

Since

2.9.2

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetColour (const wxColour & *colour*) `[pure virtual]`

Sets the colour of the inactive tabs.

Since

2.9.2

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetFlags (unsigned int *flags*) `[pure virtual]`

Sets flags.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetMeasuringFont (const wxFont & *font*) `[pure virtual]`

Sets the font used for calculating measurements.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetNormalFont (const wxFont & *font*) `[pure virtual]`

Sets the normal font for drawing labels.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetSelectedFont (const wxFont & *font*) `[pure virtual]`

Sets the font for drawing text for selected UI elements.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

virtual void wxAuiTabArt::SetSizingInfo (const wxSize & *tab_ctrl_size,* size_t *tab_count*) `[pure virtual]`

Sets sizing information.

Implemented in wxAuiSimpleTabArt, and wxAuiDefaultTabArt.

21.48 wxAuiTabContainer Class Reference

```
#include <wx/aui/auibook.h>
```

21.48.1 Detailed Description

wxAuiTabContainer is a class which contains information about each tab.

It also can render an entire tab control to a specified DC. It's not a window class itself, because this code will be used by the wxAuiNotebook, where it is disadvantageous to have separate windows for each tab control in the case of "docked tabs".

A derived class, wxAuiTabCtrl, is an actual wxWindow - derived window which can be used as a tab control in the normal sense.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiTabContainer ()
  Default ctor.
- virtual ~wxAuiTabContainer ()
  Default dtor.
- void SetArtProvider (wxAuiTabArt *art)
- wxAuiTabArt * GetArtProvider () const
- void SetFlags (unsigned int flags)
- unsigned int GetFlags () const
- bool AddPage (wxWindow *page, const wxAuiNotebookPage &info)
- bool InsertPage (wxWindow *page, const wxAuiNotebookPage &info, size_t idx)
- bool MovePage (wxWindow *page, size_t newIdx)
- bool RemovePage (wxWindow *page)
- bool SetActivePage (wxWindow *page)
- bool SetActivePage (size_t page)
- void SetNoneActive ()
- int GetActivePage () const
- bool TabHitTest (int x, int y, wxWindow **hit) const
- bool ButtonHitTest (int x, int y, wxAuiTabContainerButton **hit) const
- wxWindow * GetWindowFromIdx (size_t idx) const

- int GetIdxFromWindow (wxWindow *page) const
- size_t GetPageCount () const
- wxAuiNotebookPage & GetPage (size_t idx)
- const wxAuiNotebookPage & GetPage (size_t idx) const
- wxAuiNotebookPageArray & GetPages ()
- void SetNormalFont (const wxFont &normalFont)
- void SetSelectedFont (const wxFont &selectedFont)
- void SetMeasuringFont (const wxFont &measuringFont)
- void SetColour (const wxColour &colour)
- void SetActiveColour (const wxColour &colour)
- void DoShowHide ()
- void SetRect (const wxRect &rect)
- void RemoveButton (int id)
- void AddButton (int id, int location, const wxBitmap &normalBitmap=wxNullBitmap, const wxBitmap &disabledBitmap=wxNullBitmap)
- size_t GetTabOffset () const
- void SetTabOffset (size_t offset)
- bool IsTabVisible (int tabPage, int tabOffset, wxDC *dc, wxWindow *wnd)
- void MakeTabVisible (int tabPage, wxWindow *win)

Protected Member Functions

- virtual void Render (wxDC *dc, wxWindow *wnd)

Protected Attributes

- wxAuiTabArt * m_art
- wxAuiNotebookPageArray m_pages
- wxAuiTabContainerButtonArray m_buttons
- wxAuiTabContainerButtonArray m_tabCloseButtons
- wxRect m_rect
- size_t m_tabOffset
- unsigned int m_flags

21.48.2 Constructor & Destructor Documentation

wxAuiTabContainer::wxAuiTabContainer ( )

Default ctor.

virtual wxAuiTabContainer::~wxAuiTabContainer ( ) [virtual]

Default dtor.

21.48.3 Member Function Documentation

void wxAuiTabContainer::AddButton (int *id,* int *location,* const wxBitmap & *normalBitmap* = wxNullBitmap, const wxBitmap & *disabledBitmap* = wxNullBitmap)

bool wxAuiTabContainer::AddPage (wxWindow * *page,* const wxAuiNotebookPage & *info*)

bool wxAuiTabContainer::ButtonHitTest (int *x,* int *y,* wxAuiTabContainerButton ** *hit*) const

void wxAuiTabContainer::DoShowHide ( )

int wxAuiTabContainer::GetActivePage ( ) const

wxAuiTabArt* wxAuiTabContainer::GetArtProvider ( ) const

unsigned int wxAuiTabContainer::GetFlags ( ) const

int wxAuiTabContainer::GetIdxFromWindow (**wxWindow** * *page*) const

wxAuiNotebookPage& wxAuiTabContainer::GetPage (size_t *idx*)

const wxAuiNotebookPage& wxAuiTabContainer::GetPage (size_t *idx*) const

size_t wxAuiTabContainer::GetPageCount ( ) const

wxAuiNotebookPageArray& wxAuiTabContainer::GetPages ( )

size_t wxAuiTabContainer::GetTabOffset ( ) const

wxWindow* wxAuiTabContainer::GetWindowFromIdx (size_t *idx*) const

bool wxAuiTabContainer::InsertPage (**wxWindow** * *page,* const wxAuiNotebookPage & *info,* size_t *idx*)

bool wxAuiTabContainer::IsTabVisible (int *tabPage,* int *tabOffset,* **wxDC** * *dc,* **wxWindow** * *wnd*)

void wxAuiTabContainer::MakeTabVisible (int *tabPage,* **wxWindow** * *win*)

bool wxAuiTabContainer::MovePage (**wxWindow** * *page,* size_t *newIdx*)

void wxAuiTabContainer::RemoveButton (int *id*)

bool wxAuiTabContainer::RemovePage (**wxWindow** * *page*)

virtual void wxAuiTabContainer::Render (**wxDC** * *dc,* **wxWindow** * *wnd*) [protected], [virtual]

void wxAuiTabContainer::SetActiveColour (const **wxColour** & *colour*)

bool wxAuiTabContainer::SetActivePage (**wxWindow** * *page*)

bool wxAuiTabContainer::SetActivePage (size_t *page*)

void wxAuiTabContainer::SetArtProvider (**wxAuiTabArt** * *art*)

void wxAuiTabContainer::SetColour (const **wxColour** & *colour*)

void wxAuiTabContainer::SetFlags (unsigned int *flags*)

void wxAuiTabContainer::SetMeasuringFont (const **wxFont** & *measuringFont*)

void wxAuiTabContainer::SetNoneActive ( )

void wxAuiTabContainer::SetNormalFont (const **wxFont** & *normalFont*)

void wxAuiTabContainer::SetRect (const **wxRect** & *rect*)

void wxAuiTabContainer::SetSelectedFont (const wxFont & *selectedFont*)

void wxAuiTabContainer::SetTabOffset (size_t *offset*)

bool wxAuiTabContainer::TabHitTest (int *x*, int *y*, wxWindow ** *hit*) const

21.48.4 Member Data Documentation

wxAuiTabArt* wxAuiTabContainer::m_art [protected]

wxAuiTabContainerButtonArray wxAuiTabContainer::m_buttons [protected]

unsigned int wxAuiTabContainer::m_flags [protected]

wxAuiNotebookPageArray wxAuiTabContainer::m_pages [protected]

wxRect wxAuiTabContainer::m_rect [protected]

wxAuiTabContainerButtonArray wxAuiTabContainer::m_tabCloseButtons [protected]

size_t wxAuiTabContainer::m_tabOffset [protected]

21.49 wxAuiTabContainerButton Class Reference

`#include <wx/aui/auibook.h>`

21.49.1 Detailed Description

A simple class which holds information about wxAuiNotebook tab buttons and their state.

Library: wxAui

Category: Window Docking (wxAUI)

Public Attributes

- int id

  button's id
- int curState

  current state (normal, hover, pressed, etc.)
- int location

  buttons location (wxLEFT, wxRIGHT, or wxCENTER)
- wxBitmap bitmap

  button's hover bitmap
- wxBitmap disBitmap

  button's disabled bitmap
- wxRect rect

  button's hit rectangle

21.49.2 Member Data Documentation

wxBitmap wxAuiTabContainerButton::bitmap

button's hover bitmap

int wxAuiTabContainerButton::curState

current state (normal, hover, pressed, etc.)

wxBitmap wxAuiTabContainerButton::disBitmap

button's disabled bitmap

int wxAuiTabContainerButton::id

button's id

int wxAuiTabContainerButton::location

buttons location (wxLEFT, wxRIGHT, or wxCENTER)

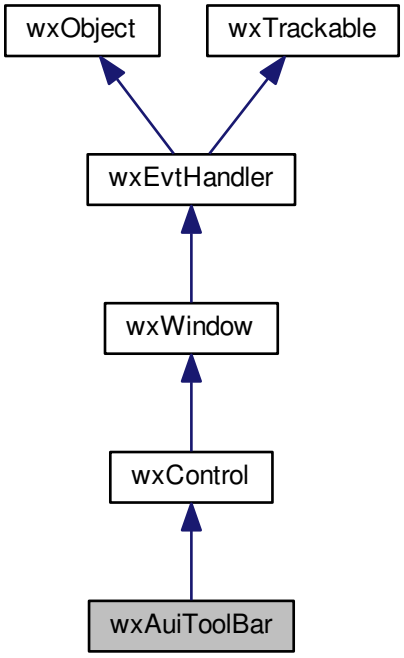
wxRect wxAuiTabContainerButton::rect

button's hit rectangle

21.50 wxAuiToolBar Class Reference

```
#include <wx/aui/auibar.h>
```

Inheritance diagram for wxAuiToolBar:



21.50.1 Detailed Description

wxAuiToolBar is a dockable toolbar, part of the wxAUI class framework.

See also wxAUI Overview.

Styles

This class supports the following styles:

- wxAUI_TB_TEXT:
- wxAUI_TB_NO_TOOLTIPS:
- wxAUI_TB_NO_AUTORESIZE:
- wxAUI_TB_GRIPPER:
- wxAUI_TB_OVERFLOW:
- wxAUI_TB_VERTICAL: using this style forces the toolbar to be vertical and be only dockable to the left or right sides of the window whereas by default it can be horizontal or vertical and be docked anywhere
- wxAUI_TB_HORZ_LAYOUT:
- wxAUI_TB_HORIZONTAL: analogous to wxAUI_TB_VERTICAL, but forces the toolbar to be horizontal
- wxAUI_TB_PLAIN_BACKGROUND: Draw a plain background (based on parent) instead of the default gradient background.

- wxAUI_TB_HORZ_TEXT: Equivalent to wxAUI_TB_HORZ_LAYOUT | wxAUI_TB_TEXT
- wxAUI_TB_DEFAULT_STYLE: The default is to have no styles

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxAuiToolBarEvent& event)

Event macros for events emitted by this class:

- EVT_AUITOOLBAR_TOOL_DROPDOWN(id, func): Process a wxEVT_AUITOOLBAR_TOOL_DROPDOWN event
- EVT_AUITOOLBAR_OVERFLOW_CLICK(id, func): Process a wxEVT_AUITOOLBAR_OVERFLOW_CLICK event
- EVT_AUITOOLBAR_RIGHT_CLICK(id, func): Process a wxEVT_AUITOOLBAR_RIGHT_CLICK event
- EVT_AUITOOLBAR_MIDDLE_CLICK(id, func): Process a wxEVT_AUITOOLBAR_MIDDLE_CLICK event
- EVT_AUITOOLBAR_BEGIN_DRAG(id, func): Process a wxEVT_AUITOOLBAR_BEGIN_DRAG event

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiToolBar ()

  Default constructor, use Create() later.
- wxAuiToolBar (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &position=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxAUI_TB_DEFAULT_STYLE)

  Constructor creating and initializing the object.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxAUI_TB_DEFAULT_STYLE)

  Really create wxAuiToolBar created using default constructor.
- virtual ~wxAuiToolBar ()
- void SetWindowStyleFlag (long style)

  Sets the style of the window.
- long GetWindowStyleFlag () const

  Gets the window style that was passed to the constructor or Create() method.
- void SetArtProvider (wxAuiToolBarArt *art)
- wxAuiToolBarArt * GetArtProvider () const
- bool SetFont (const wxFont &font)

  Sets the font for this window.
- wxAuiToolBarItem * AddTool (int tool_id, const wxString &label, const wxBitmap &bitmap, const wxString &short_help_string=wxEmptyString, wxItemKind kind=wxITEM_NORMAL)
- wxAuiToolBarItem * AddTool (int tool_id, const wxString &label, const wxBitmap &bitmap, const wxBitmap &disabled_bitmap, wxItemKind kind, const wxString &short_help_string, const wxString &long_help_string, wxObject *client_data)
- wxAuiToolBarItem * AddTool (int tool_id, const wxBitmap &bitmap, const wxBitmap &disabled_bitmap, bool toggle=false, wxObject *client_data=NULL, const wxString &short_help_string=wxEmptyString, const wxString &long_help_string=wxEmptyString)

- wxAuiToolBarItem * AddLabel (int tool_id, const wxString &label=wxEmptyString, const int width=-1)
- wxAuiToolBarItem * AddControl (wxControl *control, const wxString &label=wxEmptyString)
- wxAuiToolBarItem * AddSeparator ()
- wxAuiToolBarItem * AddSpacer (int pixels)
- wxAuiToolBarItem * AddStretchSpacer (int proportion=1)
- bool Realize ()
- wxControl * FindControl (int window_id)
- wxAuiToolBarItem * FindToolByPosition (wxCoord x, wxCoord y) const
- wxAuiToolBarItem * FindToolByIndex (int idx) const
- wxAuiToolBarItem * FindTool (int tool_id) const
- void ClearTools ()
- void Clear ()
- bool DeleteTool (int tool_id)
- bool DeleteByIndex (int tool_id)
- size_t GetToolCount () const
- int GetToolPos (int tool_id) const
- int GetToolIndex (int tool_id) const
- bool GetToolFits (int tool_id) const
- wxRect GetToolRect (int tool_id) const
- bool GetToolFitsByIndex (int tool_id) const
- bool GetToolBarFits () const
- void SetMargins (const wxSize &size)
- void SetMargins (int x, int y)
- void SetMargins (int left, int right, int top, int bottom)
- void SetToolBitmapSize (const wxSize &size)
- wxSize GetToolBitmapSize () const
- bool GetOverflowVisible () const
- void SetOverflowVisible (bool visible)
- bool GetGripperVisible () const
- void SetGripperVisible (bool visible)
- void ToggleTool (int tool_id, bool state)
- bool GetToolToggled (int tool_id) const
- void EnableTool (int tool_id, bool state)
- bool GetToolEnabled (int tool_id) const
- void SetToolDropDown (int tool_id, bool dropdown)

  Set whether the specified toolbar item has a drop down button.
- bool GetToolDropDown (int tool_id) const

  Returns whether the specified toolbar item has an associated drop down button.
- void SetToolBorderPadding (int padding)
- int GetToolBorderPadding () const
- void SetToolTextOrientation (int orientation)
- int GetToolTextOrientation () const
- void SetToolPacking (int packing)
- int GetToolPacking () const
- void SetToolProportion (int tool_id, int proportion)
- int GetToolProportion (int tool_id) const
- void SetToolSeparation (int separation)
- int GetToolSeparation () const
- void SetToolSticky (int tool_id, bool sticky)
- bool GetToolSticky (int tool_id) const
- wxString GetToolLabel (int tool_id) const
- void SetToolLabel (int tool_id, const wxString &label)
- wxBitmap GetToolBitmap (int tool_id) const
- void SetToolBitmap (int tool_id, const wxBitmap &bitmap)

- wxString GetToolShortHelp (int tool_id) const
- void SetToolShortHelp (int tool_id, const wxString &help_string)
- wxString GetToolLongHelp (int tool_id) const
- void SetToolLongHelp (int tool_id, const wxString &help_string)
- void SetCustomOverflowItems (const wxAuiToolBarItemArray &prepend, const wxAuiToolBarItemArray &append)
- wxSize GetHintSize (int dock_direction) const

  get size of hint rectangle for a particular dock location
- bool IsPaneValid (const wxAuiPaneInfo &pane) const

Additional Inherited Members

21.50.2 Constructor & Destructor Documentation

wxAuiToolBar::wxAuiToolBar ( )

Default constructor, use Create() later.

Since

2.9.5

wxAuiToolBar::wxAuiToolBar (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *position* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxAUI_TB_DEFAULT_STYLE)

Constructor creating and initializing the object.

virtual wxAuiToolBar::~wxAuiToolBar ( ) [virtual]

21.50.3 Member Function Documentation

wxAuiToolBarItem* wxAuiToolBar::AddControl (wxControl * *control*, const wxString & *label* = wxEmptyString)

wxAuiToolBarItem* wxAuiToolBar::AddLabel (int *tool_id*, const wxString & *label* = wxEmptyString, const int *width* = -1)

wxAuiToolBarItem* wxAuiToolBar::AddSeparator ( )

wxAuiToolBarItem* wxAuiToolBar::AddSpacer (int *pixels*)

wxAuiToolBarItem* wxAuiToolBar::AddStretchSpacer (int *proportion* = 1)

wxAuiToolBarItem* wxAuiToolBar::AddTool (int *tool_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxString & *short_help_string* = wxEmptyString, wxItemKind *kind* = wxITEM_NORMAL)

wxAuiToolBarItem* wxAuiToolBar::AddTool (int *tool_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxBitmap & *disabled_bitmap*, wxItemKind *kind*, const wxString & *short_help_string*, const wxString & *long_help_string*, wxObject * *client_data*)

wxAuiToolBarItem* wxAuiToolBar::AddTool (int *tool_id*, const wxBitmap & *bitmap*, const wxBitmap & *disabled_bitmap*, bool *toggle* = false, wxObject * *client_data* = NULL, const wxString & *short_help_string* = wxEmptyString, const wxString & *long_help_string* = wxEmptyString)

void wxAuiToolBar::Clear ( )

void wxAuiToolBar::ClearTools ( )

bool wxAuiToolBar::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxAUI_TB_DEFAULT_STYLE)

Really create wxAuiToolBar created using default constructor.

Since

2.9.5

bool wxAuiToolBar::DeleteByIndex (int *tool_id*)

bool wxAuiToolBar::DeleteTool (int *tool_id*)

void wxAuiToolBar::EnableTool (int *tool_id*, bool *state*)

wxControl* wxAuiToolBar::FindControl (int *window_id*)

wxAuiToolBarItem* wxAuiToolBar::FindTool (int *tool_id*) const

wxAuiToolBarItem* wxAuiToolBar::FindToolByIndex (int *idx*) const

wxAuiToolBarItem* wxAuiToolBar::FindToolByPosition (wxCoord *x*, wxCoord *y*) const

wxAuiToolBarArt* wxAuiToolBar::GetArtProvider ( ) const

bool wxAuiToolBar::GetGripperVisible ( ) const

wxSize wxAuiToolBar::GetHintSize (int *dock_direction*) const

get size of hint rectangle for a particular dock location

bool wxAuiToolBar::GetOverflowVisible ( ) const

bool wxAuiToolBar::GetToolBarFits ( ) const

wxBitmap wxAuiToolBar::GetToolBitmap (int *tool_id*) const

wxSize wxAuiToolBar::GetToolBitmapSize ( ) const

int wxAuiToolBar::GetToolBorderPadding ( ) const

size_t wxAuiToolBar::GetToolCount ( ) const

bool wxAuiToolBar::GetToolDropDown (int *tool_id*) const

Returns whether the specified toolbar item has an associated drop down button.

See also

wxAuiToolBarItem::HasDropDown()

bool wxAuiToolBar::GetToolEnabled (int *tool_id*) const

bool wxAuiToolBar::GetToolFits (int *tool_id*) const

bool wxAuiToolBar::GetToolFitsByIndex (int *tool_id*) const

int wxAuiToolBar::GetToolIndex (int *tool_id*) const

wxString wxAuiToolBar::GetToolLabel (int *tool_id*) const

wxString wxAuiToolBar::GetToolLongHelp (int *tool_id*) const

int wxAuiToolBar::GetToolPacking ( ) const

int wxAuiToolBar::GetToolPos (int *tool_id*) const

int wxAuiToolBar::GetToolProportion (int *tool_id*) const

wxRect wxAuiToolBar::GetToolRect (int *tool_id*) const

int wxAuiToolBar::GetToolSeparation ( ) const

wxString wxAuiToolBar::GetToolShortHelp (int *tool_id*) const

bool wxAuiToolBar::GetToolSticky (int *tool_id*) const

int wxAuiToolBar::GetToolTextOrientation ( ) const

bool wxAuiToolBar::GetToolToggled (int *tool_id*) const

long wxAuiToolBar::GetWindowStyleFlag ( ) const `[virtual]`

Gets the window style that was passed to the constructor or Create() method.

GetWindowStyle() is another name for the same function.

Reimplemented from wxWindow.

bool wxAuiToolBar::IsPaneValid (const wxAuiPaneInfo & *pane*) const

bool wxAuiToolBar::Realize ( )

void wxAuiToolBar::SetArtProvider (wxAuiToolBarArt * *art*)

void wxAuiToolBar::SetCustomOverflowItems (const wxAuiToolBarItemArray & *prepend*, const wxAuiToolBarItemArray & *append*)

bool wxAuiToolBar::SetFont (const wxFont & *font*) `[virtual]`

Sets the font for this window.

This function should not be called for the parent window if you don't want its font to be inherited by its children, use SetOwnFont() instead in this case and see InheritAttributes() for more explanations.

Please notice that the given font is not automatically used for wxPaintDC objects associated with this window, you need to call wxDC::SetFont too. However this font is used by any standard controls for drawing their text as well as by GetTextExtent().

Parameters

font	Font to associate with this window, pass wxNullFont to reset to the default font.

Returns

true if the font was really changed, false if it was already set to this font and nothing was done.

See also

GetFont(), InheritAttributes()

Reimplemented from wxWindow.

void wxAuiToolBar::SetGripperVisible (bool *visible*)

void wxAuiToolBar::SetMargins (const wxSize & *size*)

void wxAuiToolBar::SetMargins (int *x,* int *y*)

void wxAuiToolBar::SetMargins (int *left,* int *right,* int *top,* int *bottom*)

void wxAuiToolBar::SetOverflowVisible (bool *visible*)

void wxAuiToolBar::SetToolBitmap (int *tool_id,* const wxBitmap & *bitmap*)

void wxAuiToolBar::SetToolBitmapSize (const wxSize & *size*)

void wxAuiToolBar::SetToolBorderPadding (int *padding*)

void wxAuiToolBar::SetToolDropDown (int *tool_id,* bool *dropdown*)

Set whether the specified toolbar item has a drop down button.

This is only valid for wxITEM_NORMAL tools.

See also

wxAuiToolBarItem::SetHasDropDown()

void wxAuiToolBar::SetToolLabel (int *tool_id,* const wxString & *label*)

void wxAuiToolBar::SetToolLongHelp (int *tool_id,* const wxString & *help_string*)

void wxAuiToolBar::SetToolPacking (int *packing*)

void wxAuiToolBar::SetToolProportion (int *tool_id,* int *proportion*)

void wxAuiToolBar::SetToolSeparation (int *separation*)

void wxAuiToolBar::SetToolShortHelp (int *tool_id,* const wxString & *help_string*)

void wxAuiToolBar::SetToolSticky (int *tool_id,* bool *sticky*)

void wxAuiToolBar::SetToolTextOrientation (int *orientation*)

void wxAuiToolBar::SetWindowStyleFlag (long *style*) `[virtual]`

Sets the style of the window.

Please note that some styles cannot be changed after the window creation and that Refresh() might need to be called after changing the others for the change to take place immediately.

See Window styles for more information about flags.

See also

GetWindowStyleFlag()

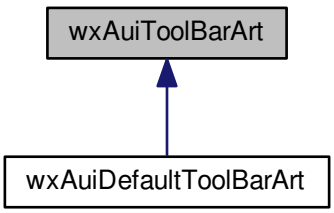
Reimplemented from wxWindow.

void wxAuiToolBar::ToggleTool (int *tool_id,* bool *state*)

21.51 wxAuiToolBarArt Class Reference

`#include <wx/aui/auibar.h>`

Inheritance diagram for wxAuiToolBarArt:



21.51.1 Detailed Description

wxAuiToolBarArt is part of the wxAUI class framework.

See also wxAuiToolBar and wxAUI Overview.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiToolBarArt ()
- virtual wxAuiToolBarArt * Clone ()=0
- virtual void SetFlags (unsigned int flags)=0
- virtual unsigned int GetFlags ()=0
- virtual void SetFont (const wxFont &font)=0

- virtual wxFont GetFont ()=0
- virtual void SetTextOrientation (int orientation)=0
- virtual int GetTextOrientation ()=0
- virtual void DrawBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0
- virtual void DrawPlainBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0
- virtual void DrawLabel (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)=0
- virtual void DrawButton (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)=0
- virtual void DrawDropDownButton (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)=0
- virtual void DrawControlLabel (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item, const wxRect &rect)=0
- virtual void DrawSeparator (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0
- virtual void DrawGripper (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0
- virtual void DrawOverflowButton (wxDC &dc, wxWindow *wnd, const wxRect &rect, int state)=0
- virtual wxSize GetLabelSize (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item)=0
- virtual wxSize GetToolSize (wxDC &dc, wxWindow *wnd, const wxAuiToolBarItem &item)=0
- virtual int GetElementSize (int element_id)=0
- virtual void SetElementSize (int element_id, int size)=0
- virtual int ShowDropDown (wxWindow *wnd, const wxAuiToolBarItemArray &items)=0

21.51.2 Constructor & Destructor Documentation

wxAuiToolBarArt::wxAuiToolBarArt ( )

21.51.3 Member Function Documentation

virtual wxAuiToolBarArt* wxAuiToolBarArt::Clone ( ) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawBackground (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawButton (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item,* const wxRect & *rect*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawControlLabel (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item,* const wxRect & *rect*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawDropDownButton (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item,* const wxRect & *rect*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawGripper (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawLabel (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item,* const wxRect & *rect*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawOverflowButton (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect,* int *state*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawPlainBackground (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::DrawSeparator (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual int wxAuiToolBarArt::GetElementSize (int *element_id*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual unsigned int wxAuiToolBarArt::GetFlags ( ) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual wxFont wxAuiToolBarArt::GetFont ( ) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual wxSize wxAuiToolBarArt::GetLabelSize (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual int wxAuiToolBarArt::GetTextOrientation ( ) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual wxSize wxAuiToolBarArt::GetToolSize (wxDC & *dc,* wxWindow * *wnd,* const wxAuiToolBarItem & *item*) [pure virtual]

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::SetElementSize (int *element_id,* int *size*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::SetFlags (unsigned int *flags*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::SetFont (const wxFont & *font*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

virtual void wxAuiToolBarArt::SetTextOrientation (int *orientation*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

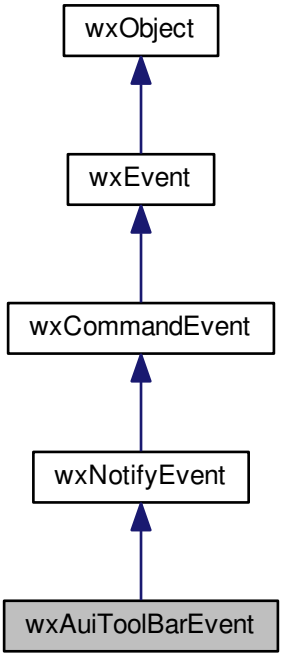
virtual int wxAuiToolBarArt::ShowDropDown (wxWindow * *wnd,* const wxAuiToolBarItemArray & *items*) `[pure virtual]`

Implemented in wxAuiDefaultToolBarArt.

21.52 wxAuiToolBarEvent Class Reference

`#include <wx/aui/auibar.h>`

Inheritance diagram for wxAuiToolBarEvent:



21.52.1 Detailed Description

wxAuiToolBarEvent is used for the events generated by wxAuiToolBar.

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- bool IsDropDownClicked () const

  Returns whether the drop down menu has been clicked.
- wxPoint GetClickPoint () const

  Returns the point where the user clicked with the mouse.
- wxRect GetItemRect () const

  Returns the wxAuiToolBarItem rectangle bounding the mouse click point.
- int GetToolId () const

  Returns the wxAuiToolBarItem identifier.

Additional Inherited Members

21.52.2 Member Function Documentation

wxPoint wxAuiToolBarEvent::GetClickPoint ( ) const

Returns the point where the user clicked with the mouse.

wxRect wxAuiToolBarEvent::GetItemRect ( ) const

Returns the wxAuiToolBarItem rectangle bounding the mouse click point.

int wxAuiToolBarEvent::GetToolId ( ) const

Returns the wxAuiToolBarItem identifier.

bool wxAuiToolBarEvent::IsDropDownClicked ( ) const

Returns whether the drop down menu has been clicked.

21.53 wxAuiToolBarItem Class Reference

`#include <wx/aui/auibar.h>`

21.53.1 Detailed Description

wxAuiToolBarItem is part of the wxAUI class framework, representing a toolbar element.

See also wxAuiToolBar and wxAUI Overview.

It has a unique id (except for the separators which always have id = -1), the style (telling whether it is a normal button, separator or a control), the state (toggled or not, enabled or not) and short and long help strings. The default implementations use the short help string for the tooltip text which is popped up when the mouse pointer enters the tool and the long help string for the applications status bar (currently not implemented).

Library: wxAui

Category: Window Docking (wxAUI)

Public Member Functions

- wxAuiToolBarItem ()

  Default Constructor.
- wxAuiToolBarItem (const wxAuiToolBarItem &c)

  Assigns the properties of the wxAuiToolBarItem "c" to this.
- wxAuiToolBarItem & operator= (const wxAuiToolBarItem &c)

  Assigns the properties of the wxAuiToolBarItem "c" to this, returning a pointer to this.
- void Assign (const wxAuiToolBarItem &c)

  Assigns the properties of the wxAuiToolBarItem "c" to this.
- void SetWindow (wxWindow *w)

  Assigns a window to the toolbar item.

- wxWindow * GetWindow ()

  Returns the wxWindow associated to the toolbar item.*
- void SetId (int new_id)

  Sets the toolbar item identifier.
- int GetId () const

  Returns the toolbar item identifier.
- void SetKind (int new_kind)

  Sets the wxAuiToolBarItem kind.
- int GetKind () const

  Returns the toolbar item kind.
- void SetState (int new_state)
- int GetState () const
- void SetSizerItem (wxSizerItem *s)
- wxSizerItem * GetSizerItem () const
- void SetLabel (const wxString &s)
- const wxString & GetLabel () const
- void SetBitmap (const wxBitmap &bmp)
- const wxBitmap & GetBitmap () const
- void SetDisabledBitmap (const wxBitmap &bmp)
- const wxBitmap & GetDisabledBitmap () const
- void SetHoverBitmap (const wxBitmap &bmp)
- const wxBitmap & GetHoverBitmap () const
- void SetShortHelp (const wxString &s)
- const wxString & GetShortHelp () const
- void SetLongHelp (const wxString &s)
- const wxString & GetLongHelp () const
- void SetMinSize (const wxSize &s)
- const wxSize & GetMinSize () const
- void SetSpacerPixels (int s)
- int GetSpacerPixels () const
- void SetProportion (int p)
- int GetProportion () const
- void SetActive (bool b)
- bool IsActive () const
- void SetHasDropDown (bool b)

  Set whether this tool has a drop down button.
- bool HasDropDown () const

  Returns whether the toolbar item has an associated drop down button.
- void SetSticky (bool b)
- bool IsSticky () const
- void SetUserData (long l)
- long GetUserData () const
- void SetAlignment (int l)
- int GetAlignment () const

21.53.2 Constructor & Destructor Documentation

wxAuiToolBarItem::wxAuiToolBarItem ( )

Default Constructor.

wxAuiToolBarItem::wxAuiToolBarItem (const wxAuiToolBarItem & *c*)

Assigns the properties of the wxAuiToolBarItem "c" to this.

21.53.3 Member Function Documentation

void wxAuiToolBarItem::Assign (const wxAuiToolBarItem & *c*)

Assigns the properties of the wxAuiToolBarItem "c" to this.

int wxAuiToolBarItem::GetAlignment ( ) const

const wxBitmap& wxAuiToolBarItem::GetBitmap ( ) const

const wxBitmap& wxAuiToolBarItem::GetDisabledBitmap ( ) const

const wxBitmap& wxAuiToolBarItem::GetHoverBitmap ( ) const

int wxAuiToolBarItem::GetId ( ) const

Returns the toolbar item identifier.

int wxAuiToolBarItem::GetKind ( ) const

Returns the toolbar item kind.

const wxString& wxAuiToolBarItem::GetLabel ( ) const

const wxString& wxAuiToolBarItem::GetLongHelp ( ) const

const wxSize& wxAuiToolBarItem::GetMinSize ( ) const

int wxAuiToolBarItem::GetProportion ( ) const

const wxString& wxAuiToolBarItem::GetShortHelp ( ) const

wxSizerItem* wxAuiToolBarItem::GetSizerItem ( ) const

int wxAuiToolBarItem::GetSpacerPixels ( ) const

int wxAuiToolBarItem::GetState ( ) const

long wxAuiToolBarItem::GetUserData ( ) const

wxWindow* wxAuiToolBarItem::GetWindow ( )

Returns the wxWindow* associated to the toolbar item.

bool wxAuiToolBarItem::HasDropDown ( ) const

Returns whether the toolbar item has an associated drop down button.

bool wxAuiToolBarItem::IsActive ( ) const

bool wxAuiToolBarItem::IsSticky ( ) const

wxAuiToolBarItem& wxAuiToolBarItem::operator= (const wxAuiToolBarItem & *c*)

Assigns the properties of the wxAuiToolBarItem "c" to this, returning a pointer to this.

void wxAuiToolBarItem::SetActive (bool *b*)

void wxAuiToolBarItem::SetAlignment (int *l*)

void wxAuiToolBarItem::SetBitmap (const wxBitmap & *bmp*)

void wxAuiToolBarItem::SetDisabledBitmap (const wxBitmap & *bmp*)

void wxAuiToolBarItem::SetHasDropDown (bool *b*)

Set whether this tool has a drop down button.

This is only valid for wxITEM_NORMAL tools.

void wxAuiToolBarItem::SetHoverBitmap (const wxBitmap & *bmp*)

void wxAuiToolBarItem::SetId (int *new_id*)

Sets the toolbar item identifier.

void wxAuiToolBarItem::SetKind (int *new_kind*)

Sets the wxAuiToolBarItem kind.

void wxAuiToolBarItem::SetLabel (const wxString & *s*)

void wxAuiToolBarItem::SetLongHelp (const wxString & *s*)

void wxAuiToolBarItem::SetMinSize (const wxSize & *s*)

void wxAuiToolBarItem::SetProportion (int *p*)

void wxAuiToolBarItem::SetShortHelp (const wxString & *s*)

void wxAuiToolBarItem::SetSizerItem (wxSizerItem * *s*)

void wxAuiToolBarItem::SetSpacerPixels (int *s*)

void wxAuiToolBarItem::SetState (int *new_state*)

void wxAuiToolBarItem::SetSticky (bool *b*)

void wxAuiToolBarItem::SetUserData (long *l*)

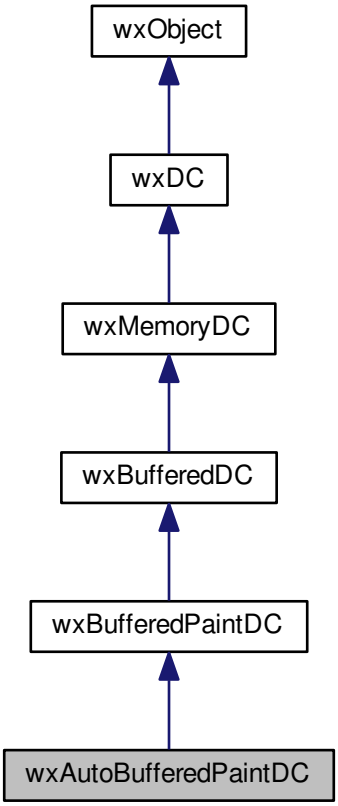
void wxAuiToolBarItem::SetWindow (wxWindow * *w*)

Assigns a window to the toolbar item.

21.54 wxAutoBufferedPaintDC Class Reference

`#include <wx/dcbuffer.h>`

Inheritance diagram for wxAutoBufferedPaintDC:



21.54.1 Detailed Description

This wxDC derivative can be used inside of an `EVT_PAINT()` event handler to achieve double-buffered drawing.

Just use this class instead of wxPaintDC and make sure wxWindow::SetBackgroundStyle() is called with wxBG_↩ STYLE_PAINT somewhere in the class initialization code, and that's all you have to do to (mostly) avoid flicker.

The difference between wxBufferedPaintDC and this class is that this class won't double-buffer on platforms which have native double-buffering already, avoiding any unnecessary buffering to avoid flicker.

wxAutoBufferedPaintDC is simply a typedef of wxPaintDC on platforms that have native double-buffering, otherwise, it is a typedef of wxBufferedPaintDC.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxBufferedPaintDC, wxPaintDC

Public Member Functions

- wxAutoBufferedPaintDC (wxWindow *window)

  Constructor.

Additional Inherited Members

21.54.2 Constructor & Destructor Documentation

wxAutoBufferedPaintDC::wxAutoBufferedPaintDC (wxWindow * *window*)

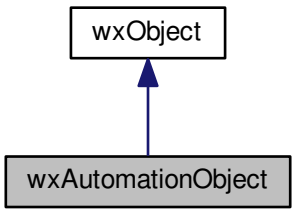
Constructor.

Pass a pointer to the window on which you wish to paint.

21.55 wxAutomationObject Class Reference

`#include <wx/msw/ole/automtn.h>`

Inheritance diagram for wxAutomationObject:



21.55.1 Detailed Description

The **wxAutomationObject** class represents an OLE automation object containing a single data member, an IDispatch pointer.

It contains a number of functions that make it easy to perform automation operations, and set and get properties. The class makes heavy use of the wxVariant class.

The usage of these classes is quite close to OLE automation usage in Visual Basic. The API is high-level, and the application can specify multiple properties in a single string. The following example gets the current Excel instance, and if it exists, makes the active cell bold.

```
wxAutomationObject excelObject;
  if (excelObject.GetInstance("Excel.Application"))
      excelObject.PutProperty("ActiveCell.Font.Bold", @true);
```

Note that this class obviously works under Windows only.

Availability: only available for the wxMSW port.

Library: wxCore

Category: Data Structures

See also

wxVariant, wxVariantDataCurrency, wxVariantDataErrorCode, wxVariantDataSafeArray

Public Member Functions

- wxAutomationObject (WXIDISPATCH *dispatchPtr=NULL)

  Constructor, taking an optional IDispatch pointer which will be released when the object is deleted.
- ~wxAutomationObject ()

  Destructor.
- bool CreateInstance (const wxString &progId) const

  Creates a new object based on the ProgID, returning true if the object was successfully created, or false if not.
- bool IsOk () const

  Checks if the object is in a valid state.
- void * GetDispatchPtr () const

  Gets the IDispatch pointer.
- bool GetInstance (const wxString &progId, int flags=wxAutomationInstance_CreateIfNeeded) const

  Retrieves the current object associated with the specified ProgID, and attaches the IDispatch pointer to this object.
- bool GetObject (wxAutomationObject &obj, const wxString &property, int noArgs=0, wxVariant args[]=NULL) const

  Retrieves a property from this object, assumed to be a dispatch pointer, and initialises obj with it.
- bool Invoke (const wxString &member, int action, wxVariant &retValue, int noArgs, wxVariant args[], const wxVariant *ptrArgs[]=0) const

  This function is a low-level implementation that allows access to the IDispatch Invoke function.
- void SetDispatchPtr (WXIDISPATCH *dispatchPtr)

  Sets the IDispatch pointer.
- LCID GetLCID () const

  Returns the locale identifier used in automation calls.
- void SetLCID (LCID lcid)

  Sets the locale identifier to be used in automation calls performed by this object.
- long GetConvertVariantFlags () const

  Returns the flags used for conversions between wxVariant and OLE VARIANT, see wxOleConvertVariantFlags.
- void SetConvertVariantFlags (long flags)

  Sets the flags used for conversions between wxVariant and OLE VARIANT, see wxOleConvertVariantFlags.

- wxVariant CallMethod (const wxString &method, int noArgs, wxVariant args[]) const

  Calls an automation method for this object.
- const wxVariant CallMethod (const wxString &method,...) const

  Calls an automation method for this object.

- wxVariant GetProperty (const wxString &property, int noArgs, wxVariant args[]) const

  Gets a property value from this object.
- const wxVariant GetProperty (const wxString &property,...) const

Gets a property value from this object.

- bool PutProperty (const wxString &property, int noArgs, wxVariant args[])

  Puts a property value into this object.
- const bool PutProperty (const wxString &property,...)

  Puts a property value into this object.

Additional Inherited Members

21.55.2 Constructor & Destructor Documentation

wxAutomationObject::wxAutomationObject (WXIDISPATCH * *dispatchPtr* = NULL)

Constructor, taking an optional IDispatch pointer which will be released when the object is deleted.

wxAutomationObject::~wxAutomationObject ( )

Destructor.

If the internal IDispatch pointer is non-null, it will be released.

21.55.3 Member Function Documentation

wxVariant wxAutomationObject::CallMethod (const wxString & *method,* int *noArgs,* wxVariant *args[]*) const

Calls an automation method for this object.

The first form takes a method name, number of arguments, and an array of variants. The second form takes a method name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

Note that *method* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects. For example:

const wxVariant wxAutomationObject::CallMethod (const wxString & *method,* ...) const

Calls an automation method for this object.

The first form takes a method name, number of arguments, and an array of variants. The second form takes a method name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

Note that *method* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects. For example:

bool wxAutomationObject::CreateInstance (const wxString & *progId*) const

Creates a new object based on the ProgID, returning true if the object was successfully created, or false if not.

long wxAutomationObject::GetConvertVariantFlags ( ) const

Returns the flags used for conversions between wxVariant and OLE VARIANT, see wxOleConvertVariantFlags.

The default value is wxOleConvertVariant_Default for compatibility but it can be changed using SetConvertVariant↩Flags().

Notice that objects obtained by GetObject() inherit the flags from the one that created them.

Since

3.0

void* wxAutomationObject::GetDispatchPtr ( ) const

Gets the IDispatch pointer.

Notice that the return value of this function is an untyped pointer but it can be safely cast to `IDispatch`.

bool wxAutomationObject::GetInstance (const wxString & *progId,* int *flags* = wxAutomationInstance_CreateIfNeeded) const

Retrieves the current object associated with the specified ProgID, and attaches the IDispatch pointer to this object.

If attaching to an existing object failed and *flags* includes wxAutomationInstance_CreateIfNeeded flag, a new object will be created. Otherwise this function will normally log an error message which may be undesirable if the object may or may not exist. The wxAutomationInstance_SilentIfNone flag can be used to prevent the error from being logged in this case.

Returns true if a pointer was successfully retrieved, false otherwise.

Note that this cannot cope with two instances of a given OLE object being active simultaneously, such as two copies of Excel running. Which object is referenced cannot currently be specified.

Parameters

progId	COM ProgID, e.g. "Excel.Application"
flags	The creation flags (this parameters was added in wxWidgets 2.9.2)

LCID wxAutomationObject::GetLCID ( ) const

Returns the locale identifier used in automation calls.

The default is LOCALE_SYSTEM_DEFAULT but the objects obtained by GetObject() inherit the locale identifier from the one that created them.

Since

2.9.5

bool wxAutomationObject::GetObject (wxAutomationObject & *obj,* const wxString & *property,* int *noArgs* = `0`, wxVariant *args[]* = `NULL`) const

Retrieves a property from this object, assumed to be a dispatch pointer, and initialises *obj* with it.

To avoid having to deal with IDispatch pointers directly, use this function in preference to GetProperty() when retrieving objects from other objects. Note that an IDispatch pointer is stored as a void* pointer in wxVariant objects.

See also

GetProperty()

wxVariant wxAutomationObject::GetProperty (const wxString & *property,* int *noArgs,* wxVariant *args[]*) const

Gets a property value from this object.

The first form takes a property name, number of arguments, and an array of variants. The second form takes a property name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

Note that *property* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects.

const wxVariant wxAutomationObject::GetProperty (const wxString & *property,* ...) const

Gets a property value from this object.

The first form takes a property name, number of arguments, and an array of variants. The second form takes a property name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

Note that *property* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects.

bool wxAutomationObject::Invoke (const wxString & *member,* int *action,* wxVariant & *retValue,* int *noArgs,* wxVariant *args[],* const wxVariant * *ptrArgs[]* = 0) const

This function is a low-level implementation that allows access to the IDispatch Invoke function.

It is not meant to be called directly by the application, but is used by other convenience functions.

Parameters

member	The member function or property name.
action	Bitlist: may contain DISPATCH_PROPERTYPUT, DISPATCH_PROPERTYPUTREF, DISPATCH_METHOD.
retValue	Return value (ignored if there is no return value)
noArgs	Number of arguments in args or ptrArgs.
args	If non-null, contains an array of variants.
ptrArgs	If non-null, contains an array of constant pointers to variants.

Returns

true if the operation was successful, false otherwise.

Remarks

Two types of argument array are provided, so that when possible pointers are used for efficiency.

bool wxAutomationObject::IsOk ( ) const

Checks if the object is in a valid state.

Returns true if the object was successfully initialized or false if it has no valid IDispatch pointer.

See also

GetDispatchPtr()

bool wxAutomationObject::PutProperty (const wxString & *property,* int *noArgs,* wxVariant *args[]*)

Puts a property value into this object.

The first form takes a property name, number of arguments, and an array of variants. The second form takes a property name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

Note that *property* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects.

const bool wxAutomationObject::PutProperty (const wxString & *property,* ...)

Puts a property value into this object.

The first form takes a property name, number of arguments, and an array of variants. The second form takes a property name and zero to six constant references to variants. Since the variant class has constructors for the basic data types, and C++ provides temporary objects automatically, both of the following lines are syntactically valid:

Note that *property* can contain dot-separated property names, to save the application needing to call GetProperty several times using several temporary objects.

void wxAutomationObject::SetConvertVariantFlags (long *flags*)

Sets the flags used for conversions between wxVariant and OLE VARIANT, see wxOleConvertVariantFlags.

The default value is wxOleConvertVariant_Default.

Since

3.0

void wxAutomationObject::SetDispatchPtr (WXIDISPATCH * *dispatchPtr*)

Sets the IDispatch pointer.

This function does not check if there is already an IDispatch pointer. You may need to cast from IDispatch* to WXIDISPATCH* when calling this function.

void wxAutomationObject::SetLCID (LCID *lcid*)

Sets the locale identifier to be used in automation calls performed by this object.

The default value is LOCALE_SYSTEM_DEFAULT.

Notice that any automation objects created by this one inherit the same LCID.

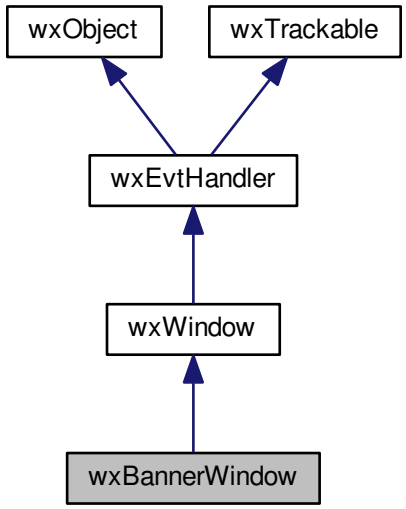
Since

2.9.5

21.56 wxBannerWindow Class Reference

`#include <wx/bannerwindow.h>`

Inheritance diagram for wxBannerWindow:



21.56.1 Detailed Description

A simple banner window showing either a bitmap or text.

Banner windows can be used to present some overview of the current window contents to the user in an aesthetically pleasant way. They are typically positioned along the left or top edge of the window (although this class also supports right- and bottom-aligned banners) and show either a bitmap with a logo or a few lines of text on a gradient-filled background.

Using this class is very simple, e.g.:

```
MyFrame::MyFrame(...)
{
    ... create the frame itself ...

    // Create and initialize the banner.
    wxBannerWindow* banner = new wxBannerWindow(this,
      wxTOP);
    banner->SetText("Welcome to my wonderful program",
                    "  Before doing anything else, you need to connect to "
                    "the online server.\n"
                    "  Please enter your credentials in the controls below.");

    // And position it along the left edge of the window.
    wxSizer* sizer = new wxBoxSizer(wxVERTICAL);
    sizer->Add(banner, wxSizerFlags().Expand());

    ... add the rest of the window contents to the same sizer ...

    SetSizerAndFit(sizer);
}
```

This class is currently implemented generically and so looks the same under all platforms.

Library: wxAdvanced

Category: Miscellaneous Windows

Since

2.9.3

Public Member Functions

- wxBannerWindow ()

  Default constructor, use Create() later.
- wxBannerWindow (wxWindow *parent, wxDirection dir=wxLEFT)

  Convenient constructor that should be used in the majority of cases.
- wxBannerWindow (wxWindow *parent, wxWindowID winid, wxDirection dir=wxLEFT, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxBannerWindowNameStr)

  Full constructor provided for consistency with the other classes only.
- bool Create (wxWindow *parent, wxWindowID winid, wxDirection dir=wxLEFT, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxBannerWindowNameStr)

  Really create the banner window for the objects created using the default constructor.
- void SetBitmap (const wxBitmap &bmp)

  Provide the bitmap to use as background.
- void SetText (const wxString &title, const wxString &message)

  Set the text to display.
- void SetGradient (const wxColour &start, const wxColour &end)

  Set the colours between which the gradient runs.

Additional Inherited Members

21.56.2 Constructor & Destructor Documentation

wxBannerWindow::wxBannerWindow ( )

Default constructor, use Create() later.

This constructor is only used for two-step creation, if possible, prefer using the constructor below directly instead of using this one and calling Create() later.

wxBannerWindow::wxBannerWindow (wxWindow * *parent*, wxDirection *dir* = wxLEFT)

Convenient constructor that should be used in the majority of cases.

The only really important arguments of the full constructor below are *parent* and *dir* so this class provides a convenient constructor taking only them.

The banner orientation changes how the text in it is displayed and also defines where is the bitmap truncated if it's too big to fit but doesn't do anything for the banner position, this is supposed to be taken care of in the usual way, e.g. using sizers.

wxBannerWindow::wxBannerWindow (wxWindow * *parent*, wxWindowID *winid*, wxDirection *dir* = wxLEFT, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxBannerWindowNameStr)

Full constructor provided for consistency with the other classes only.

Prefer to use the shorter constructor documented above. You should rarely, if ever, need to use non-default values for any other parameters: as the banner window doesn't generate any events, its identifier is not particularly useful; its position and size will be almost always managed by the containing sizer and it doesn't have any specific styles. So only the parent and the banner direction need to be specified.

21.56.3 Member Function Documentation

bool wxBannerWindow::Create (wxWindow * *parent*, wxWindowID *winid*, wxDirection *dir* = wxLEFT, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxBannerWindowNameStr)

Really create the banner window for the objects created using the default constructor.

It's an error to call Create() for the objects created using non-default constructor.

void wxBannerWindow::SetBitmap (const wxBitmap & *bmp*)

Provide the bitmap to use as background.

Notice that ideally the bitmap should be big enough to always cover the entire banner, e.g. for a horizontal banner with wxTOP style its width should be bigger than any reasonable window size. Otherwise the bitmap is extended to cover the entire window area with a solid colour taken from the bitmap pixel on the edge in which direction the extension occurs so all bitmap pixels on this edge (top for wxLEFT, right for wxTOP and wxBOTTOM and bottom for wxRIGHT) should have the same colour to avoid jarring discontinuity.

If, on the other hand, the bitmap is bigger than the window size, then it is truncated. For wxLEFT orientation the bitmap is truncated from the top, for wxTOP and wxBOTTOM – from the right and for wxRIGHT – from the bottom, so put the most important part of the bitmap information in the opposite direction where it will never be truncated.

If no valid background bitmap is specified, the banner draws gradient background but if a valid bitmap is given here, the gradient is not draw and the start and end colours specified for it are ignored.

Parameters

bmp	Bitmap to use as background. May be invalid to indicate that no background bitmap should be used.

void wxBannerWindow::SetGradient (const wxColour & *start*, const wxColour & *end*)

Set the colours between which the gradient runs.

The gradient colours are ignored if SetBitmap() is used.

void wxBannerWindow::SetText (const wxString & *title*, const wxString & *message*)

Set the text to display.

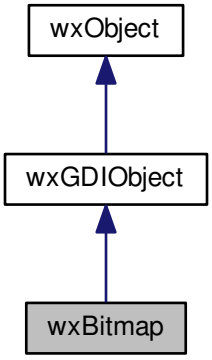
This is mutually exclusive with SetBitmap().

Title is rendered in bold and should be single line, message can have multiple lines but is not wrapped automatically, include explicit line breaks in the string if you want to have multiple lines.

21.57 wxBitmap Class Reference

```
#include <wx/bitmap.h>
```

Inheritance diagram for wxBitmap:



21.57.1 Detailed Description

This class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour or colour with alpha channel support.

If you need direct access the bitmap data instead going through drawing to it using wxMemoryDC you need to use the wxPixelData class (either wxNativePixelData for RGB bitmaps or wxAlphaPixelData for bitmaps with an additionally alpha channel).

Note that many wxBitmap functions take a *type* parameter, which is a value of the wxBitmapType enumeration. The validity of those values depends however on the platform where your program is running and from the wxWidgets configuration. If all possible wxWidgets settings are used:

- wxMSW supports BMP and ICO files, BMP and ICO resources;
- wxGTK supports any file supported by gdk-pixbuf;
- wxMac supports PICT resources;
- wxX11 supports XPM files, XPM data, XBM data;

In addition, wxBitmap can load and save all formats that wxImage can; see wxImage for more info. Of course, you must have loaded the wxImage handlers (see wxInitAllImageHandlers() and wxImage::AddHandler). Note that all available wxBitmapHandlers for a given wxWidgets port are automatically loaded at startup so you won't need to use wxBitmap::AddHandler.

More on the difference between wxImage and wxBitmap: wxImage is just a buffer of RGB bytes with an optional buffer for the alpha bytes. It is all generic, platform independent and image file format independent code. It includes generic code for scaling, resizing, clipping, and other manipulations of the image data. OTOH, wxBitmap is intended to be a wrapper of whatever is the native image format that is quickest/easiest to draw to a DC or to be the target of the drawing operations performed on a wxMemoryDC. By splitting the responsibilities between wxImage/wxBitmap like this then it's easier to use generic code shared by all platforms and image types for generic operations and platform specific code where performance or compatibility is needed.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullBitmap

See also

Bitmaps and Icons, Supported Bitmap File Formats, wxDC::Blit, wxIcon, wxCursor, wxMemoryDC, wxImage, wxPixelData

Public Member Functions

- wxBitmap ()

  Default constructor.
- wxBitmap (const wxBitmap &bitmap)

  Copy constructor, uses reference counting.
- wxBitmap (const char bits[], int width, int height, int depth=1)

  Creates a bitmap from the given array bits.
- wxBitmap (int width, int height, int depth=wxBITMAP_SCREEN_DEPTH)

  Creates a new bitmap.
- wxBitmap (const wxSize &sz, int depth=wxBITMAP_SCREEN_DEPTH)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxBitmap (const char *const *bits)

  Creates a bitmap from XPM data.
- wxBitmap (const wxString &name, wxBitmapType type=wxBITMAP_DEFAULT_TYPE)

  Loads a bitmap from a file or resource.
- wxBitmap (const wxImage &img, int depth=wxBITMAP_SCREEN_DEPTH)

  Creates this bitmap object from the given image.
- wxBitmap (const wxCursor &cursor)

  Creates bitmap corresponding to the given cursor.
- virtual ~wxBitmap ()

  Destructor.
- virtual wxImage ConvertToImage () const

  Creates an image from a platform-dependent bitmap.
- virtual bool CopyFromIcon (const wxIcon &icon)

  Creates the bitmap from an icon.
- virtual bool Create (int width, int height, int depth=wxBITMAP_SCREEN_DEPTH)

  Creates a fresh bitmap.
- virtual bool Create (const wxSize &sz, int depth=wxBITMAP_SCREEN_DEPTH)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- virtual int GetDepth () const

  Gets the colour depth of the bitmap.
- virtual int GetHeight () const

  Gets the height of the bitmap in pixels.
- virtual wxMask * GetMask () const

  Gets the associated mask (if any) which may have been loaded from a file or set for the bitmap.
- virtual wxPalette * GetPalette () const

  Gets the associated palette (if any) which may have been loaded from a file or set for the bitmap.
- virtual wxBitmap GetSubBitmap (const wxRect &rect) const

  Returns a sub bitmap of the current one as long as the rect belongs entirely to the bitmap.
- wxSize GetSize () const

  Returns the size of the bitmap in pixels.

- wxBitmap ConvertToDisabled (unsigned char brightness=255) const

  Returns disabled (dimmed) version of the bitmap.
- virtual int GetWidth () const

  Gets the width of the bitmap in pixels.
- virtual bool IsOk () const

  Returns true if bitmap data is present.
- virtual bool LoadFile (const wxString &name, wxBitmapType type=wxBITMAP_DEFAULT_TYPE)

  Loads a bitmap from a file or resource.
- virtual bool SaveFile (const wxString &name, wxBitmapType type, const wxPalette *palette=NULL) const

  Saves a bitmap in the named file.
- virtual void SetDepth (int depth)

  Sets the depth member (does not affect the bitmap data).
- virtual void SetHeight (int height)

  Sets the height member (does not affect the bitmap data).
- virtual void SetMask (wxMask *mask)

  Sets the mask for this bitmap.
- virtual void SetPalette (const wxPalette &palette)

  Sets the associated palette.
- virtual void SetWidth (int width)

  Sets the width member (does not affect the bitmap data).

Static Public Member Functions

- static void AddHandler (wxBitmapHandler *handler)

  Adds a handler to the end of the static list of format handlers.
- static void CleanUpHandlers ()

  Deletes all bitmap handlers.
- static wxBitmapHandler * FindHandler (const wxString &name)

  Finds the handler with the given name.
- static wxBitmapHandler * FindHandler (const wxString &extension, wxBitmapType bitmapType)

  Finds the handler associated with the given extension and type.
- static wxBitmapHandler * FindHandler (wxBitmapType bitmapType)

  Finds the handler associated with the given bitmap type.
- static wxList & GetHandlers ()

  Returns the static list of bitmap format handlers.
- static void InitStandardHandlers ()

  Adds the standard bitmap format handlers, which, depending on wxWidgets configuration, can be handlers for Windows bitmap, Windows bitmap resource, and XPM.
- static void InsertHandler (wxBitmapHandler *handler)

  Adds a handler at the start of the static list of format handlers.
- static wxBitmap NewFromPNGData (const void *data, size_t size)

  Loads a bitmap from the memory containing image data in PNG format.
- static bool RemoveHandler (const wxString &name)

  Finds the handler with the given name, and removes it.

Additional Inherited Members

21.57.2 Constructor & Destructor Documentation

wxBitmap::wxBitmap ( )

Default constructor.

Constructs a bitmap object with no data; an assignment or another member function such as Create() or LoadFile() must be called subsequently.

wxBitmap::wxBitmap (const wxBitmap & *bitmap*)

Copy constructor, uses reference counting.

To make a real copy, you can use:

```
wxBitmap newBitmap = oldBitmap.GetSubBitmap(
                     wxRect(0, 0, oldBitmap.GetWidth(), oldBitmap.GetHeight()));
```

wxBitmap::wxBitmap (const char *bits[]*, int *width*, int *height*, int *depth* = 1)

Creates a bitmap from the given array *bits*.

You should only use this function for monochrome bitmaps (depth 1) in portable programs: in this case the bits parameter should contain an XBM image.

For other bit depths, the behaviour is platform dependent: under Windows, the data is passed without any changes to the underlying CreateBitmap() API. Under other platforms, only monochrome bitmaps may be created using this constructor and wxImage should be used for creating colour bitmaps from static data.

Parameters

bits	Specifies an array of pixel values.
width	Specifies the width of the bitmap.
height	Specifies the height of the bitmap.
depth	Specifies the depth of the bitmap. If this is omitted, then a value of 1 (monochrome bitmap) is used.

wxPerl Note: In wxPerl use Wx::Bitmap->newFromBits(bits, width, height, depth).

wxBitmap::wxBitmap (int *width*, int *height*, int *depth* = wxBITMAP_SCREEN_DEPTH)

Creates a new bitmap.

A depth of wxBITMAP_SCREEN_DEPTH indicates the depth of the current screen or visual.

Some platforms only support 1 for monochrome and wxBITMAP_SCREEN_DEPTH for the current colour setting.

A depth of 32 including an alpha channel is supported under MSW, Mac and GTK+.

Parameters

width	The width of the bitmap in pixels, must be strictly positive.
height	The height of the bitmap in pixels, must be strictly positive.
depth	The number of bits used to represent each bitmap pixel.

wxBitmap::wxBitmap (const wxSize & *sz*, int *depth* = wxBITMAP_SCREEN_DEPTH)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxBitmap::wxBitmap (const char ∗const ∗ *bits*)

Creates a bitmap from XPM data.

wxPerl Note: In wxPerl use Wx::Bitmap->newFromXPM(data).

wxBitmap::wxBitmap (const wxString & *name*, wxBitmapType *type* = `wxBITMAP_DEFAULT_TYPE`)

Loads a bitmap from a file or resource.

Parameters

name	This can refer to a resource name or a filename under MS Windows and X. Its meaning is determined by the *type* parameter.
type	May be one of the wxBitmapType values and indicates which type of bitmap should be loaded. See the note in the class detailed description. Note that the wxBITMAP_DEFAULT_TYPE constant has different value under different wxWidgets ports. See the bitmap.h header for the value it takes for a specific port.

See also

LoadFile()

wxBitmap::wxBitmap (const wxImage & *img*, int *depth* = wxBITMAP_SCREEN_DEPTH)

Creates this bitmap object from the given image.

This has to be done to actually display an image as you cannot draw an image directly on a window.

The resulting bitmap will use the provided colour depth (or that of the current system if depth is wxBITMAP_SCR↩EEN_DEPTH) which entails that a colour reduction may take place.

On Windows, if there is a palette present (set with SetPalette), it will be used when creating the wxBitmap (most useful in 8-bit display mode). On other platforms, the palette is currently ignored.

Parameters

img	Platform-independent wxImage object.
depth	Specifies the depth of the bitmap. If this is omitted, the display depth of the screen is used.

wxBitmap::wxBitmap (const wxCursor & *cursor*) `[explicit]`

Creates bitmap corresponding to the given cursor.

This can be useful to display a cursor as it cannot be drawn directly on a window.

This constructor only exists in wxMSW and wxGTK (where it is implemented for GTK+ 2.8 or later) only.

Parameters

cursor	A valid wxCursor.

Since

3.1.0

virtual wxBitmap::∼wxBitmap ( ) `[virtual]`

Destructor.

See Object Destruction for more info.

If the application omits to delete the bitmap explicitly, the bitmap will be destroyed automatically by wxWidgets when the application exits.

Warning

Do not delete a bitmap that is selected into a memory device context.

21.57.3 Member Function Documentation

static void wxBitmap::AddHandler (wxBitmapHandler * *handler*) `[static]`

Adds a handler to the end of the static list of format handlers.

Parameters

handler	A new bitmap format handler object. There is usually only one instance of a given handler class in an application session.

Note that unlike wxImage::AddHandler, there's no documented list of the wxBitmapHandlers available in wxWidgets. This is because they are platform-specific and most important, they are all automatically loaded at startup.

If you want to be sure that wxBitmap can load a certain type of image, you'd better use wxImage::AddHandler.

See also

wxBitmapHandler

static void wxBitmap::CleanUpHandlers ( ) `[static]`

Deletes all bitmap handlers.

This function is called by wxWidgets on exit.

wxBitmap wxBitmap::ConvertToDisabled (unsigned char *brightness* = `255`) const

Returns disabled (dimmed) version of the bitmap.

This method is not available when `wxUSE_IMAGE == 0`.

Since

2.9.0

virtual wxImage wxBitmap::ConvertToImage ( ) const `[virtual]`

Creates an image from a platform-dependent bitmap.

This preserves mask information so that bitmaps and images can be converted back and forth without loss in that respect.

virtual bool wxBitmap::CopyFromIcon (const wxIcon & *icon*) `[virtual]`

Creates the bitmap from an icon.

virtual bool wxBitmap::Create (int *width,* int *height,* int *depth* = wxBITMAP_SCREEN_DEPTH) [virtual]

Creates a fresh bitmap.

If the final argument is omitted, the display depth of the screen is used.

Parameters

width	The width of the bitmap in pixels, must be strictly positive.
height	The height of the bitmap in pixels, must be strictly positive.
depth	The number of bits used to represent each bitmap pixel.

Returns

true if the creation was successful.

virtual bool wxBitmap::Create (const wxSize & *sz*, int *depth* = wxBITMAP_SCREEN_DEPTH) [virtual]

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

static wxBitmapHandler* wxBitmap::FindHandler (const wxString & *name*) [static]

Finds the handler with the given *name*.

Returns

A pointer to the handler if found, NULL otherwise.

static wxBitmapHandler* wxBitmap::FindHandler (const wxString & *extension*, wxBitmapType *bitmapType*) [static]

Finds the handler associated with the given *extension* and *type*.

Parameters

extension	The file extension, such as "bmp" (without the dot).
bitmapType	The bitmap type managed by the handler, see wxBitmapType.

Returns

A pointer to the handler if found, NULL otherwise.

static wxBitmapHandler* wxBitmap::FindHandler (wxBitmapType *bitmapType*) [static]

Finds the handler associated with the given bitmap type.

Parameters

bitmapType	The bitmap type managed by the handler, see wxBitmapType.

Returns

A pointer to the handler if found, NULL otherwise.

See also

wxBitmapHandler

virtual int wxBitmap::GetDepth ( ) const `[virtual]`

Gets the colour depth of the bitmap.

A value of 1 indicates a monochrome bitmap.

static wxList& wxBitmap::GetHandlers ( ) `[static]`

Returns the static list of bitmap format handlers.

See also

wxBitmapHandler

virtual int wxBitmap::GetHeight ( ) const `[virtual]`

Gets the height of the bitmap in pixels.

See also

GetWidth(), GetSize()

virtual wxMask* wxBitmap::GetMask ( ) const `[virtual]`

Gets the associated mask (if any) which may have been loaded from a file or set for the bitmap.

See also

SetMask(), wxMask

virtual wxPalette* wxBitmap::GetPalette ( ) const `[virtual]`

Gets the associated palette (if any) which may have been loaded from a file or set for the bitmap.

See also

wxPalette

wxSize wxBitmap::GetSize ( ) const

Returns the size of the bitmap in pixels.

Since

2.9.0

See also

GetHeight(), GetWidth()

virtual wxBitmap wxBitmap::GetSubBitmap (const wxRect & *rect*) const `[virtual]`

Returns a sub bitmap of the current one as long as the rect belongs entirely to the bitmap.

This function preserves bit depth and mask information.

virtual int wxBitmap::GetWidth ( ) const `[virtual]`

Gets the width of the bitmap in pixels.

See also

GetHeight(), GetSize()

static void wxBitmap::InitStandardHandlers ( ) `[static]`

Adds the standard bitmap format handlers, which, depending on wxWidgets configuration, can be handlers for Windows bitmap, Windows bitmap resource, and XPM.

This function is called by wxWidgets on startup.

See also

wxBitmapHandler

static void wxBitmap::InsertHandler (wxBitmapHandler * *handler*) `[static]`

Adds a handler at the start of the static list of format handlers.

Parameters

handler	A new bitmap format handler object. There is usually only one instance of a given handler class in an application session.

See also

wxBitmapHandler

virtual bool wxBitmap::IsOk ( ) const `[virtual]`

Returns true if bitmap data is present.

virtual bool wxBitmap::LoadFile (const wxString & *name*, wxBitmapType *type* = wxBITMAP_DEFAULT_TYPE) `[virtual]`

Loads a bitmap from a file or resource.

Parameters

name	Either a filename or a Windows resource name. The meaning of name is determined by the *type* parameter.
type	One of the wxBitmapType values; see the note in the class detailed description. Note that the wxBITMAP_DEFAULT_TYPE constant has different value under different wxWidgets ports. See the bitmap.h header for the value it takes for a specific port.

Returns

true if the operation succeeded, false otherwise.

Remarks

A palette may be associated with the bitmap if one exists (especially for colour Windows bitmaps), and if the code supports it. You can check if one has been created by using the GetPalette() member.

See also

SaveFile()

static wxBitmap wxBitmap::NewFromPNGData (const void * *data*, size_t *size*) [static]

Loads a bitmap from the memory containing image data in PNG format.

This helper function provides the simplest way to create a wxBitmap from PNG image data. On most platforms, it's simply a wrapper around wxImage loading functions and so requires the PNG image handler to be registered by either calling wxInitAllImageHandlers() which also registers all the other image formats or including the necessary header:

```
#include <wx/imagpng.h>
```

and calling

```
wxImage::AddHandler(new wxPNGHandler);
```

in your application startup code.

However under OS X this function uses native image loading and so doesn't require wxWidgets PNG support.

Since

2.9.5

static bool wxBitmap::RemoveHandler (const wxString & *name*) [static]

Finds the handler with the given name, and removes it.

The handler is not deleted.

Parameters

name	The handler name.

Returns

true if the handler was found and removed, false otherwise.

See also

wxBitmapHandler

virtual bool wxBitmap::SaveFile (const wxString & *name*, wxBitmapType *type*, const wxPalette * *palette* = NULL) const [virtual]

Saves a bitmap in the named file.

Parameters

name	A filename. The meaning of name is determined by the type parameter.
type	One of the wxBitmapType values; see the note in the class detailed description.
palette	An optional palette used for saving the bitmap.

Returns

true if the operation succeeded, false otherwise.

Remarks

Depending on how wxWidgets has been configured, not all formats may be available.

See also

LoadFile()

virtual void wxBitmap::SetDepth (int *depth*) [virtual]

Sets the depth member (does not affect the bitmap data).

Todo since these functions do not affect the bitmap data, why they exist??

Parameters

depth	Bitmap depth.

virtual void wxBitmap::SetHeight (int *height*) [virtual]

Sets the height member (does not affect the bitmap data).

Parameters

height	Bitmap height in pixels.

virtual void wxBitmap::SetMask (wxMask * *mask*) [virtual]

Sets the mask for this bitmap.

Remarks

The bitmap object owns the mask once this has been called.

See also

GetMask(), wxMask

virtual void wxBitmap::SetPalette (const wxPalette & *palette*) [virtual]

Sets the associated palette.

(Not implemented under GTK+).

Parameters

palette	The palette to set.

See also

wxPalette

virtual void wxBitmap::SetWidth (int *width*) `[virtual]`

Sets the width member (does not affect the bitmap data).

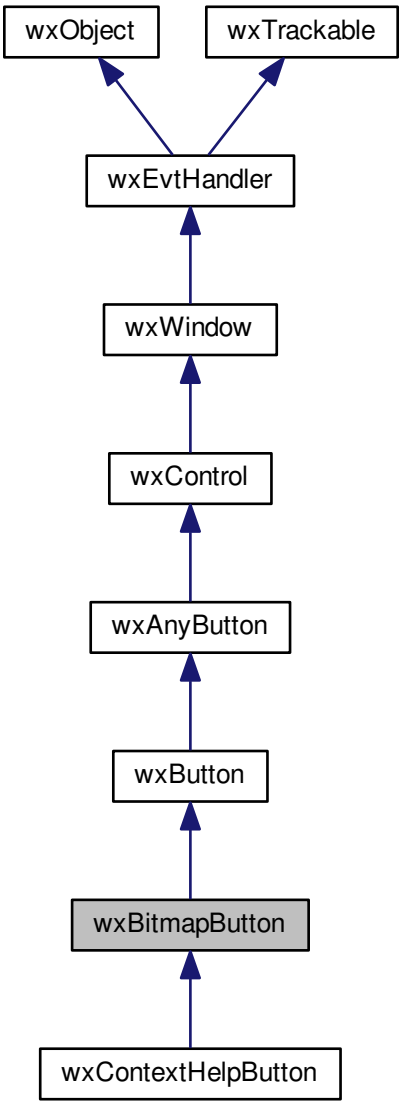
Parameters

width	Bitmap width in pixels.

21.58 wxBitmapButton Class Reference

`#include <wx/bmpbuttn.h>`

Inheritance diagram for wxBitmapButton:



21.58.1 Detailed Description

A bitmap button is a control that contains a bitmap.

Notice that since wxWidgets 2.9.1 bitmap display is supported by the base wxButton class itself and the only tiny advantage of using this class is that it allows to specify the bitmap in its constructor, unlike wxButton. Please see the base class documentation for more information about images support in wxButton.

Styles

This class supports the following styles:

- wxBU_LEFT: Left-justifies the bitmap label.
- wxBU_TOP: Aligns the bitmap label to the top of the button.
- wxBU_RIGHT: Right-justifies the bitmap label.
- wxBU_BOTTOM: Aligns the bitmap label to the bottom of the button.

Note that the wxBU_EXACTFIT style supported by wxButton is not used by this class as bitmap buttons don't have any minimal standard size by default.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_BUTTON(id, func): Process a `wxEVT_BUTTON` event, when the button is clicked.

Library: wxCore

Category: Controls

See also

wxButton

Public Member Functions

- wxBitmapButton ()

  Default ctor.
- wxBitmapButton (wxWindow *parent, wxWindowID id, const wxBitmap &bitmap, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxBU_AUTODRAW, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)

  Constructor, creating and showing a button.
- bool Create (wxWindow *parent, wxWindowID id, const wxBitmap &bitmap, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxBU_AUTODRAW, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)

  Button creation function for two-step creation.

Static Public Member Functions

- static wxBitmapButton * NewCloseButton (wxWindow *parent, wxWindowID winid)

  Helper function creating a standard-looking "Close" button.

Additional Inherited Members

21.58.2 Constructor & Destructor Documentation

wxBitmapButton::wxBitmapButton ( )

Default ctor.

wxBitmapButton::wxBitmapButton (wxWindow * *parent*, wxWindowID *id*, const wxBitmap & *bitmap*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxBU_AUTODRAW, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxButtonNameStr`)

Constructor, creating and showing a button.

Parameters

parent	Parent window. Must not be NULL.
id	Button identifier. The value wxID_ANY indicates a default value.
bitmap	Bitmap to be displayed.
pos	Button position. If wxDefaultPosition is specified then a default position is chosen.
size	Button size. If wxDefaultSize is specified then the button is sized appropriately for the bitmap.
style	Window style. See wxBitmapButton.
validator	Window validator.
name	Window name.

Remarks

The bitmap parameter is normally the only bitmap you need to provide, and wxWidgets will draw the button correctly in its different states. If you want more control, call any of the functions SetBitmapPressed(), SetBitmapFocus(), SetBitmapDisabled().

See also

Create(), wxValidator

21.58.3 Member Function Documentation

bool wxBitmapButton::Create (wxWindow * *parent*, wxWindowID *id*, const wxBitmap & *bitmap*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxBU_AUTODRAW, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxButtonNameStr`)

Button creation function for two-step creation.

For more details, see wxBitmapButton().

static wxBitmapButton* wxBitmapButton::NewCloseButton (wxWindow * *parent*, wxWindowID *winid*) `[static]`

Helper function creating a standard-looking "Close" button.

To get the best results, platform-specific code may need to be used to create a small, title bar-like "Close" button. This function is provided to avoid the need to test for the current platform and creates the button with as native look as possible.

Parameters

parent	The button parent window, must be non-NULL.
winid	The identifier for the new button.

Returns

The new button.

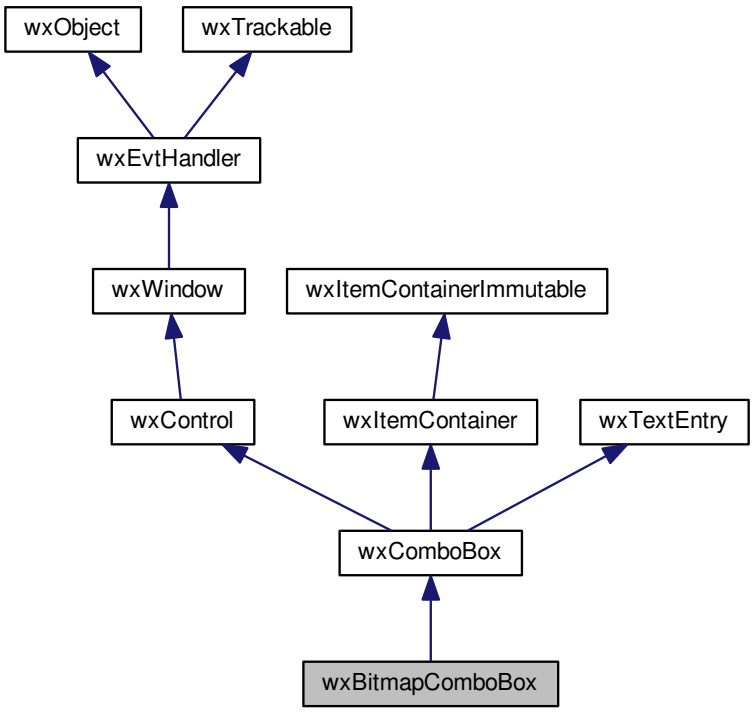
Since

2.9.5

21.59 wxBitmapComboBox Class Reference

```
#include <wx/bmpcbox.h>
```

Inheritance diagram for wxBitmapComboBox:



21.59.1 Detailed Description

A combobox that displays bitmap in front of the list items.

It currently only allows using bitmaps of one size, and resizes itself so that a bitmap can be shown next to the text field.

Remarks

While wxBitmapComboBox contains the wxComboBox API, but it might not actually be derived from that class. In fact, if the platform does not have a native implementation, wxBitmapComboBox will inherit from wxOwnerDrawnComboBox. You can determine if the implementation is generic by checking whether `wxG↩ ENERIC_BITMAPCOMBOBOX` is defined. Currently wxBitmapComboBox is implemented natively for MSW and GTK+.

Styles

This class supports the following styles:

- wxCB_READONLY: Creates a combobox without a text editor. On some platforms the control may appear very different when this style is used.
- wxCB_SORT: Sorts the entries in the list alphabetically.
- wxTE_PROCESS_ENTER: The control will generate the event wxEVT_TEXT_ENTER (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls). Windows only.

Todo create wxCB_PROCESS_ENTER rather than reusing wxTE_PROCESS_ENTER!

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_COMBOBOX(id, func): Process a `wxEVT_COMBOBOX` event, when an item on the list is selected.
- EVT_TEXT(id, func): Process a `wxEVT_TEXT` event, when the combobox text changes.
- EVT_TEXT_ENTER(id, func): Process a `wxEVT_TEXT_ENTER` event, when RETURN is pressed in the combobox.

Library: wxAdvanced

Category: Controls

See also

wxComboBox, wxChoice, wxOwnerDrawnComboBox, wxCommandEvent

Public Member Functions

- wxBitmapComboBox ()

  Default ctor.
- wxBitmapComboBox (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxBitmapComboBoxNameStr)

  Constructor, creating and showing a combobox.
- wxBitmapComboBox (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxBitmapComboBoxNameStr)

  Constructor, creating and showing a combobox.
- virtual ~wxBitmapComboBox ()

  Destructor, destroying the combobox.
- int Append (const wxString &item, const wxBitmap &bitmap=wxNullBitmap)

  Adds the item to the end of the combo box.
- int Append (const wxString &item, const wxBitmap &bitmap, void *clientData)

  Adds the item to the end of the combo box, associating the given untyped, client data pointer clientData with the item.
- int Append (const wxString &item, const wxBitmap &bitmap, wxClientData *clientData)

  Adds the item to the end of the combo box, associating the given typed client data pointer clientData with the item.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, int n, const wxString choices[], long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxBitmapComboBoxNameStr)

  Creates the combobox for two-step construction.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxBitmapComboBoxNameStr)

  Creates the combobox for two-step construction.

- virtual wxSize GetBitmapSize () const

  Returns the size of the bitmaps used in the combo box.
- virtual wxBitmap GetItemBitmap (unsigned int n) const

  Returns the bitmap of the item with the given index.
- int Insert (const wxString &item, const wxBitmap &bitmap, unsigned int pos)

  Inserts the item into the list before pos.
- int Insert (const wxString &item, const wxBitmap &bitmap, unsigned int pos, void *clientData)

  Inserts the item into the list before pos, associating the given untyped, client data pointer with the item.
- int Insert (const wxString &item, const wxBitmap &bitmap, unsigned int pos, wxClientData *clientData)

  Inserts the item into the list before pos, associating the given typed client data pointer with the item.
- virtual void SetItemBitmap (unsigned int n, const wxBitmap &bitmap)

  Sets the bitmap for the given item.

Additional Inherited Members

21.59.2 Constructor & Destructor Documentation

wxBitmapComboBox::wxBitmapComboBox ( )

Default ctor.

wxBitmapComboBox::wxBitmapComboBox (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxBitmapComboBoxNameStr)

Constructor, creating and showing a combobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial selection string. An empty string indicates no selection.
pos	Initial position.
size	Initial size.
n	Number of strings with which to initialise the control.
choices	An array of strings with which to initialise the control.
style	The window style, see wxCB_* flags.
validator	Validator which can be used for additional data checks.
name	Control name.

See also

Create(), wxValidator

wxBitmapComboBox::wxBitmapComboBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style*, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxBitmapComboBoxNameStr)

Constructor, creating and showing a combobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial selection string. An empty string indicates no selection.
pos	Initial position.
size	Initial size.
choices	An wxArrayString with which to initialise the control.
style	The window style, see wxCB_* flags.
validator	Validator which can be used for additional data checks.
name	Control name.

See also

Create(), wxValidator

virtual wxBitmapComboBox::~wxBitmapComboBox ( ) [virtual]

Destructor, destroying the combobox.

21.59.3 Member Function Documentation

int wxBitmapComboBox::Append (const wxString & *item*, const wxBitmap & *bitmap* = wxNullBitmap)

Adds the item to the end of the combo box.

int wxBitmapComboBox::Append (const wxString & *item*, const wxBitmap & *bitmap*, void * *clientData*)

Adds the item to the end of the combo box, associating the given untyped, client data pointer *clientData* with the item.

int wxBitmapComboBox::Append (const wxString & *item*, const wxBitmap & *bitmap*, wxClientData * *clientData*)

Adds the item to the end of the combo box, associating the given typed client data pointer *clientData* with the item.

bool wxBitmapComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, int *n*, const wxString *choices[]*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxBitmapComboBoxNameStr)

Creates the combobox for two-step construction.

bool wxBitmapComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxBitmapComboBoxNameStr)

Creates the combobox for two-step construction.

virtual wxSize wxBitmapComboBox::GetBitmapSize ( ) const [virtual]

Returns the size of the bitmaps used in the combo box.

If the combo box is empty, then wxDefaultSize is returned.

virtual wxBitmap wxBitmapComboBox::GetItemBitmap (unsigned int *n*) const `[virtual]`

Returns the bitmap of the item with the given index.

int wxBitmapComboBox::Insert (const wxString & *item,* const wxBitmap & *bitmap,* unsigned int *pos*)

Inserts the item into the list before *pos*.

Not valid for `wxCB_SORT` style, use Append() instead.

int wxBitmapComboBox::Insert (const wxString & *item,* const wxBitmap & *bitmap,* unsigned int *pos,* void * *clientData*)

Inserts the item into the list before pos, associating the given untyped, client data pointer with the item.

Not valid for `wxCB_SORT` style, use Append() instead.

int wxBitmapComboBox::Insert (const wxString & *item,* const wxBitmap & *bitmap,* unsigned int *pos,* wxClientData * *clientData*)

Inserts the item into the list before pos, associating the given typed client data pointer with the item.

Not valid for `wxCB_SORT` style, use Append() instead.

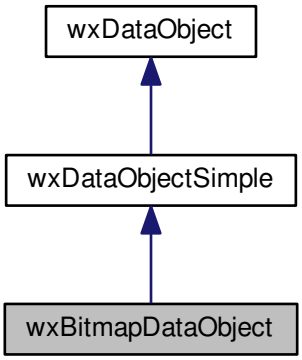
virtual void wxBitmapComboBox::SetItemBitmap (unsigned int *n,* const wxBitmap & *bitmap*) `[virtual]`

Sets the bitmap for the given item.

21.60 wxBitmapDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxBitmapDataObject:



21.60.1 Detailed Description

wxBitmapDataObject is a specialization of wxDataObject for bitmap data.

It can be used without change to paste data into the wxClipboard or a wxDropSource. A user may wish to derive a new class from this class for providing a bitmap on-demand in order to minimize memory consumption when offering data in several formats, such as a bitmap and GIF.

This class may be used as is, but GetBitmap() may be overridden to increase efficiency.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject, wxDataObjectSimple, wxFileDataObject, wxTextDataObject, wxDataObject

Public Member Functions

- wxBitmapDataObject (const wxBitmap &bitmap=wxNullBitmap)

  Constructor, optionally passing a bitmap (otherwise use SetBitmap() later).
- virtual wxBitmap GetBitmap () const

  Returns the bitmap associated with the data object.
- virtual void SetBitmap (const wxBitmap &bitmap)

  Sets the bitmap associated with the data object.

Additional Inherited Members

21.60.2 Constructor & Destructor Documentation

wxBitmapDataObject::wxBitmapDataObject (const wxBitmap & *bitmap* = wxNullBitmap)

Constructor, optionally passing a bitmap (otherwise use SetBitmap() later).

21.60.3 Member Function Documentation

virtual wxBitmap wxBitmapDataObject::GetBitmap ( ) const `[virtual]`

Returns the bitmap associated with the data object.

You may wish to override this method when offering data on-demand, but this is not required by wxWidgets' internals. Use this method to get data in bitmap form from the wxClipboard.

virtual void wxBitmapDataObject::SetBitmap (const wxBitmap & *bitmap*) `[virtual]`

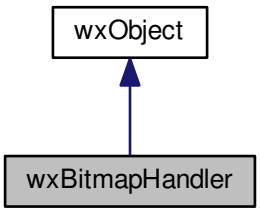
Sets the bitmap associated with the data object.

This method is called when the data object receives data. Usually there will be no reason to override this function.

21.61 wxBitmapHandler Class Reference

`#include <wx/bitmap.h>`

Inheritance diagram for wxBitmapHandler:



21.61.1 Detailed Description

This is the base class for implementing bitmap file loading/saving, and bitmap creation from data.

It is used within wxBitmap and is not normally seen by the application.

If you wish to extend the capabilities of wxBitmap, derive a class from wxBitmapHandler and add the handler using wxBitmap::AddHandler() in your application initialization.

Note that all wxBitmapHandlers provided by wxWidgets are part of the wxCore library. For details about the default handlers, please see the note in the wxBitmap class documentation.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

Bitmaps and Icons, wxBitmap, wxIcon, wxCursor

Public Member Functions

- wxBitmapHandler ()

  Default constructor.
- virtual ~wxBitmapHandler ()

  Destroys the wxBitmapHandler object.
- virtual bool Create (wxBitmap *bitmap, const void *data, wxBitmapType type, int width, int height, int depth=1)

  Creates a bitmap from the given data, which can be of arbitrary type.
- const wxString & GetExtension () const

  Gets the file extension associated with this handler.
- const wxString & GetName () const

  Gets the name of this handler.
- wxBitmapType GetType () const

  Gets the bitmap type associated with this handler.

- virtual bool LoadFile (wxBitmap *bitmap, const wxString &name, wxBitmapType type, int desiredWidth, int desiredHeight)

  Loads a bitmap from a file or resource, putting the resulting data into bitmap.
- virtual bool SaveFile (const wxBitmap *bitmap, const wxString &name, wxBitmapType type, const wxPalette *palette=NULL) const

  Saves a bitmap in the named file.
- void SetExtension (const wxString &extension)

  Sets the handler extension.
- void SetName (const wxString &name)

  Sets the handler name.
- void SetType (wxBitmapType type)

  Sets the handler type.

Additional Inherited Members

21.61.2 Constructor & Destructor Documentation

wxBitmapHandler::wxBitmapHandler ( )

Default constructor.

In your own default constructor, initialise the members m_name, m_extension and m_type.

virtual wxBitmapHandler::~wxBitmapHandler ( ) [virtual]

Destroys the wxBitmapHandler object.

21.61.3 Member Function Documentation

virtual bool wxBitmapHandler::Create (wxBitmap * *bitmap,* const void * *data,* wxBitmapType *type,* int *width,* int *height,* int *depth* = 1) [virtual]

Creates a bitmap from the given data, which can be of arbitrary type.

The wxBitmap object *bitmap* is manipulated by this function.

Parameters

bitmap	The wxBitmap object.
width	The width of the bitmap in pixels.
height	The height of the bitmap in pixels.
depth	The depth of the bitmap in pixels. If this is wxBITMAP_SCREEN_DEPTH, the screen depth is used.
data	Data whose type depends on the value of type.
type	A bitmap type identifier - see wxBitmapType for a list of possible values.

Returns

true if the call succeeded, false otherwise (the default).

const wxString& wxBitmapHandler::GetExtension ( ) const

Gets the file extension associated with this handler.

const wxString& wxBitmapHandler::GetName ( ) const

Gets the name of this handler.

wxBitmapType wxBitmapHandler::GetType ( ) const

Gets the bitmap type associated with this handler.

virtual bool wxBitmapHandler::LoadFile (wxBitmap * *bitmap,* const wxString & *name,* wxBitmapType *type,* int *desiredWidth,* int *desiredHeight*) [virtual]

Loads a bitmap from a file or resource, putting the resulting data into *bitmap*.

Note

Under MSW, when loading a bitmap from resources (i.e. using `wxBITMAP_TYPE_BMP_RESOURCE` as *type*), the light grey colour is considered to be transparent, for historical reasons. If you want to handle the light grey pixels normally instead, call SetMask(NULL) after loading the bitmap.

Parameters

bitmap	The bitmap object which is to be affected by this operation.
name	Either a filename or a Windows resource name. The meaning of name is determined by the type parameter.
type	See wxBitmapType for values this can take.
desiredWidth	The desired width for the loaded bitmap.
desiredHeight	The desired height for the loaded bitmap.

Returns

true if the operation succeeded, false otherwise.

See also

wxBitmap::LoadFile, wxBitmap::SaveFile, SaveFile()

virtual bool wxBitmapHandler::SaveFile (const wxBitmap * *bitmap,* const wxString & *name,* wxBitmapType *type,* const wxPalette * *palette* = `NULL`) const [virtual]

Saves a bitmap in the named file.

Parameters

bitmap	The bitmap object which is to be affected by this operation.
name	A filename. The meaning of name is determined by the type parameter.
type	See wxBitmapType for values this can take.
palette	An optional palette used for saving the bitmap.

Returns

true if the operation succeeded, false otherwise.

See also

wxBitmap::LoadFile, wxBitmap::SaveFile, LoadFile()

void wxBitmapHandler::SetExtension (const wxString & *extension*)

Sets the handler extension.

Parameters

extension	Handler extension.

void wxBitmapHandler::SetName (const wxString & *name*)

Sets the handler name.

Parameters

name	Handler name.

void wxBitmapHandler::SetType (wxBitmapType *type*)

Sets the handler type.

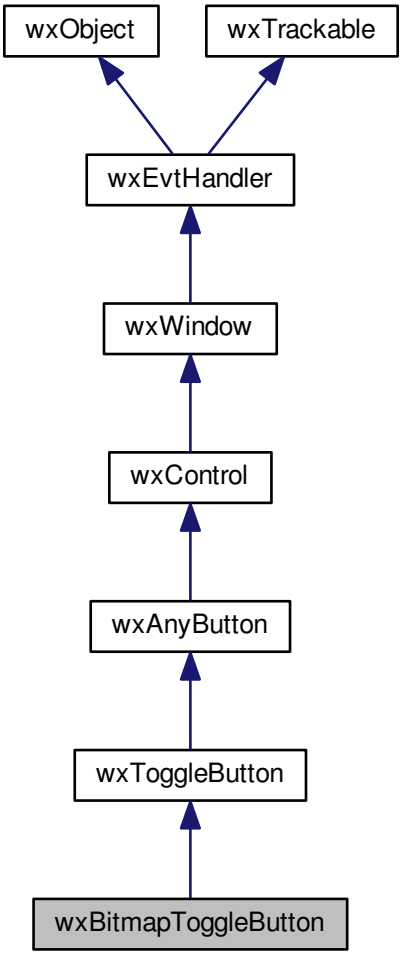
Parameters

type	Handler type.

21.62 wxBitmapToggleButton Class Reference

```
#include <wx/tglbtn.h>
```

Inheritance diagram for wxBitmapToggleButton:



21.62.1 Detailed Description

wxBitmapToggleButton is a wxToggleButton that contains a bitmap instead of text.

This class is not available in all ports currently (although it is available in the major ones), test for `wxHAS_BITMAPTOGGLEBUTTON` to determine whether it can be used (in addition for possibly testing for `wxUSE_TOGGLEBTN` which can be set to 0 to explicitly disable support for this class and wxToggleButton).

This control emits an update UI event.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_TOGGLEBUTTON(id, func): Handles a wxEVT_TOGGLEBUTTON event.

Library: wxCore

Category: Controls

Public Member Functions

- wxBitmapToggleButton ()

  Default constructor.
- wxBitmapToggleButton (wxWindow *parent, wxWindowID id, const wxBitmap &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &val=wxDefaultValidator, const wxString &name=wxCheckBoxNameStr)

  Constructor, creating and showing a toggle button with the bitmap label.
- bool Create (wxWindow *parent, wxWindowID id, const wxBitmap &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &val=wxDefaultValidator, const wxString &name=wxCheckBoxNameStr)

  Create method for two-step construction.
- virtual bool GetValue () const

  Gets the state of the toggle button.
- virtual void SetValue (bool state)

  Sets the toggle button to the given state.

Additional Inherited Members

21.62.2 Constructor & Destructor Documentation

wxBitmapToggleButton::wxBitmapToggleButton ( )

Default constructor.

wxBitmapToggleButton::wxBitmapToggleButton (wxWindow * *parent*, wxWindowID *id*, const wxBitmap & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxValidator & *val* = wxDefaultValidator, const wxString & *name* = `wxCheckBoxNameStr`)

Constructor, creating and showing a toggle button with the bitmap *label*.

Internally calls Create().

21.62.3 Member Function Documentation

bool wxBitmapToggleButton::Create (wxWindow * *parent*, wxWindowID *id*, const wxBitmap & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxValidator & *val* = wxDefaultValidator, const wxString & *name* = `wxCheckBoxNameStr`)

Create method for two-step construction.

virtual bool wxBitmapToggleButton::GetValue ( ) const `[virtual]`

Gets the state of the toggle button.

Returns

Returns true if it is pressed, false otherwise.

Reimplemented from wxToggleButton.

virtual void wxBitmapToggleButton::SetValue (bool *state*) [virtual]

Sets the toggle button to the given state.

This does not cause a EVT_TOGGLEBUTTON event to be emitted.

Parameters

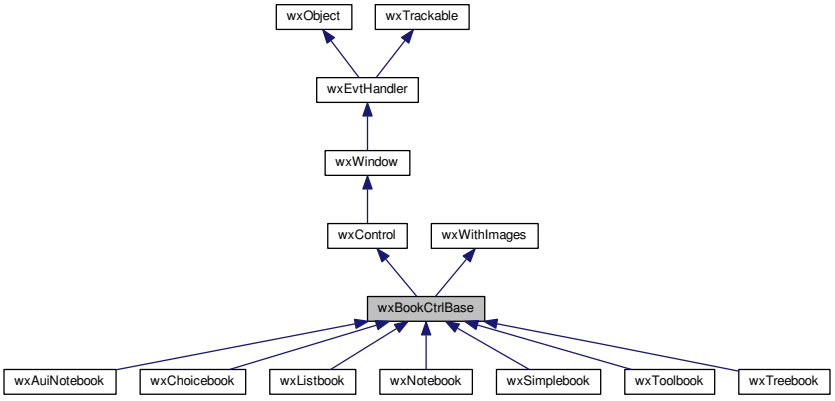
state	If true, the button is pressed.

Reimplemented from wxToggleButton.

21.63 wxBookCtrlBase Class Reference

#include <wx/bookctrl.h>

Inheritance diagram for wxBookCtrlBase:



21.63.1 Detailed Description

A book control is a convenient way of displaying multiple pages of information, displayed one page at a time.

wxWidgets has five variants of this control:

- wxChoicebook: controlled by a wxChoice
- wxListbook: controlled by a wxListCtrl
- wxNotebook: uses a row of tabs
- wxTreebook: controlled by a wxTreeCtrl
- wxToolbook: controlled by a wxToolBar

This abstract class is the parent of all these book controls, and provides their basic interface. This is a pure virtual class so you cannot allocate it directly.

Library: wxCore

Category: Book Controls

See also

wxBookCtrl Overview

Public Types

- enum { NO_IMAGE = -1 }

Public Member Functions

- wxBookCtrlBase ()

  Default ctor.
- wxBookCtrlBase (wxWindow *parent, wxWindowID winid, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Constructs the book control with the given parameters.
- bool Create (wxWindow *parent, wxWindowID winid, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Constructs the book control with the given parameters.
- virtual void SetPageSize (const wxSize &size)

  Sets the width and height of the pages.
- virtual int HitTest (const wxPoint &pt, long *flags=NULL) const

  Returns the index of the tab at the specified position or `wxNOT_FOUND` if none.

Image list functions

Each page may have an attached image.

The functions of this group manipulate that image.

- virtual int GetPageImage (size_t nPage) const =0

  Returns the image index for the given page.
- virtual bool SetPageImage (size_t page, int image)=0

  Sets the image index for the given page.

Page text functions

Each page has a text string attached.

The functions of this group manipulate that text.

- virtual wxString GetPageText (size_t nPage) const =0

  Returns the string for the given page.
- virtual bool SetPageText (size_t page, const wxString &text)=0

  Sets the text for the given page.

Selection functions

The functions of this group manipulate the selection.

- virtual int GetSelection () const =0

  Returns the currently selected page, or `wxNOT_FOUND` if none was selected.
- wxWindow * GetCurrentPage () const

  Returns the currently selected page or NULL.
- virtual int SetSelection (size_t page)=0

Sets the selection to the given page, returning the previous selection.

- void AdvanceSelection (bool forward=true)

  Cycles through the tabs.
- virtual int ChangeSelection (size_t page)=0

  Changes the selection to the given page, returning the previous selection.
- int FindPage (const wxWindow *page) const

  Returns the index of the specified tab window or `wxNOT_FOUND` if not found.

Page management functions

Functions for adding/removing pages from this control.

- virtual bool AddPage (wxWindow *page, const wxString &text, bool select=false, int imageId=NO_IMAGE)

  Adds a new page.
- virtual bool DeleteAllPages ()

  Deletes all pages.
- virtual bool DeletePage (size_t page)

  Deletes the specified page, and the associated window.
- virtual bool InsertPage (size_t index, wxWindow *page, const wxString &text, bool select=false, int imageId=NO_IMAGE)=0

  Inserts a new page at the specified position.
- virtual bool RemovePage (size_t page)

  Deletes the specified page, without deleting the associated window.
- virtual size_t GetPageCount () const

  Returns the number of pages in the control.
- wxWindow * GetPage (size_t page) const

  Returns the window at the given page position.

Additional Inherited Members

21.63.2 Member Enumeration Documentation

anonymous enum

Enumerator

NO_IMAGE Symbolic constant indicating that no image should be used.

21.63.3 Constructor & Destructor Documentation

wxBookCtrlBase::wxBookCtrlBase ( )

Default ctor.

wxBookCtrlBase::wxBookCtrlBase (wxWindow * *parent*, wxWindowID *winid*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = wxEmptyString)

Constructs the book control with the given parameters.

See Create() for two-step construction.

21.63.4 Member Function Documentation

virtual bool wxBookCtrlBase::AddPage (wxWindow * *page*, const wxString & *text*, bool *select* = `false`, int *imageId* = NO_IMAGE) `[virtual]`

Adds a new page.

The page must have the book control itself as the parent and must not have been added to this control previously.

The call to this function may generate the page changing events.

Parameters

page	Specifies the new page.
text	Specifies the text for the new page.
select	Specifies whether the page should be selected.
imageId	Specifies the optional image index for the new page.

Returns

true if successful, false otherwise.

Remarks

Do not delete the page, it will be deleted by the book control.

See also

InsertPage()

Reimplemented in wxAuiNotebook, and wxTreebook.

void wxBookCtrlBase::AdvanceSelection (bool *forward* = `true`)

Cycles through the tabs.

The call to this function generates the page changing events.

virtual int wxBookCtrlBase::ChangeSelection (size_t *page*) `[pure virtual]`

Changes the selection to the given page, returning the previous selection.

This function behaves as SetSelection() but does *not* generate the page changing events.

See User Generated Events vs Programmatically Generated Events for more information.

Implemented in wxNotebook, and wxAuiNotebook.

bool wxBookCtrlBase::Create (wxWindow * *parent*, wxWindowID *winid*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = wxEmptyString)

Constructs the book control with the given parameters.

virtual bool wxBookCtrlBase::DeleteAllPages ( ) `[virtual]`

Deletes all pages.

Reimplemented in wxAuiNotebook.

virtual bool wxBookCtrlBase::DeletePage (size_t *page*) `[virtual]`

Deletes the specified page, and the associated window.

The call to this function generates the page changing events.

Reimplemented in wxAuiNotebook, and wxTreebook.

int wxBookCtrlBase::FindPage (const wxWindow * *page*) const

Returns the index of the specified tab window or `wxNOT_FOUND` if not found.

Parameters

page	One of the control pages.

Returns

The zero-based tab index or `wxNOT_FOUND` if not found.

Since

2.9.5

wxWindow* wxBookCtrlBase::GetCurrentPage ( ) const

Returns the currently selected page or NULL.

wxWindow* wxBookCtrlBase::GetPage (size_t *page*) const

Returns the window at the given page position.

virtual size_t wxBookCtrlBase::GetPageCount ( ) const [virtual]

Returns the number of pages in the control.

Reimplemented in wxAuiNotebook.

virtual int wxBookCtrlBase::GetPageImage (size_t *nPage*) const [pure virtual]

Returns the image index for the given page.

Implemented in wxNotebook.

virtual wxString wxBookCtrlBase::GetPageText (size_t *nPage*) const [pure virtual]

Returns the string for the given page.

Implemented in wxAuiNotebook, and wxNotebook.

virtual int wxBookCtrlBase::GetSelection ( ) const [pure virtual]

Returns the currently selected page, or `wxNOT_FOUND` if none was selected.

Note that this method may return either the previously or newly selected page when called from the `EVT_BOOKCTRL_PAGE_CHANGED` handler depending on the platform and so wxBookCtrlEvent::GetSelection should be used instead in this case.

Implemented in wxAuiNotebook, wxNotebook, and wxTreebook.

virtual int wxBookCtrlBase::HitTest (const wxPoint & *pt*, long * *flags* = NULL) const [virtual]

Returns the index of the tab at the specified position or `wxNOT_FOUND` if none.

If *flags* parameter is non-NULL, the position of the point inside the tab is returned as well.

Parameters

pt	Specifies the point for the hit test.
flags	Return more details about the point, see returned value is a combination of wxBK_HITTEST_NOWHERE, wxBK_HITTEST_ONICON, wxBK_HITTEST_ONLABEL, wxBK_HITTEST_ONITEM, wxBK_HITTEST_ONPAGE.

Returns

Returns the zero-based tab index or `wxNOT_FOUND` if there is no tab at the specified position.

virtual bool wxBookCtrlBase::InsertPage (size_t *index*, wxWindow * *page*, const wxString & *text*, bool *select* = `false`, int *imageId* = NO_IMAGE) `[pure virtual]`

Inserts a new page at the specified position.

Parameters

index	Specifies the position for the new page.
page	Specifies the new page.
text	Specifies the text for the new page.
select	Specifies whether the page should be selected.
imageId	Specifies the optional image index for the new page.

Returns

true if successful, false otherwise.

Remarks

Do not delete the page, it will be deleted by the book control.

See also

AddPage()

Implemented in wxAuiNotebook, wxNotebook, and wxTreebook.

virtual bool wxBookCtrlBase::RemovePage (size_t *page*) `[virtual]`

Deletes the specified page, without deleting the associated window.

Reimplemented in wxAuiNotebook.

virtual bool wxBookCtrlBase::SetPageImage (size_t *page*, int *image*) `[pure virtual]`

Sets the image index for the given page.

image is an index into the image list which was set with SetImageList().

Implemented in wxAuiNotebook, and wxNotebook.

virtual void wxBookCtrlBase::SetPageSize (const wxSize & *size*) `[virtual]`

Sets the width and height of the pages.

Note

This method is currently not implemented for wxGTK.

virtual bool wxBookCtrlBase::SetPageText (size_t *page*, const wxString & *text*) [pure virtual]

Sets the text for the given page.

Implemented in wxAuiNotebook, and wxNotebook.

virtual int wxBookCtrlBase::SetSelection (size_t *page*) [pure virtual]

Sets the selection to the given page, returning the previous selection.

Notice that the call to this function generates the page changing events, use the ChangeSelection() function if you don't want these events to be generated.

See also

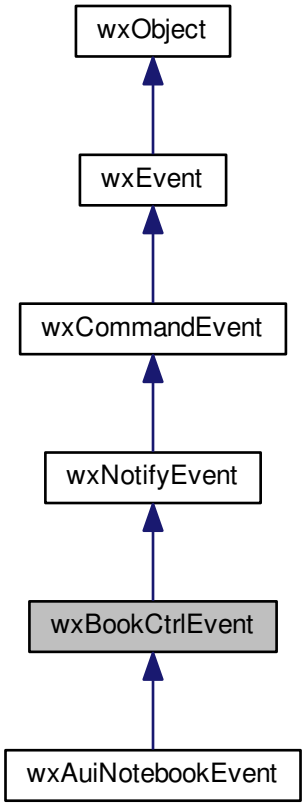
GetSelection()

Implemented in wxAuiNotebook, and wxNotebook.

21.64 wxBookCtrlEvent Class Reference

`#include <wx/bookctrl.h>`

Inheritance diagram for wxBookCtrlEvent:



21.64.1 Detailed Description

This class represents the events generated by book controls (wxNotebook, wxListbook, wxChoicebook, wxTreebook, wxAuiNotebook).

The PAGE_CHANGING events are sent before the current page is changed. It allows the program to examine the current page (which can be retrieved with wxBookCtrlEvent::GetOldSelection) and to veto the page change by calling wxNotifyEvent::Veto if, for example, the current values in the controls of the old page are invalid.

The PAGE_CHANGED events are sent after the page has been changed and the program cannot veto it any more, it just informs it about the page change.

To summarize, if the program is interested in validating the page values before allowing the user to change it, it should process the PAGE_CHANGING event, otherwise PAGE_CHANGED is probably enough. In any case, it is probably unnecessary to process both events at once.

Library: wxCore

Category: Events, Book Controls

See also

wxNotebook, wxListbook, wxChoicebook, wxTreebook, wxToolbook, wxAuiNotebook

Public Member Functions

- wxBookCtrlEvent (wxEventType eventType=wxEVT_NULL, int id=0, int sel=wxNOT_FOUND, int oldSel=wxNOT_FOUND)

  Constructor (used internally by wxWidgets only).
- int GetOldSelection () const

  Returns the page that was selected before the change, `wxNOT_FOUND` if none was selected.
- int GetSelection () const

  Returns the currently selected page, or `wxNOT_FOUND` if none was selected.
- void SetOldSelection (int page)

  Sets the id of the page selected before the change.
- void SetSelection (int page)

  Sets the selection member variable.

Additional Inherited Members

21.64.2 Constructor & Destructor Documentation

wxBookCtrlEvent::wxBookCtrlEvent (wxEventType *eventType* = wxEVT_NULL, int *id* = 0, int *sel* = wxNOT_FOUND, int *oldSel* = wxNOT_FOUND)

Constructor (used internally by wxWidgets only).

21.64.3 Member Function Documentation

int wxBookCtrlEvent::GetOldSelection ( ) const

Returns the page that was selected before the change, `wxNOT_FOUND` if none was selected.

int wxBookCtrlEvent::GetSelection ( ) const

Returns the currently selected page, or `wxNOT_FOUND` if none was selected.

Note

under Windows, GetSelection() will return the same value as GetOldSelection() when called from the `EVT_BOOKCTRL_PAGE_CHANGING` handler and not the page which is going to be selected.

void wxBookCtrlEvent::SetOldSelection (int *page*)

Sets the id of the page selected before the change.

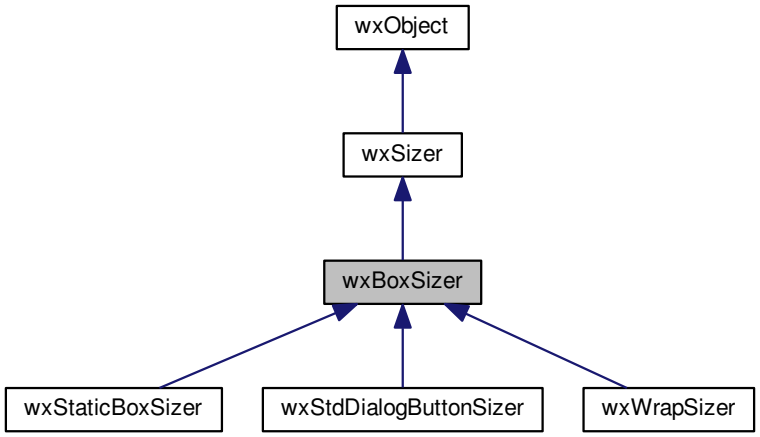
void wxBookCtrlEvent::SetSelection (int *page*)

Sets the selection member variable.

21.65 wxBoxSizer Class Reference

`#include <wx/sizer.h>`

Inheritance diagram for wxBoxSizer:



21.65.1 Detailed Description

The basic idea behind a box sizer is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.

For more information, please see Programming with wxBoxSizer.

Library: wxCore

Category: Window Layout

See also

wxSizer, Sizers Overview

Public Member Functions

- wxBoxSizer (int orient)

  Constructor for a wxBoxSizer.
- virtual wxSizerItem * AddSpacer (int size)

  Adds non-stretchable space to the main orientation of the sizer only.
- virtual wxSize CalcMin ()

  Implements the calculation of a box sizer's minimal.
- int GetOrientation () const

  Returns the orientation of the box sizer, either wxVERTICAL or wxHORIZONTAL.
- void SetOrientation (int orient)

  Sets the orientation of the box sizer, either wxVERTICAL or wxHORIZONTAL.
- virtual void RecalcSizes ()

  Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

Additional Inherited Members

21.65.2 Constructor & Destructor Documentation

wxBoxSizer::wxBoxSizer (int *orient*)

Constructor for a wxBoxSizer.

orient may be either of wxVERTICAL or wxHORIZONTAL for creating either a column sizer or a row sizer.

21.65.3 Member Function Documentation

virtual wxSizerItem* wxBoxSizer::AddSpacer (int *size*) `[virtual]`

Adds non-stretchable space to the main orientation of the sizer only.

More readable way of calling:

```
if ( wxBoxSizer::IsVertical() )
{
    wxBoxSizer::Add(0, size, 0).
}
else
{
    wxBoxSizer::Add(size, 0, 0).
}
```

Reimplemented from wxSizer.

virtual wxSize wxBoxSizer::CalcMin ( ) `[virtual]`

Implements the calculation of a box sizer's minimal.

It is used internally only and must not be called by the user. Documented for information.

Implements wxSizer.

Reimplemented in wxStaticBoxSizer, wxStdDialogButtonSizer, and wxWrapSizer.

int wxBoxSizer::GetOrientation ( ) const

Returns the orientation of the box sizer, either wxVERTICAL or wxHORIZONTAL.

virtual void wxBoxSizer::RecalcSizes ( ) [virtual]

Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

It is used internally only and must not be called by the user (call Layout() if you want to resize). Documented for information.

Implements wxSizer.

Reimplemented in wxStaticBoxSizer, wxStdDialogButtonSizer, and wxWrapSizer.

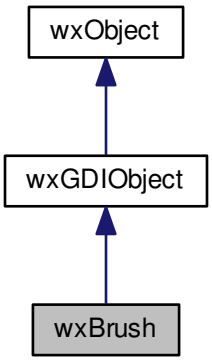
void wxBoxSizer::SetOrientation (int *orient*)

Sets the orientation of the box sizer, either wxVERTICAL or wxHORIZONTAL.

21.66 wxBrush Class Reference

`#include <wx/brush.h>`

Inheritance diagram for wxBrush:



21.66.1 Detailed Description

A brush is a drawing tool for filling in areas.

It is used for painting the background of rectangles, ellipses, etc. It has a colour and a style.

On a monochrome display, wxWidgets shows all brushes as white unless the colour is really black.

Do not initialize objects on the stack before the program commences, since other required structures may not have been set up yet. Instead, define global pointers to objects and create them in wxApp::OnInit or when required.

An application may wish to create brushes with different characteristics dynamically, and there is the consequent danger that a large number of duplicate brushes will be created. Therefore an application may wish to get a pointer

to a brush by using the global list of brushes wxTheBrushList, and calling the member function wxBrushList::FindOrCreateBrush().

This class uses reference counting and copy-on-write internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers:

- wxNullBrush
- wxBLACK_BRUSH
- wxBLUE_BRUSH
- wxCYAN_BRUSH
- wxGREEN_BRUSH
- wxYELLOW_BRUSH
- wxGREY_BRUSH
- wxLIGHT_GREY_BRUSH
- wxMEDIUM_GREY_BRUSH
- wxRED_BRUSH
- wxTRANSPARENT_BRUSH
- wxWHITE_BRUSH

See also

wxBrushList, wxDC, wxDC::SetBrush

Public Member Functions

- wxBrush ()

  Default constructor.
- wxBrush (const wxColour &colour, wxBrushStyle style=wxBRUSHSTYLE_SOLID)

  Constructs a brush from a colour object and style.
- wxBrush (const wxBitmap &stippleBitmap)

  Constructs a stippled brush using a bitmap.
- wxBrush (const wxBrush &brush)

  Copy constructor, uses reference counting.
- virtual ~wxBrush ()

  Destructor.
- virtual wxColour GetColour () const

  Returns a reference to the brush colour.
- virtual wxBitmap * GetStipple () const

  Gets a pointer to the stipple bitmap.

- virtual wxBrushStyle GetStyle () const

  Returns the brush style, one of the wxBrushStyle values.
- virtual bool IsHatch () const

  Returns true if the style of the brush is any of hatched fills.
- virtual bool IsOk () const

  Returns true if the brush is initialised.
- bool IsNonTransparent () const

  Returns true if the brush is a valid non-transparent brush.
- bool IsTransparent () const

  Returns true if the brush is transparent.
- virtual void SetStipple (const wxBitmap &bitmap)

  Sets the stipple bitmap.
- virtual void SetStyle (wxBrushStyle style)

  Sets the brush style.
- bool operator!= (const wxBrush &brush) const

  Inequality operator.
- bool operator== (const wxBrush &brush) const

  Equality operator.

- virtual void SetColour (const wxColour &colour)

  Sets the brush colour using red, green and blue values.
- virtual void SetColour (unsigned char red, unsigned char green, unsigned char blue)

  Sets the brush colour using red, green and blue values.

Additional Inherited Members

21.66.2 Constructor & Destructor Documentation

wxBrush::wxBrush ( )

Default constructor.

The brush will be uninitialised, and wxBrush:IsOk() will return false.

wxBrush::wxBrush (const wxColour & *colour*, wxBrushStyle *style* = wxBRUSHSTYLE_SOLID)

Constructs a brush from a colour object and *style*.

Parameters

colour	Colour object.
style	One of the wxBrushStyle enumeration values.

wxBrush::wxBrush (const wxBitmap & *stippleBitmap*)

Constructs a stippled brush using a bitmap.

The brush style will be set to `wxBRUSHSTYLE_STIPPLE`.

wxBrush::wxBrush (const wxBrush & *brush*)

Copy constructor, uses reference counting.

virtual wxBrush::~wxBrush ( ) [virtual]

Destructor.

See Object Destruction for more info.

Remarks

Although all remaining brushes are deleted when the application exits, the application should try to clean up all brushes itself. This is because wxWidgets cannot know if a pointer to the brush object is stored in an application data structure, and there is a risk of double deletion.

21.66.3 Member Function Documentation

virtual wxColour wxBrush::GetColour ( ) const [virtual]

Returns a reference to the brush colour.

See also

SetColour()

virtual wxBitmap* wxBrush::GetStipple ( ) const [virtual]

Gets a pointer to the stipple bitmap.

If the brush does not have a wxBRUSHSTYLE_STIPPLE style, this bitmap may be non-NULL but uninitialised (i.e. wxBitmap:IsOk() returns false).

See also

SetStipple()

virtual wxBrushStyle wxBrush::GetStyle ( ) const [virtual]

Returns the brush style, one of the wxBrushStyle values.

See also

SetStyle(), SetColour(), SetStipple()

virtual bool wxBrush::IsHatch ( ) const [virtual]

Returns true if the style of the brush is any of hatched fills.

See also

GetStyle()

bool wxBrush::IsNonTransparent ( ) const

Returns true if the brush is a valid non-transparent brush.

This method returns true if the brush object is initialized and has a non-transparent style. Notice that this should be used instead of simply testing whether GetStyle() returns a style different from wxBRUSHSTYLE_TRANSPARENT if the brush may be invalid as GetStyle() would assert in this case.

See also

IsTransparent()

Since

2.9.2.

virtual bool wxBrush::IsOk ( ) const `[virtual]`

Returns true if the brush is initialised.

Notice that an uninitialized brush object can't be queried for any brush properties and all calls to the accessor methods on it will result in an assert failure.

bool wxBrush::IsTransparent ( ) const

Returns true if the brush is transparent.

A transparent brush is simply a brush with wxBRUSHSTYLE_TRANSPARENT style.

Notice that this function works even for non-initialized brushes (for which it returns false) unlike tests of the form `GetStyle() == wxBRUSHSTYLE_TRANSPARENT` which would assert if the brush is invalid.

See also

IsNonTransparent()

Since

2.9.2.

bool wxBrush::operator!= (const wxBrush & *brush*) const

Inequality operator.

See Object Comparison for more info.

bool wxBrush::operator== (const wxBrush & *brush*) const

Equality operator.

See Object Comparison for more info.

virtual void wxBrush::SetColour (const wxColour & *colour*) `[virtual]`

Sets the brush colour using red, green and blue values.

See also

GetColour()

virtual void wxBrush::SetColour (unsigned char *red,* unsigned char *green,* unsigned char *blue*) `[virtual]`

Sets the brush colour using red, green and blue values.

See also

GetColour()

virtual void wxBrush::SetStipple (const wxBitmap & *bitmap*) `[virtual]`

Sets the stipple bitmap.

Parameters

bitmap	The bitmap to use for stippling.

Remarks

The style will be set to `wxBRUSHSTYLE_STIPPLE`, unless the bitmap has a mask associated to it, in which case the style will be set to `wxBRUSHSTYLE_STIPPLE_MASK_OPAQUE`.

See also

wxBitmap

virtual void wxBrush::SetStyle (wxBrushStyle *style*) `[virtual]`

Sets the brush style.

Parameters

style	One of the wxBrushStyle values.

See also

GetStyle()

21.67 wxBrushList Class Reference

`#include <wx/brush.h>`

21.67.1 Detailed Description

A brush list is a list containing all brushes which have been created.

The application should not construct its own brush list: it should use the object pointer wxTheBrushList.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxBrush

Public Member Functions

- wxBrush * FindOrCreateBrush (const wxColour &colour, wxBrushStyle style=wxBRUSHSTYLE_SOLID)

  Finds a brush with the specified attributes and returns it, else creates a new brush, adds it to the brush list, and returns it.

21.67.2 Member Function Documentation

wxBrush* wxBrushList::FindOrCreateBrush (const wxColour & *colour,* wxBrushStyle *style* = wxBRUSHSTYLE_SOLID)

Finds a brush with the specified attributes and returns it, else creates a new brush, adds it to the brush list, and returns it.

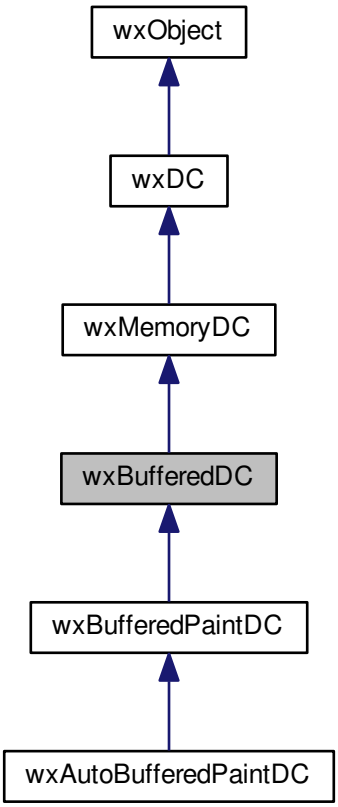
Parameters

colour	Colour object.
style	Brush style. See wxBrushStyle for a list of styles.

21.68 wxBufferedDC Class Reference

```
#include <wx/dcbuffer.h>
```

Inheritance diagram for wxBufferedDC:



21.68.1 Detailed Description

This class provides a simple way to avoid flicker: when drawing on it, everything is in fact first drawn on an in-memory buffer (a wxBitmap) and then copied to the screen, using the associated wxDC, only once, when this object is destroyed.

wxBufferedDC itself is typically associated with wxClientDC, if you want to use it in your `EVT_PAINT` handler, you should look at wxBufferedPaintDC instead.

When used like this, a valid *DC* must be specified in the constructor while the *buffer* bitmap doesn't have to be explicitly provided, by default this class will allocate the bitmap of required size itself. However using a dedicated bitmap can speed up the redrawing process by eliminating the repeated creation and destruction of a possibly big bitmap. Otherwise, wxBufferedDC can be used in the same way as any other device context.

There is another possible use for wxBufferedDC is to use it to maintain a backing store for the window contents. In this case, the associated *DC* may be NULL but a valid backing store bitmap should be specified.

Finally, please note that GTK+ 2.0 as well as OS X provide double buffering themselves natively. You can either use wxWindow::IsDoubleBuffered() to determine whether you need to use buffering or not, or use wxAutoBufferedPaintDC to avoid needless double buffering on the systems which already do it automatically.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxMemoryDC, wxBufferedPaintDC, wxAutoBufferedPaintDC

Public Member Functions

- wxBufferedDC ()

  Default constructor.
- wxBufferedDC (wxDC *dc, const wxSize &area, int style=wxBUFFER_CLIENT_AREA)

  Creates a buffer for the provided dc.
- wxBufferedDC (wxDC *dc, wxBitmap &buffer=wxNullBitmap, int style=wxBUFFER_CLIENT_AREA)

  Creates a buffer for the provided dc.
- virtual ~wxBufferedDC ()

  Copies everything drawn on the DC so far to the underlying DC associated with this object, if any.
- void UnMask ()

  Blits the buffer to the dc, and detaches the dc from the buffer (so it can be effectively used once only).
- void SetStyle (int style)

  Set the style.
- int GetStyle () const

  Get the style.

- void Init (wxDC *dc, const wxSize &area, int style=wxBUFFER_CLIENT_AREA)

  Initializes the object created using the default constructor.
- void Init (wxDC *dc, wxBitmap &buffer=wxNullBitmap, int style=wxBUFFER_CLIENT_AREA)

  Initializes the object created using the default constructor.

Additional Inherited Members

21.68.2 Constructor & Destructor Documentation

wxBufferedDC::wxBufferedDC ( )

Default constructor.

You must call one of the Init() methods later in order to use the device context.

wxBufferedDC::wxBufferedDC (wxDC * *dc,* const wxSize & *area,* int *style* = wxBUFFER_CLIENT_AREA)

Creates a buffer for the provided *dc*.

Init() must not be called when using this constructor.

Parameters

dc	The underlying DC: everything drawn to this object will be flushed to this DC when this object is destroyed. You may pass NULL in order to just initialize the buffer, and not flush it.

area	The size of the bitmap to be used for buffering (this bitmap is created internally when it is not given explicitly).
style	wxBUFFER_CLIENT_AREA to indicate that just the client area of the window is buffered, or wxBUFFER_VIRTUAL_AREA to indicate that the buffer bitmap covers the virtual area.

wxBufferedDC::wxBufferedDC (wxDC * *dc*, wxBitmap & *buffer* = wxNullBitmap, int *style* = wxBUFFER_CLIENT_AREA)

Creates a buffer for the provided dc.

Init() must not be called when using this constructor.

Parameters

dc	The underlying DC: everything drawn to this object will be flushed to this DC when this object is destroyed. You may pass NULL in order to just initialize the buffer, and not flush it.
buffer	Explicitly provided bitmap to be used for buffering: this is the most efficient solution as the bitmap doesn't have to be recreated each time but it also requires more memory as the bitmap is never freed. The bitmap should have appropriate size, anything drawn outside of its bounds is clipped.
style	wxBUFFER_CLIENT_AREA to indicate that just the client area of the window is buffered, or wxBUFFER_VIRTUAL_AREA to indicate that the buffer bitmap covers the virtual area.

virtual wxBufferedDC::~wxBufferedDC ( ) [virtual]

Copies everything drawn on the DC so far to the underlying DC associated with this object, if any.

21.68.3 Member Function Documentation

int wxBufferedDC::GetStyle ( ) const

Get the style.

void wxBufferedDC::Init (wxDC * *dc*, const wxSize & *area*, int *style* = wxBUFFER_CLIENT_AREA)

Initializes the object created using the default constructor.

Please see the constructors for parameter details.

void wxBufferedDC::Init (wxDC * *dc*, wxBitmap & *buffer* = wxNullBitmap, int *style* = wxBUFFER_CLIENT_AREA)

Initializes the object created using the default constructor.

Please see the constructors for parameter details.

void wxBufferedDC::SetStyle (int *style*)

Set the style.

void wxBufferedDC::UnMask ( )

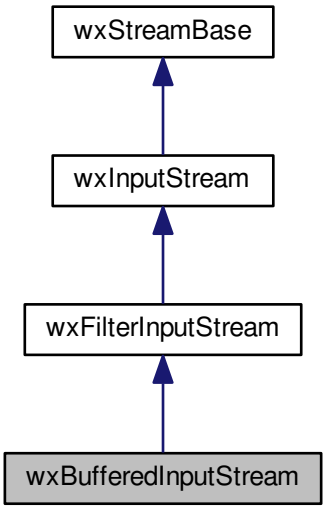
Blits the buffer to the dc, and detaches the dc from the buffer (so it can be effectively used once only).

Usually only called in the destructor or by the destructor of derived classes if the BufferedDC must blit before the derived class (which may own the dc it's blitting to) is destroyed.

21.69 wxBufferedInputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxBufferedInputStream:



21.69.1 Detailed Description

This stream acts as a cache.

It caches the bytes read from the specified input stream (see wxFilterInputStream). It uses wxStreamBuffer and sets the default in-buffer size to 1024 bytes. This class may not be used without some other stream to read the data from (such as a file stream or a memory stream).

Library: wxBase

Category: Streams

See also

wxStreamBuffer, wxInputStream, wxBufferedOutputStream

Public Member Functions

- wxBufferedInputStream (wxInputStream &stream, wxStreamBuffer *buffer=NULL)

  Constructor using the provided buffer or default.

- wxBufferedInputStream (wxInputStream &stream, size_t bufsize)

  Constructor allowing to specify the size of the buffer.

- virtual ~wxBufferedInputStream ()

  Destructor.

Additional Inherited Members

21.69.2 Constructor & Destructor Documentation

wxBufferedInputStream::wxBufferedInputStream (wxInputStream & *stream*, wxStreamBuffer * *buffer* = NULL)

Constructor using the provided buffer or default.

Parameters

stream	The associated low-level stream.
buffer	The buffer to use if non-NULL. Notice that the ownership of this buffer is taken by the stream, i.e. it will delete it. If this parameter is NULL a default 1KB buffer is used.

wxBufferedInputStream::wxBufferedInputStream (wxInputStream & *stream*, size_t *bufsize*)

Constructor allowing to specify the size of the buffer.

This is just a more convenient alternative to creating a wxStreamBuffer of the given size and using the other overloaded constructor of this class.

Parameters

stream	The associated low-level stream.
bufsize	The size of the buffer, in bytes.

Since

2.9.0

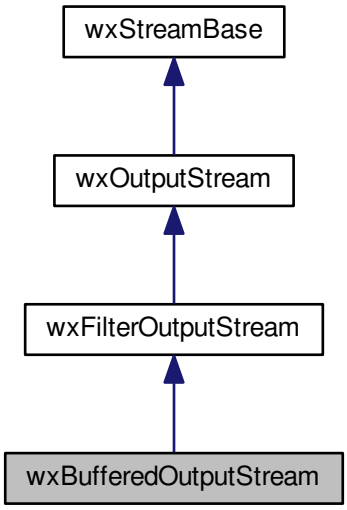
virtual wxBufferedInputStream::~wxBufferedInputStream ( ) [virtual]

Destructor.

21.70 wxBufferedOutputStream Class Reference

```
#include <wx/stream.h>
```

Inheritance diagram for wxBufferedOutputStream:



21.70.1 Detailed Description

This stream acts as a cache.

It caches the bytes to be written to the specified output stream (See wxFilterOutputStream). The data is only written when the cache is full, when the buffered stream is destroyed or when calling SeekO().

This class may not be used without some other stream to write the data to (such as a file stream or a memory stream).

Library: wxBase

Category: Streams

See also

wxStreamBuffer, wxOutputStream

Public Member Functions

- wxBufferedOutputStream (wxOutputStream &stream, wxStreamBuffer *buffer=NULL)

  Constructor using the provided buffer or default.
- wxBufferedOutputStream (wxOutputStream &stream, size_t bufsize)

  Constructor allowing to specify the size of the buffer.
- virtual ~wxBufferedOutputStream ()

  Destructor.
- virtual wxFileOffset SeekO (wxFileOffset pos, wxSeekMode mode=wxFromStart)

  Calls Sync() and changes the stream position.

- virtual void Sync ()

  Flushes the buffer and calls Sync() on the parent stream.

Additional Inherited Members

21.70.2 Constructor & Destructor Documentation

wxBufferedOutputStream::wxBufferedOutputStream (wxOutputStream & *stream*, wxStreamBuffer * *buffer* = NULL)

Constructor using the provided buffer or default.

Parameters

stream	The associated low-level stream.
buffer	The buffer to use if non-NULL. Notice that the ownership of this buffer is taken by the stream, i.e. it will delete it. If this parameter is NULL a default 1KB buffer is used.

wxBufferedOutputStream::wxBufferedOutputStream (wxOutputStream & *stream*, size_t *bufsize*)

Constructor allowing to specify the size of the buffer.

This is just a more convenient alternative to creating a wxStreamBuffer of the given size and using the other overloaded constructor of this class.

Parameters

stream	The associated low-level stream.
bufsize	The size of the buffer, in bytes.

Since

2.9.0

virtual wxBufferedOutputStream::~wxBufferedOutputStream ( ) [virtual]

Destructor.

Calls Sync() and destroys the internal buffer.

21.70.3 Member Function Documentation

virtual wxFileOffset wxBufferedOutputStream::SeekO (wxFileOffset *pos*, wxSeekMode *mode* = wxFromStart) [virtual]

Calls Sync() and changes the stream position.

Reimplemented from wxOutputStream.

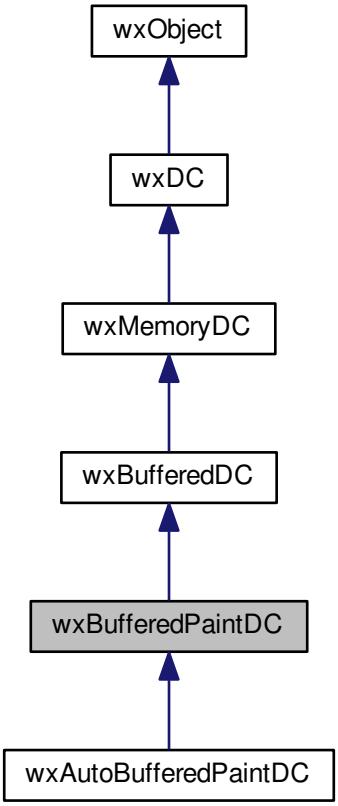
virtual void wxBufferedOutputStream::Sync ( ) [virtual]

Flushes the buffer and calls Sync() on the parent stream.

21.71 wxBufferedPaintDC Class Reference

`#include <wx/dcbuffer.h>`

Inheritance diagram for wxBufferedPaintDC:



21.71.1 Detailed Description

This is a subclass of wxBufferedDC which can be used inside of an `EVT_PAINT()` event handler to achieve double-buffered drawing.

Just use this class instead of wxPaintDC and make sure wxWindow::SetBackgroundStyle() is called with wxBG_↩ STYLE_PAINT somewhere in the class initialization code, and that's all you have to do to (mostly) avoid flicker. The only thing to watch out for is that if you are using this class together with wxScrolled, you probably do **not** want to call wxScrolled::PrepareDC() on it as it already does this internally for the real underlying wxPaintDC.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxBufferedDC, wxAutoBufferedPaintDC, wxPaintDC

Public Member Functions

- virtual ~wxBufferedPaintDC ()

  Copies everything drawn on the DC so far to the window associated with this object, using a wxPaintDC.

- wxBufferedPaintDC (wxWindow *window, wxBitmap &buffer, int style=wxBUFFER_CLIENT_AREA)

  As with wxBufferedDC, you may either provide the bitmap to be used for buffering or let this object create one internally (in the latter case, the size of the client part of the window is used).

- wxBufferedPaintDC (wxWindow *window, int style=wxBUFFER_CLIENT_AREA)

  As with wxBufferedDC, you may either provide the bitmap to be used for buffering or let this object create one internally (in the latter case, the size of the client part of the window is used).

Additional Inherited Members

21.71.2 Constructor & Destructor Documentation

wxBufferedPaintDC::wxBufferedPaintDC (wxWindow * *window,* wxBitmap & *buffer,* int *style* = wxBUFFER_CLIENT_AREA)

As with wxBufferedDC, you may either provide the bitmap to be used for buffering or let this object create one internally (in the latter case, the size of the client part of the window is used).

Pass wxBUFFER_CLIENT_AREA for the *style* parameter to indicate that just the client area of the window is buffered, or wxBUFFER_VIRTUAL_AREA to indicate that the buffer bitmap covers the virtual area.

wxBufferedPaintDC::wxBufferedPaintDC (wxWindow * *window,* int *style* = wxBUFFER_CLIENT_AREA)

As with wxBufferedDC, you may either provide the bitmap to be used for buffering or let this object create one internally (in the latter case, the size of the client part of the window is used).

Pass wxBUFFER_CLIENT_AREA for the *style* parameter to indicate that just the client area of the window is buffered, or wxBUFFER_VIRTUAL_AREA to indicate that the buffer bitmap covers the virtual area.

virtual wxBufferedPaintDC::~wxBufferedPaintDC ( ) [virtual]

Copies everything drawn on the DC so far to the window associated with this object, using a wxPaintDC.

21.72 wxBusyCursor Class Reference

```
#include <wx/utils.h>
```

21.72.1 Detailed Description

This class makes it easy to tell your user that the program is temporarily busy.

Just create a wxBusyCursor object on the stack, and within the current scope, the hourglass will be shown.

For example:

```
wxBusyCursor wait;

for (int i = 0; i < 100000; i++)
    DoACalculation();
```

It works by calling wxBeginBusyCursor() in the constructor, and wxEndBusyCursor() in the destructor.

Library: wxCore

Category: Miscellaneous

See also

wxBeginBusyCursor(), wxEndBusyCursor(), wxWindowDisabler

Public Member Functions

- wxBusyCursor (const wxCursor *cursor=wxHOURGLASS_CURSOR)

  Constructs a busy cursor object, calling wxBeginBusyCursor().
- ~wxBusyCursor ()

  Destroys the busy cursor object, calling wxEndBusyCursor().

21.72.2 Constructor & Destructor Documentation

wxBusyCursor::wxBusyCursor (const wxCursor * *cursor* = wxHOURGLASS_CURSOR)

Constructs a busy cursor object, calling wxBeginBusyCursor().

wxBusyCursor::~wxBusyCursor ( )

Destroys the busy cursor object, calling wxEndBusyCursor().

21.73 wxBusyInfo Class Reference

```
#include <wx/busyinfo.h>
```

21.73.1 Detailed Description

This class makes it easy to tell your user that the program is temporarily busy.

Normally the main thread should always return to the main loop to continue dispatching events as quickly as possible, hence this class shouldn't be needed. However if the main thread does need to block, this class provides a simple way to at least show this to the user: just create a wxBusyInfo object on the stack, and within the current scope, a message window will be shown.

For example:

```
wxBusyInfo wait("Please wait, working...");

for (int i = 0; i < 100000; i++)
{
    DoACalculation();
}
```

It works by creating a window in the constructor, and deleting it in the destructor.

This window is rather plain by default but can be customized by passing wxBusyInfo constructor an object of wxBusyInfoFlags class instead of a simple message. Here is an example from the page_samples_dialogs:

```
wxBusyInfo info
    (
        wxBusyInfoFlags()
            .Parent(this)
            .Icon(wxArtProvider::GetIcon(wxART_PRINT,
                                         wxART_OTHER, wxSize(128, 128)))
            .Title("<b>Printing your document</b>")
            .Text("Please wait...")
            .Foreground(*wxWHITE)
            .Background(*wxBLACK)
            .Transparency(4*wxALPHA_OPAQUE/5)
    );
```

showing that separate title and text can be set, and that simple markup (wxControl::SetLabelMarkup()) can be used in them, and that it's also possible to add an icon and customize the colours and transparency of the window.

You may also want to call wxTheApp->Yield() to refresh the window periodically (in case it had been obscured by other windows, for example) like this:

```
wxWindowDisabler disableAll;
wxBusyInfo wait("Please wait, working...");

for (int i = 0; i < 100000; i++)
{
    DoACalculation();

    if ( !(i % 1000) )
        wxTheApp->Yield();
}
```

but take care to not cause undesirable reentrancies when doing it (see wxApp::Yield for more details). The simplest way to do it is to use wxWindowDisabler class as illustrated in the above example.

Note that a wxBusyInfo is always built with the `wxSTAY_ON_TOP` window style (see wxFrame window styles for more info).

Library: wxCore

Category: Common Dialogs

Public Member Functions

- wxBusyInfo (const wxBusyInfoFlags &flags)

  General constructor.
- wxBusyInfo (const wxString &msg, wxWindow *parent=NULL)

  Simple constructor specifying only the message and the parent.
- virtual ~wxBusyInfo ()

  Hides and closes the window containing the information text.

21.73.2 Constructor & Destructor Documentation

wxBusyInfo::wxBusyInfo (const wxBusyInfoFlags & *flags*)

General constructor.

This constructor allows to specify all supported attributes by calling the appropriate methods on wxBusyInfoFlags object passed to it as parameter. All of them are optional but usually at least the message should be specified.

Since

3.1.0

wxBusyInfo::wxBusyInfo (const wxString & *msg*, wxWindow * *parent* = NULL)

Simple constructor specifying only the message and the parent.

This constructs a busy info window as child of *parent* and displays *msg* in it. It is exactly equivalent to using

```
wxBusyInfo(wxBusyInfoFlags().Parent(parent).Label(message))
```

Note

If *parent* is not NULL you must ensure that it is not closed while the busy info is shown.

virtual wxBusyInfo::~wxBusyInfo ( ) [virtual]

Hides and closes the window containing the information text.

21.74 wxBusyInfoFlags Class Reference

```
#include <wx/busyinfo.h>
```

21.74.1 Detailed Description

Parameters for wxBusyInfo.

This class exists only in order to make passing attributes to wxBusyInfo constructor easier and the code doing it more readable.

All methods of this class return the reference to the object on which they are called, making it possible to chain them together, e.g. typically you would just create a temporary wxBusyInfoFlags object and then call the methods corresponding to the attributes you want to set, before finally passing the result to wxBusyInfo constructor, e.g.:

```
wxBusyInfo info
    (
        wxBusyInfoFlags()
            .Parent(window)
            .Icon(icon)
            .Title("Some text")
            .Text("Some more text")
            .Foreground(wxColour(...))
            .Background(wxColour(...))
    );
```

Since

3.1.0

Public Member Functions

- wxBusyInfoFlags ()

  Default constructor initializes all attributes to default values.
- wxBusyInfoFlags & Parent (wxWindow *parent)

  Sets the parent for wxBusyInfo.

- wxBusyInfoFlags & Icon (const wxIcon &icon)

  Sets the icon to show in wxBusyInfo.
- wxBusyInfoFlags & Title (const wxString &title)

  Sets the title, shown prominently in wxBusyInfo window.
- wxBusyInfoFlags & Text (const wxString &text)

  Sets the more detailed text, shown under the title, if any.
- wxBusyInfoFlags & Label (const wxString &label)

  Same as Text() but doesn't interpret the string as containing markup.
- wxBusyInfoFlags & Foreground (const wxColour &foreground)

  Sets the foreground colour of the title and text strings.
- wxBusyInfoFlags & Background (const wxColour &background)

  Sets the background colour of wxBusyInfo window.
- wxBusyInfoFlags & Transparency (wxByte alpha)

  Sets the transparency of wxBusyInfo window.

21.74.2 Constructor & Destructor Documentation

wxBusyInfoFlags::wxBusyInfoFlags ( )

Default constructor initializes all attributes to default values.

Call the other methods to really fill in the object.

21.74.3 Member Function Documentation

wxBusyInfoFlags& wxBusyInfoFlags::Background (const wxColour & *background*)

Sets the background colour of wxBusyInfo window.

wxBusyInfoFlags& wxBusyInfoFlags::Foreground (const wxColour & *foreground*)

Sets the foreground colour of the title and text strings.

wxBusyInfoFlags& wxBusyInfoFlags::Icon (const wxIcon & *icon*)

Sets the icon to show in wxBusyInfo.

wxBusyInfoFlags& wxBusyInfoFlags::Label (const wxString & *label*)

Same as Text() but doesn't interpret the string as containing markup.

This method should be used if the text shown in wxBusyInfo comes from external source and so may contain characters having special meaning in simple markup, e.g. '<'.

wxBusyInfoFlags& wxBusyInfoFlags::Parent (wxWindow * *parent*)

Sets the parent for wxBusyInfo.

wxBusyInfoFlags& wxBusyInfoFlags::Text (const wxString & *text*)

Sets the more detailed text, shown under the title, if any.

The *text* string may contain markup as described in wxControl::SetLabelMarkup().

wxBusyInfoFlags& wxBusyInfoFlags::Title (const wxString & *title*)

Sets the title, shown prominently in wxBusyInfo window.

The *title* string may contain markup as described in wxControl::SetLabelMarkup().

wxBusyInfoFlags& wxBusyInfoFlags::Transparency (wxByte *alpha*)

Sets the transparency of wxBusyInfo window.

Parameters

alpha	Value in wxALPHA_TRANSPARENT (0) to wxALPHA_OPAQUE (255) range.

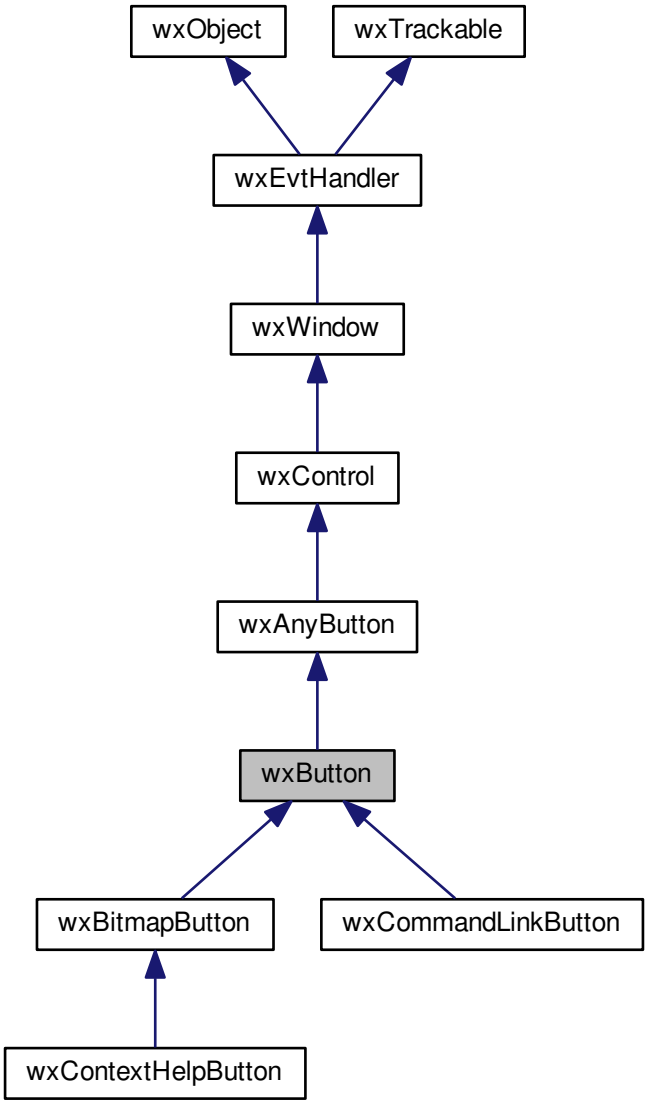
See also

wxTopLevelWindow::SetTransparent()

21.75 wxButton Class Reference

```
#include <wx/button.h>
```

Inheritance diagram for wxButton:



21.75.1 Detailed Description

A button is a control that contains a text string, and is one of the most common elements of a GUI.

It may be placed on a dialog box or on a wxPanel panel, or indeed on almost any other window.

By default, i.e. if none of the alignment styles are specified, the label is centered both horizontally and vertically. If the button has both a label and a bitmap, the alignment styles above specify the location of the rectangle combining both the label and the bitmap and the bitmap position set with wxButton::SetBitmapPosition() defines the relative position of the bitmap with respect to the label (however currently non-default alignment combinations are not implemented on all platforms).

Since version 2.9.1 wxButton supports showing both text and an image (currently only when using wxMSW, wxGTK or wxOSX/Cocoa ports), see SetBitmap() and SetBitmapLabel(), SetBitmapDisabled() &c methods. In the previous

wxWidgets versions this functionality was only available in (the now trivial) wxBitmapButton class which was only capable of showing an image without text.

A button may have either a single image for all states or different images for the following states (different images are not currently supported under OS X where the normal image is used for all states):

- **normal:** the default state
- **disabled:** bitmap shown when the button is disabled.
- **pressed:** bitmap shown when the button is pushed (e.g. while the user keeps the mouse button pressed on it)
- **focus:** bitmap shown when the button has keyboard focus (but is not pressed as in this case the button is in the pressed state)
- **current:** bitmap shown when the mouse is over the button (but it is not pressed although it may have focus). Notice that if current bitmap is not specified but the current platform UI uses hover images for the buttons (such as Windows XP or GTK+), then the focus bitmap is used for hover state as well. This makes it possible to set focus bitmap only to get reasonably good behaviour on all platforms.

All of the bitmaps must be of the same size and the normal bitmap must be set first (to a valid bitmap), before setting any other ones. Also, if the size of the bitmaps is changed later, you need to change the size of the normal bitmap before setting any other bitmaps with the new size (and you do need to reset all of them as their original values can be lost when the normal bitmap size changes).

The position of the image inside the button be configured using SetBitmapPosition(). By default the image is on the left of the text.

Please also notice that GTK+ uses a global setting called `gtk-button-images` to determine if the images should be shown in the buttons at all. If it is off (which is the case in e.g. Gnome 2.28 by default), no images will be shown, consistently with the native behaviour.

Styles

This class supports the following styles:

- wxBU_LEFT: Left-justifies the label. Windows and GTK+ only.
- wxBU_TOP: Aligns the label to the top of the button. Windows and GTK+ only.
- wxBU_RIGHT: Right-justifies the bitmap label. Windows and GTK+ only.
- wxBU_BOTTOM: Aligns the label to the bottom of the button. Windows and GTK+ only.
- wxBU_EXACTFIT: By default, all buttons are made of at least the standard button size, even if their contents is small enough to fit into a smaller size. This is done for consistency as most platforms use buttons of the same size in the native dialogs, but can be overridden by specifying this flag. If it is given, the button will be made just big enough for its contents. Notice that under MSW the button will still have at least the standard height, even with this style, if it has a non-empty label.
- wxBU_NOTEXT: Disables the display of the text label in the button even if it has one or its id is one of the standard stock ids with an associated label: without using this style a button which is only supposed to show a bitmap but uses a standard id would display a label too.
- wxBORDER_NONE: Creates a button without border. This is currently implemented in MSW, GTK2 and OSX/Cocoa and OSX/Carbon ports but in the latter only applies to buttons with bitmaps and using bitmap of one of the standard sizes only, namely 128∗128, 48∗48, 24∗24 or 16∗16. In all the other cases wxBORDER_NONE is ignored under OSX/Carbon (these restrictions don't exist in OSX/Cocoa however).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_BUTTON(id, func): Process a `wxEVT_BUTTON` event, when the button is clicked.

Library: wxCore

Category: Controls

See also

wxBitmapButton

Public Member Functions

- wxButton ()

  Default ctor.
- wxButton (wxWindow *parent, wxWindowID id, const wxString &label=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)

  Constructor, creating and showing a button.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)

  Button creation function for two-step creation.
- bool GetAuthNeeded () const

  Returns true if an authentication needed symbol is displayed on the button.
- wxString GetLabel () const

  Returns the string label for the button.
- void SetAuthNeeded (bool needed=true)

  Sets whether an authentication needed symbol should be displayed on the button.
- virtual wxWindow * SetDefault ()

  This sets the button to be the default item in its top-level window (e.g.
- void SetLabel (const wxString &label)

  Sets the string label for the button.

Static Public Member Functions

- static wxSize GetDefaultSize ()

  Returns the default size for the buttons.

Additional Inherited Members

21.75.2 Constructor & Destructor Documentation

wxButton::wxButton ( )

Default ctor.

wxButton::wxButton (wxWindow * *parent*, wxWindowID *id*, const wxString & *label* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxButtonNameStr)

Constructor, creating and showing a button.

The preferred way to create standard buttons is to use default value of *label*. If no label is supplied and *id* is one of standard IDs from this list, a standard label will be used. In other words, if you use a predefined `wxID_XXX` constant, just omit the label completely rather than specifying it. In particular, help buttons (the ones with *id* of `wxID_HELP`) under Mac OS X can't display any label at all and while wxButton will detect if the standard "Help" label is used and ignore it, using any other label will prevent the button from correctly appearing as a help button and so should be avoided.

In addition to that, the button will be decorated with stock icons under GTK+ 2.

Parameters

parent	Parent window. Must not be NULL.
id	Button identifier. A value of `wxID_ANY` indicates a default value.
label	Text to be displayed on the button.
pos	Button position.
size	Button size. If the default size is specified then the button is sized appropriately for the text.
style	Window style. See wxButton class description.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

21.75.3 Member Function Documentation

bool wxButton::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxButtonNameStr)

Button creation function for two-step creation.

For more details, see wxButton().

bool wxButton::GetAuthNeeded ( ) const

Returns true if an authentication needed symbol is displayed on the button.

Remarks

This method always returns false if the platform is not Windows Vista or newer.

See also

SetAuthNeeded()

Since

2.9.1

static wxSize wxButton::GetDefaultSize ( ) [static]

Returns the default size for the buttons.

It is advised to make all the dialog buttons of the same size and this function allows to retrieve the (platform and current font dependent size) which should be the best suited for this.

wxString wxButton::GetLabel ( ) const [virtual]

Returns the string label for the button.

See also

SetLabel()

Reimplemented from wxWindow.

Reimplemented in wxCommandLinkButton.

void wxButton::SetAuthNeeded (bool *needed* = true)

Sets whether an authentication needed symbol should be displayed on the button.

Remarks

This method doesn't do anything if the platform is not Windows Vista or newer.

See also

GetAuthNeeded()

Since

2.9.1

virtual wxWindow* wxButton::SetDefault ( ) [virtual]

This sets the button to be the default item in its top-level window (e.g.

the panel or the dialog box containing it).

As normal, pressing return causes the default button to be depressed when the return key is pressed.

See also wxWindow::SetFocus() which sets the keyboard focus for windows and text panel items, and wxTopLevelWindow::SetDefaultItem().

Remarks

Under Windows, only dialog box buttons respond to this function.

Returns

the old default item (possibly NULL)

void wxButton::SetLabel (const wxString & *label*) [virtual]

Sets the string label for the button.

Parameters

label	The label to set.

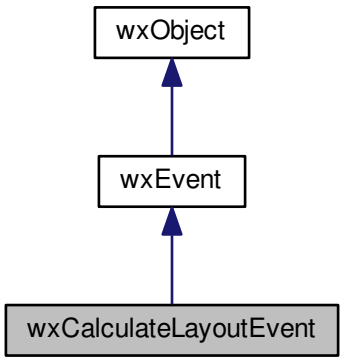
Reimplemented from wxWindow.

Reimplemented in wxCommandLinkButton.

21.76 wxCalculateLayoutEvent Class Reference

`#include <wx/laywin.h>`

Inheritance diagram for wxCalculateLayoutEvent:



21.76.1 Detailed Description

This event is sent by wxLayoutAlgorithm to calculate the amount of the remaining client area that the window should occupy.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCalculateLayoutEvent& event)

Event macros:

- EVT_CALCULATE_LAYOUT(func): Process a `wxEVT_CALCULATE_LAYOUT` event, which asks the window to take a 'bite' out of a rectangle provided by the algorithm.

Library: wxAdvanced

Category: Events

See also

wxQueryLayoutInfoEvent, wxSashLayoutWindow, wxLayoutAlgorithm.

Public Member Functions

- wxCalculateLayoutEvent (wxWindowID id=0)

  Constructor.
- int GetFlags () const

  Returns the flags associated with this event.
- wxRect GetRect () const

  Before the event handler is entered, returns the remaining parent client area that the window could occupy.
- void SetFlags (int flags)

  Sets the flags associated with this event.
- void SetRect (const wxRect &rect)

  Call this to specify the new remaining parent client area, after the space occupied by the window has been subtracted.

Additional Inherited Members

21.76.2 Constructor & Destructor Documentation

wxCalculateLayoutEvent::wxCalculateLayoutEvent (wxWindowID *id* = 0)

Constructor.

21.76.3 Member Function Documentation

int wxCalculateLayoutEvent::GetFlags ( ) const

Returns the flags associated with this event.

Not currently used.

wxRect wxCalculateLayoutEvent::GetRect ( ) const

Before the event handler is entered, returns the remaining parent client area that the window could occupy.

When the event handler returns, this should contain the remaining parent client rectangle, after the event handler has subtracted the area that its window occupies.

void wxCalculateLayoutEvent::SetFlags (int *flags*)

Sets the flags associated with this event.

Not currently used.

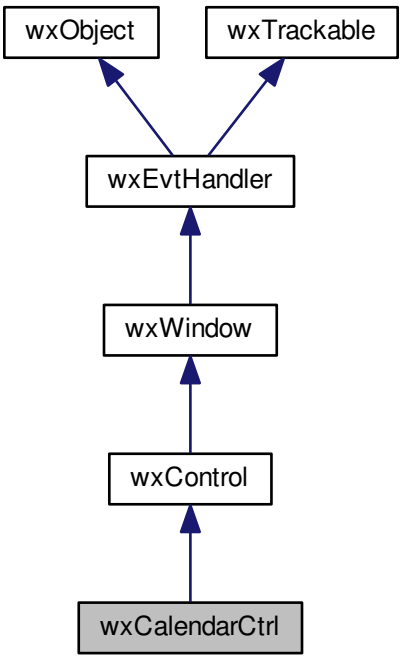
void wxCalculateLayoutEvent::SetRect (const wxRect & *rect*)

Call this to specify the new remaining parent client area, after the space occupied by the window has been subtracted.

21.77 wxCalendarCtrl Class Reference

`#include <wx/calctrl.h>`

Inheritance diagram for wxCalendarCtrl:



21.77.1 Detailed Description

The calendar control allows the user to pick a date.

The user can move the current selection using the keyboard and select the date (generating `EVT_CALENDAR` event) by pressing `<Return>` or double clicking it.

Generic calendar has advanced possibilities for the customization of its display, described below. If you want to use these possibilities on every platform, use wxGenericCalendarCtrl instead of wxCalendarCtrl.

All global settings (such as colours and fonts used) can, of course, be changed. But also, the display style for each day in the month can be set independently using wxCalendarDateAttr class.

An item without custom attributes is drawn with the default colours and font and without border, but setting custom attributes with SetAttr() allows to modify its appearance. Just create a custom attribute object and set it for the day you want to be displayed specially (note that the control will take ownership of the pointer, i.e. it will delete it itself). A day may be marked as being a holiday, even if it is not recognized as one by wxDateTime using the wxCalendarDateAttr::SetHoliday() method.

As the attributes are specified for each day, they may change when the month is changed, so you will often want to update them in `EVT_CALENDAR_PAGE_CHANGED` event handler.

Styles

This class supports the following styles:

- wxCAL_SUNDAY_FIRST: Show Sunday as the first day in the week (not in wxGTK)
- wxCAL_MONDAY_FIRST: Show Monday as the first day in the week (not in wxGTK)
- wxCAL_SHOW_HOLIDAYS: Highlight holidays in the calendar (only generic)
- wxCAL_NO_YEAR_CHANGE: Disable the year changing (deprecated, only generic)
- wxCAL_NO_MONTH_CHANGE: Disable the month (and, implicitly, the year) changing
- wxCAL_SHOW_SURROUNDING_WEEKS: Show the neighbouring weeks in the previous and next months (only generic, always on for the native controls)
- wxCAL_SEQUENTIAL_MONTH_SELECTION: Use alternative, more compact, style for the month and year selection controls. (only generic)
- wxCAL_SHOW_WEEK_NUMBERS: Show week numbers on the left side of the calendar. (not in generic)

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCalendarEvent& event)

Event macros for events emitted by this class:

- EVT_CALENDAR(id, func): A day was double clicked in the calendar.
- EVT_CALENDAR_SEL_CHANGED(id, func): The selected date changed.
- EVT_CALENDAR_PAGE_CHANGED(id, func): The selected month (and/or year) changed.
- EVT_CALENDAR_WEEKDAY_CLICKED(id, func): User clicked on the week day header (only generic).
- EVT_CALENDAR_WEEK_CLICKED(id, func): User clicked on the week of the year number (only generic).

Note

Changing the selected date will trigger an EVT_CALENDAR_DAY, MONTH or YEAR event as well as an EVT_CALENDAR_SEL_CHANGED event.

Library: wxAdvanced

Category: Controls

Implementations: native under wxGTK, wxMSW ports; a generic implementation is used elsewhere.

See also

Calendar Sample, wxCalendarDateAttr, wxCalendarEvent, wxDatePickerCtrl

Public Member Functions

- wxCalendarCtrl ()

  Default constructor.
- wxCalendarCtrl (wxWindow *parent, wxWindowID id, const wxDateTime &date=wxDefaultDateTime, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCAL_SHOW_HOLIDAYS, const wxString &name=wxCalendarNameStr)

  Does the same as Create() method.
- ~wxCalendarCtrl ()

  Destroys the control.
- bool Create (wxWindow *parent, wxWindowID id, const wxDateTime &date=wxDefaultDateTime, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCAL_SHOW_HOLIDAYS, const wxString &name=wxCalendarNameStr)

  Creates the control.
- virtual void EnableHolidayDisplay (bool display=true)

  This function should be used instead of changing `wxCAL_SHOW_HOLIDAYS` style bit directly.
- virtual bool EnableMonthChange (bool enable=true)

  This function should be used instead of changing `wxCAL_NO_MONTH_CHANGE` style bit.
- virtual void EnableYearChange (bool enable=true)
- virtual wxCalendarDateAttr * GetAttr (size_t day) const

  Returns the attribute for the given date (should be in the range 1...31).
- virtual wxDateTime GetDate () const

  Gets the currently selected date.
- virtual const wxColour & GetHeaderColourBg () const

  Gets the background colour of the header part of the calendar window.
- virtual const wxColour & GetHeaderColourFg () const

  Gets the foreground colour of the header part of the calendar window.
- virtual const wxColour & GetHighlightColourBg () const

  Gets the background highlight colour.
- virtual const wxColour & GetHighlightColourFg () const

  Gets the foreground highlight colour.
- virtual const wxColour & GetHolidayColourBg () const

  Return the background colour currently used for holiday highlighting.
- virtual const wxColour & GetHolidayColourFg () const

  Return the foreground colour currently used for holiday highlighting.
- virtual wxCalendarHitTestResult HitTest (const wxPoint &pos, wxDateTime *date=NULL, wxDateTime::WeekDay *wd=NULL)

  Returns one of wxCalendarHitTestResult constants and fills either date or wd pointer with the corresponding value depending on the hit test code.
- virtual void ResetAttr (size_t day)

  Clears any attributes associated with the given day (in the range 1...31).
- virtual void SetAttr (size_t day, wxCalendarDateAttr *attr)

  Associates the attribute with the specified date (in the range 1...31).
- virtual bool SetDate (const wxDateTime &date)

  Sets the current date.
- virtual void SetHeaderColours (const wxColour &colFg, const wxColour &colBg)

  Set the colours used for painting the weekdays at the top of the control.
- virtual void SetHighlightColours (const wxColour &colFg, const wxColour &colBg)

  Set the colours to be used for highlighting the currently selected date.
- virtual void SetHoliday (size_t day)

  Marks the specified day as being a holiday in the current month.
- virtual void SetHolidayColours (const wxColour &colFg, const wxColour &colBg)

Sets the colours to be used for the holidays highlighting.

- virtual void Mark (size_t day, bool mark)

  Mark or unmark the day.

Date Range Functions

- virtual bool SetDateRange (const wxDateTime &lowerdate=wxDefaultDateTime, const wxDateTime &upperdate=wxDefaultDateTime)

  Restrict the dates that can be selected in the control to the specified range.
- virtual bool GetDateRange (wxDateTime *lowerdate, wxDateTime *upperdate) const

  Returns the limits currently being used.

Additional Inherited Members

21.77.2 Constructor & Destructor Documentation

wxCalendarCtrl::wxCalendarCtrl ( )

Default constructor.

wxCalendarCtrl::wxCalendarCtrl (wxWindow * *parent*, wxWindowID *id*, const wxDateTime & *date* = wxDefaultDateTime, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCAL_SHOW_HOLIDAYS, const wxString & *name* = `wxCalendarNameStr`)

Does the same as Create() method.

wxCalendarCtrl::~wxCalendarCtrl ( )

Destroys the control.

21.77.3 Member Function Documentation

bool wxCalendarCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxDateTime & *date* = wxDefaultDateTime, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCAL_SHOW_HOLIDAYS, const wxString & *name* = `wxCalendarNameStr`)

Creates the control.

See wxWindow::wxWindow() for the meaning of the parameters and the control overview for the possible styles.

virtual void wxCalendarCtrl::EnableHolidayDisplay (bool *display* = `true`) `[virtual]`

This function should be used instead of changing `wxCAL_SHOW_HOLIDAYS` style bit directly.

It enables or disables the special highlighting of the holidays.

virtual bool wxCalendarCtrl::EnableMonthChange (bool *enable* = `true`) `[virtual]`

This function should be used instead of changing `wxCAL_NO_MONTH_CHANGE` style bit.

It allows or disallows the user to change the month interactively. Note that if the month cannot be changed, the year cannot be changed neither.

Returns

true if the value of this option really changed or false if it was already set to the requested value.

virtual void wxCalendarCtrl::EnableYearChange (bool *enable* = `true`) `[virtual]`

Deprecated

This function should be used instead of changing `wxCAL_NO_YEAR_CHANGE` style bit directly. It allows or disallows the user to change the year interactively. Only in generic wxCalendarCtrl.

virtual wxCalendarDateAttr* wxCalendarCtrl::GetAttr (size_t *day*) const `[virtual]`

Returns the attribute for the given date (should be in the range 1...31).

The returned pointer may be NULL. Only in generic wxCalendarCtrl.

virtual wxDateTime wxCalendarCtrl::GetDate ( ) const `[virtual]`

Gets the currently selected date.

virtual bool wxCalendarCtrl::GetDateRange (wxDateTime * *lowerdate*, wxDateTime * *upperdate*) const `[virtual]`

Returns the limits currently being used.

See also

SetDateRange()

Parameters

lowerdate	If non-NULL, the value of the low limit for the dates shown by the control is returned (which may be wxDefaultDateTime if no limit is set).
upperdate	If non-NULL, the value of the upper limit for the dates shown by the control is returned (which may be wxDefaultDateTime if no limit is set).

Returns

true if either limit is set, false otherwise

virtual const wxColour& wxCalendarCtrl::GetHeaderColourBg ( ) const `[virtual]`

Gets the background colour of the header part of the calendar window.

This method is currently only implemented in generic wxCalendarCtrl and always returns `wxNullColour` in the native versions.

See also

SetHeaderColours()

virtual const wxColour& wxCalendarCtrl::GetHeaderColourFg ( ) const `[virtual]`

Gets the foreground colour of the header part of the calendar window.

This method is currently only implemented in generic wxCalendarCtrl and always returns `wxNullColour` in the native versions.

See also

SetHeaderColours()

virtual const wxColour& wxCalendarCtrl::GetHighlightColourBg ( ) const `[virtual]`

Gets the background highlight colour.

Only in generic wxCalendarCtrl.

This method is currently only implemented in generic wxCalendarCtrl and always returns `wxNullColour` in the native versions.

See also

SetHighlightColours()

virtual const wxColour& wxCalendarCtrl::GetHighlightColourFg ( ) const `[virtual]`

Gets the foreground highlight colour.

Only in generic wxCalendarCtrl.

This method is currently only implemented in generic wxCalendarCtrl and always returns `wxNullColour` in the native versions.

See also

SetHighlightColours()

virtual const wxColour& wxCalendarCtrl::GetHolidayColourBg ( ) const `[virtual]`

Return the background colour currently used for holiday highlighting.

Only useful with generic wxCalendarCtrl as native versions currently don't support holidays display at all and always return `wxNullColour`.

See also

SetHolidayColours()

virtual const wxColour& wxCalendarCtrl::GetHolidayColourFg ( ) const `[virtual]`

Return the foreground colour currently used for holiday highlighting.

Only useful with generic wxCalendarCtrl as native versions currently don't support holidays display at all and always return `wxNullColour`.

See also

SetHolidayColours()

virtual **wxCalendarHitTestResult** wxCalendarCtrl::HitTest (const **wxPoint** & *pos*, **wxDateTime** * *date* = NULL, **wxDateTime::WeekDay** * *wd* = NULL) [virtual]

Returns one of wxCalendarHitTestResult constants and fills either *date* or *wd* pointer with the corresponding value depending on the hit test code.

Not implemented in wxGTK currently.

virtual void wxCalendarCtrl::Mark (size_t *day*, bool *mark*) [virtual]

Mark or unmark the day.

This day of month will be marked in every month. In generic wxCalendarCtrl,

virtual void wxCalendarCtrl::ResetAttr (size_t *day*) [virtual]

Clears any attributes associated with the given day (in the range 1...31).

Only in generic wxCalendarCtrl.

virtual void wxCalendarCtrl::SetAttr (size_t *day*, **wxCalendarDateAttr** * *attr*) [virtual]

Associates the attribute with the specified date (in the range 1...31).

If the pointer is NULL, the items attribute is cleared. Only in generic wxCalendarCtrl.

virtual bool wxCalendarCtrl::SetDate (const **wxDateTime** & *date*) [virtual]

Sets the current date.

The *date* parameter must be valid and in the currently valid range as set by SetDateRange(), otherwise the current date is not changed and the function returns false.

virtual bool wxCalendarCtrl::SetDateRange (const **wxDateTime** & *lowerdate* = **wxDefaultDateTime**, const **wxDateTime** & *upperdate* = **wxDefaultDateTime**) [virtual]

Restrict the dates that can be selected in the control to the specified range.

If either date is set, the corresponding limit will be enforced and true returned. If none are set, the existing restrictions are removed and false is returned.

See also

GetDateRange()

Parameters

lowerdate	The low limit for the dates shown by the control or wxDefaultDateTime.
upperdate	The high limit for the dates shown by the control or wxDefaultDateTime.

Returns

true if either limit is valid, false otherwise

virtual void wxCalendarCtrl::SetHeaderColours (const **wxColour** & *colFg*, const **wxColour** & *colBg*) [virtual]

Set the colours used for painting the weekdays at the top of the control.

This method is currently only implemented in generic wxCalendarCtrl and does nothing in the native versions.

virtual void wxCalendarCtrl::SetHighlightColours (const wxColour & *colFg,* const wxColour & *colBg*) `[virtual]`

Set the colours to be used for highlighting the currently selected date.

This method is currently only implemented in generic wxCalendarCtrl and does nothing in the native versions.

virtual void wxCalendarCtrl::SetHoliday (size_t *day*) `[virtual]`

Marks the specified day as being a holiday in the current month.

This method is only implemented in the generic version of the control and does nothing in the native ones.

virtual void wxCalendarCtrl::SetHolidayColours (const wxColour & *colFg,* const wxColour & *colBg*) `[virtual]`

Sets the colours to be used for the holidays highlighting.

This method is only implemented in the generic version of the control and does nothing in the native ones. It should also only be called if the window style includes `wxCAL_SHOW_HOLIDAYS` flag or EnableHolidayDisplay() had been called.

21.78 wxCalendarDateAttr Class Reference

`#include <wx/calctrl.h>`

21.78.1 Detailed Description

wxCalendarDateAttr is a custom attributes for a calendar date.

The objects of this class are used with wxCalendarCtrl.

Library: wxAdvanced

Category: Data Structures

See also

wxCalendarCtrl

Public Member Functions

- wxCalendarDateAttr (const wxColour &colText=wxNullColour, const wxColour &colBack=wxNullColour, const wxColour &colBorder=wxNullColour, const wxFont &font=wxNullFont, wxCalendarDateBorder border=wxCAL_BORDER_NONE)

  Constructor for specifying all wxCalendarDateAttr properties.
- wxCalendarDateAttr (wxCalendarDateBorder border, const wxColour &colBorder=wxNullColour)

  Constructor using default properties except the given border.
- const wxColour & GetBackgroundColour () const

  Returns the background colour set for the calendar date.
- wxCalendarDateBorder GetBorder () const

  Returns the border set for the calendar date.
- const wxColour & GetBorderColour () const

  Returns the border colour set for the calendar date.

- const wxFont & GetFont () const

  Returns the font set for the calendar date.
- const wxColour & GetTextColour () const

  Returns the text colour set for the calendar date.
- bool HasBackgroundColour () const

  Returns true if a non-default text background colour is set.
- bool HasBorder () const

  Returns true if a non-default (i.e. any) border is set.
- bool HasBorderColour () const

  Returns true if a non-default border colour is set.
- bool HasFont () const

  Returns true if a non-default font is set.
- bool HasTextColour () const

  Returns true if a non-default text foreground colour is set.
- bool IsHoliday () const

  Returns true if this calendar day is displayed as a holiday.
- void SetBackgroundColour (const wxColour &colBack)

  Sets the text background colour to use.
- void SetBorder (wxCalendarDateBorder border)

  Sets the border to use.
- void SetBorderColour (const wxColour &col)

  Sets the border colour to use.
- void SetFont (const wxFont &font)

  Sets the font to use.
- void SetHoliday (bool holiday)

  If holiday is true, this calendar day will be displayed as a holiday.
- void SetTextColour (const wxColour &colText)

  Sets the text (foreground) colour to use.

Static Public Member Functions

- static const wxCalendarDateAttr & GetMark ()

  Used (internally) by the generic wxCalendarCtrl::Mark().
- static void SetMark (const wxCalendarDateAttr &m)

  Set the attributes that will be used to Mark() days on the generic wxCalendarCtrl.

21.78.2 Constructor & Destructor Documentation

wxCalendarDateAttr::wxCalendarDateAttr (const wxColour & *colText* = wxNullColour, const wxColour & *colBack* = wxNullColour, const wxColour & *colBorder* = wxNullColour, const wxFont & *font* = wxNullFont, wxCalendarDateBorder *border* = wxCAL_BORDER_NONE)

Constructor for specifying all wxCalendarDateAttr properties.

wxCalendarDateAttr::wxCalendarDateAttr (wxCalendarDateBorder *border*, const wxColour & *colBorder* = wxNullColour)

Constructor using default properties except the given border.

21.78.3 Member Function Documentation

const wxColour& wxCalendarDateAttr::GetBackgroundColour ( ) const

Returns the background colour set for the calendar date.

wxCalendarDateBorder wxCalendarDateAttr::GetBorder ( ) const

Returns the border set for the calendar date.

const wxColour& wxCalendarDateAttr::GetBorderColour ( ) const

Returns the border colour set for the calendar date.

const wxFont& wxCalendarDateAttr::GetFont ( ) const

Returns the font set for the calendar date.

static const wxCalendarDateAttr& wxCalendarDateAttr::GetMark ( ) [static]

Used (internally) by the generic wxCalendarCtrl::Mark().

const wxColour& wxCalendarDateAttr::GetTextColour ( ) const

Returns the text colour set for the calendar date.

bool wxCalendarDateAttr::HasBackgroundColour ( ) const

Returns true if a non-default text background colour is set.

bool wxCalendarDateAttr::HasBorder ( ) const

Returns true if a non-default (i.e. any) border is set.

bool wxCalendarDateAttr::HasBorderColour ( ) const

Returns true if a non-default border colour is set.

bool wxCalendarDateAttr::HasFont ( ) const

Returns true if a non-default font is set.

bool wxCalendarDateAttr::HasTextColour ( ) const

Returns true if a non-default text foreground colour is set.

bool wxCalendarDateAttr::IsHoliday ( ) const

Returns true if this calendar day is displayed as a holiday.

void wxCalendarDateAttr::SetBackgroundColour (const wxColour & *colBack*)

Sets the text background colour to use.

void wxCalendarDateAttr::SetBorder (wxCalendarDateBorder *border*)

Sets the border to use.

void wxCalendarDateAttr::SetBorderColour (const wxColour & *col*)

Sets the border colour to use.

void wxCalendarDateAttr::SetFont (const wxFont & *font*)

Sets the font to use.

void wxCalendarDateAttr::SetHoliday (bool *holiday*)

If *holiday* is true, this calendar day will be displayed as a holiday.

static void wxCalendarDateAttr::SetMark (const wxCalendarDateAttr & *m*) `[static]`

Set the attributes that will be used to Mark() days on the generic wxCalendarCtrl.

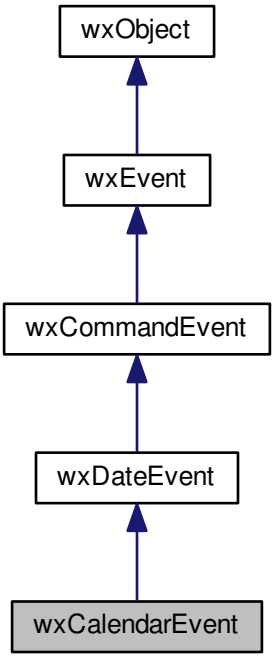
void wxCalendarDateAttr::SetTextColour (const wxColour & *colText*)

Sets the text (foreground) colour to use.

21.79 wxCalendarEvent Class Reference

`#include <wx/calctrl.h>`

Inheritance diagram for wxCalendarEvent:



21.79.1 Detailed Description

The wxCalendarEvent class is used together with wxCalendarCtrl.

Library: wxAdvanced

Category: Events

See also

wxCalendarCtrl

Public Member Functions

- wxCalendarEvent ()
- wxCalendarEvent (wxWindow *win, const wxDateTime &dt, wxEventType type)
- wxDateTime::WeekDay GetWeekDay () const

  Returns the week day on which the user clicked in `EVT_CALENDAR_WEEKDAY_CLICKED` *handler.*
- void SetWeekDay (const wxDateTime::WeekDay day)

  Sets the week day carried by the event, normally only used by the library internally.

Additional Inherited Members

21.79.2 Constructor & Destructor Documentation

wxCalendarEvent::wxCalendarEvent ( )

wxCalendarEvent::wxCalendarEvent (wxWindow * *win,* const wxDateTime & *dt,* wxEventType *type*)

21.79.3 Member Function Documentation

wxDateTime::WeekDay wxCalendarEvent::GetWeekDay ( ) const

Returns the week day on which the user clicked in `EVT_CALENDAR_WEEKDAY_CLICKED` handler.

It doesn't make sense to call this function in other handlers.

void wxCalendarEvent::SetWeekDay (const wxDateTime::WeekDay *day*)

Sets the week day carried by the event, normally only used by the library internally.

21.80 wxCaret Class Reference

```
#include <wx/caret.h>
```

21.80.1 Detailed Description

A caret is a blinking cursor showing the position where the typed text will appear.

Text controls usually have their own caret but wxCaret provides a way to use a caret in other windows.

Currently, the caret appears as a rectangle of the given size. In the future, it will be possible to specify a bitmap to be used for the caret shape.

A caret is always associated with a window and the current caret can be retrieved using wxWindow::GetCaret(). The same caret can't be reused in two different windows.

Library: wxCore

Category: Miscellaneous

Public Member Functions

- wxCaret ()

  Default constructor.
- wxWindow * GetWindow () const

  Get the window the caret is associated with.
- virtual void Hide ()

  Hides the caret, same as Show(false).
- bool IsOk () const

  Returns true if the caret was created successfully.
- bool IsVisible () const

Returns true if the caret is visible and false if it is permanently hidden (if it is blinking and not shown currently but will be after the next blink, this method still returns true).

- virtual void Show (bool show=true)

  Shows or hides the caret.

- wxCaret (wxWindow *window, int width, int height)

  Creates a caret with the given size (in pixels) and associates it with the window.
- wxCaret (wxWindow *window, const wxSize &size)

  Creates a caret with the given size (in pixels) and associates it with the window.

- bool Create (wxWindow *window, int width, int height)

  Creates a caret with the given size (in pixels) and associates it with the window (same as the equivalent constructors).
- bool Create (wxWindow *window, const wxSize &size)

  Creates a caret with the given size (in pixels) and associates it with the window (same as the equivalent constructors).

- void GetPosition (int *x, int *y) const

  Get the caret position (in pixels).
- wxPoint GetPosition () const

  Get the caret position (in pixels).

- void GetSize (int *width, int *height) const

  Get the caret size.
- wxSize GetSize () const

  Get the caret size.

- void Move (int x, int y)

  Move the caret to given position (in logical coordinates).
- void Move (const wxPoint &pt)

  Move the caret to given position (in logical coordinates).

- void SetSize (int width, int height)

  Changes the size of the caret.
- void SetSize (const wxSize &size)

  Changes the size of the caret.

Static Public Member Functions

- static int GetBlinkTime ()

  Returns the blink time which is measured in milliseconds and is the time elapsed between 2 inversions of the caret (blink time of the caret is the same for all carets, so this functions is static).
- static void SetBlinkTime (int milliseconds)

  Sets the blink time for all the carets.

21.80.2 Constructor & Destructor Documentation

wxCaret::wxCaret ( )

Default constructor.

wxCaret::wxCaret (wxWindow * *window,* int *width,* int *height*)

Creates a caret with the given size (in pixels) and associates it with the *window*.

wxCaret::wxCaret (wxWindow * *window,* const wxSize & *size*)

Creates a caret with the given size (in pixels) and associates it with the *window*.

21.80.3 Member Function Documentation

bool wxCaret::Create (wxWindow * *window,* int *width,* int *height*)

Creates a caret with the given size (in pixels) and associates it with the *window* (same as the equivalent constructors).

bool wxCaret::Create (wxWindow * *window,* const wxSize & *size*)

Creates a caret with the given size (in pixels) and associates it with the *window* (same as the equivalent constructors).

static int wxCaret::GetBlinkTime ( ) [static]

Returns the blink time which is measured in milliseconds and is the time elapsed between 2 inversions of the caret (blink time of the caret is the same for all carets, so this functions is static).

void wxCaret::GetPosition (int * *x,* int * *y*) const

Get the caret position (in pixels).

wxPerl Note: In wxPerl there are two methods instead of a single overloaded method:

- GetPosition(): returns a Wx::Point object.
- GetPositionXY(): returns a 2-element list (x, y).

wxPoint wxCaret::GetPosition ( ) const

Get the caret position (in pixels).

wxPerl Note: In wxPerl there are two methods instead of a single overloaded method:

- GetPosition(): returns a Wx::Point object.
- GetPositionXY(): returns a 2-element list (x, y).

void wxCaret::GetSize (int * *width,* int * *height*) const

Get the caret size.

wxPerl Note: In wxPerl there are two methods instead of a single overloaded method:

- GetSize(): returns a Wx::Size object.
- GetSizeWH(): returns a 2-element list (width, height).

wxSize wxCaret::GetSize ( ) const

Get the caret size.

wxPerl Note: In wxPerl there are two methods instead of a single overloaded method:

- GetSize(): returns a Wx::Size object.
- GetSizeWH(): returns a 2-element list (width, height).

wxWindow* wxCaret::GetWindow ( ) const

Get the window the caret is associated with.

virtual void wxCaret::Hide ( ) [virtual]

Hides the caret, same as Show(false).

bool wxCaret::IsOk ( ) const

Returns true if the caret was created successfully.

bool wxCaret::IsVisible ( ) const

Returns true if the caret is visible and false if it is permanently hidden (if it is blinking and not shown currently but will be after the next blink, this method still returns true).

void wxCaret::Move (int *x*, int *y*)

Move the caret to given position (in logical coordinates).

void wxCaret::Move (const wxPoint & *pt*)

Move the caret to given position (in logical coordinates).

static void wxCaret::SetBlinkTime (int *milliseconds*) [static]

Sets the blink time for all the carets.

Warning

Under Windows, this function will change the blink time for all carets permanently (until the next time it is called), even for carets in other applications.

See also

GetBlinkTime()

void wxCaret::SetSize (int *width*, int *height*)

Changes the size of the caret.

void wxCaret::SetSize (const wxSize & *size*)

Changes the size of the caret.

virtual void wxCaret::Show (bool *show* = `true`) [virtual]

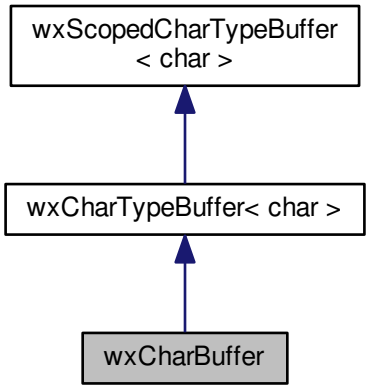
Shows or hides the caret.

Notice that if the caret was hidden N times, it must be shown N times as well to reappear on the screen.

21.81 wxCharBuffer Class Reference

`#include <wx/buffer.h>`

Inheritance diagram for wxCharBuffer:



21.81.1 Detailed Description

This is a specialization of wxCharTypeBuffer<T> for `char` type.

Library: None; this class implementation is entirely header-based.

Category: Data Structures

Public Types

- typedef wxCharTypeBuffer< char > wxCharTypeBufferBase
- typedef wxScopedCharTypeBuffer < char > wxScopedCharTypeBufferBase

Public Member Functions

- wxCharBuffer (const wxCharTypeBufferBase &buf)
- wxCharBuffer (const wxScopedCharTypeBufferBase &buf)
- wxCharBuffer (const CharType *str=NULL)
- wxCharBuffer (size_t len)
- wxCharBuffer (const wxCStrData &cstr)

Additional Inherited Members

21.81.2 Member Typedef Documentation

typedef wxCharTypeBuffer<char> wxCharBuffer::wxCharTypeBufferBase

typedef wxScopedCharTypeBuffer<char> wxCharBuffer::wxScopedCharTypeBufferBase

21.81.3 Constructor & Destructor Documentation

wxCharBuffer::wxCharBuffer (const wxCharTypeBufferBase & *buf*)

wxCharBuffer::wxCharBuffer (const wxScopedCharTypeBufferBase & *buf*)

wxCharBuffer::wxCharBuffer (const CharType * *str* = NULL)

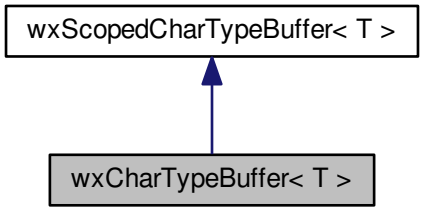
wxCharBuffer::wxCharBuffer (size_t *len*)

wxCharBuffer::wxCharBuffer (const wxCStrData & *cstr*)

21.82 wxCharTypeBuffer< T > Class Template Reference

`#include <wx/buffer.h>`

Inheritance diagram for wxCharTypeBuffer< T >:



21.82.1 Detailed Description

template<typename T>class wxCharTypeBuffer< T >

wxCharTypeBuffer<T> is a template class for storing characters.

The difference from wxScopedCharTypeBuffer<T> is that this class doesn't have non-owned mode and the data stored in it are valid for as long as the buffer instance exists. Other than that, this class' behaviour is the same as wxScopedCharTypeBuffer<T>'s – in particular, the data are reference-counted and copying the buffer is cheap.

wxScopedCharTypeBuffer<T> buffers can be converted into wxCharTypeBuffer<T>.

Template Parameters

T	The type of the characters stored in this class.

Since

2.9.0

Library: None; this class implementation is entirely header-based.

Category: Data Structures

Public Member Functions

- wxCharTypeBuffer (const CharType *str=NULL, size_t len=wxNO_LEN)

  Creates (owned) buffer from str and takes ownership of it.
- wxCharTypeBuffer (size_t len)

  Creates (owned) buffer of size len.
- wxCharTypeBuffer (const wxCharTypeBuffer &src)

  Copy constructor.
- wxCharTypeBuffer (const wxScopedCharTypeBuffer< T > &src)

  Makes a copy of scoped buffer src.
- wxCharTypeBuffer & operator= (const CharType *str)

  Assigns str to this buffer and takes ownership of it (i.e. the buffer becomes "owned").
- wxCharTypeBuffer & operator= (const wxCharTypeBuffer &src)

  Assignment operator behaves in the same way as the copy constructor.
- wxCharTypeBuffer & operator= (const wxScopedCharTypeBuffer< T > &src)

  Assigns a scoped buffer to this buffer.
- bool extend (size_t len)

  Extends the buffer to have size len.
- bool shrink (size_t len)

  Shrinks the buffer to have size len and NUL-terminates the string at this length.

Additional Inherited Members

21.82.2 Constructor & Destructor Documentation

template<typename T> wxCharTypeBuffer< T >::wxCharTypeBuffer (const CharType * *str* = NULL, size_t *len* = wxNO_LEN)

Creates (owned) buffer from *str* and takes ownership of it.

Parameters

str	String data.
len	If specified, length of the string, otherwise the string is considered to be NUL-terminated.

See also

wxScopedCharTypeBuffer<T>::CreateOwned()

template<typename T> wxCharTypeBuffer< T >::wxCharTypeBuffer (size_t *len*)

Creates (owned) buffer of size *len*.

See also

wxScopedCharTypeBuffer<T>::CreateOwned()

template<typename T> wxCharTypeBuffer< T >::wxCharTypeBuffer (const wxCharTypeBuffer< T > & *src*)

Copy constructor.

Increases reference count on the data, does *not* make wxStrdup() copy of the data.

template<typename T> wxCharTypeBuffer< T >::wxCharTypeBuffer (const wxScopedCharTypeBuffer< T > & *src*)

Makes a copy of scoped buffer *src*.

If *src* is a non-owned buffer, a copy of its data is made using wxStrdup(). If *src* is an owned buffer, this constructor behaves in the usual way (reference count on buffer data is incremented).

21.82.3 Member Function Documentation

template<typename T> bool wxCharTypeBuffer< T >::extend (size_t *len*)

Extends the buffer to have size *len*.

Can only be called on buffers that don't share data with another buffer (i.e. reference count of the data is 1).

See also

shrink()

template<typename T> wxCharTypeBuffer& wxCharTypeBuffer< T >::operator= (const CharType * *str*)

Assigns *str* to this buffer and takes ownership of it (i.e. the buffer becomes "owned").

template<typename T> wxCharTypeBuffer& wxCharTypeBuffer< T >::operator= (const wxCharTypeBuffer< T > & *src*)

Assignment operator behaves in the same way as the copy constructor.

template<typename T> wxCharTypeBuffer& wxCharTypeBuffer< T >::operator= (const wxScopedCharTypeBuffer< T > & *src*)

Assigns a scoped buffer to this buffer.

If *src* is a non-owned buffer, a copy of its data is made using wxStrdup(). If *src* is an owned buffer, the assignment behaves in the usual way (reference count on buffer data is incremented).

template<typename T> bool wxCharTypeBuffer< T >::shrink (size_t *len*)

Shrinks the buffer to have size *len* and NUL-terminates the string at this length.

Can only be called on buffers that don't share data with another buffer (i.e. reference count of the data is 1).

Parameters

len	Length to shrink to. Must not be larger than current length.

Note

The string is not reallocated to take less memory.

Since

2.9.0

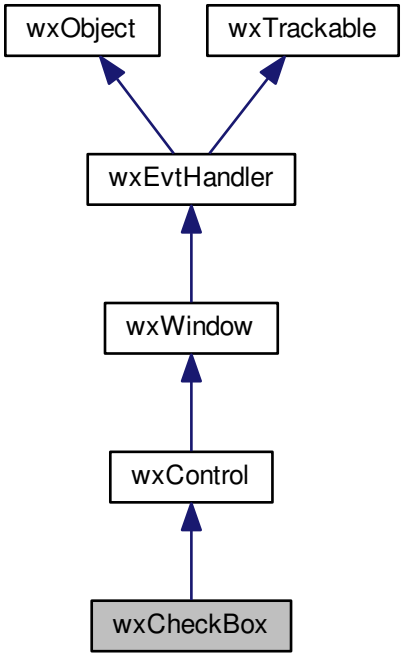
See also

extend()

21.83 wxCheckBox Class Reference

```
#include <wx/checkbox.h>
```

Inheritance diagram for wxCheckBox:



21.83.1 Detailed Description

A checkbox is a labelled box which by default is either on (checkmark is visible) or off (no checkmark).

Optionally (when the wxCHK_3STATE style flag is set) it can have a third state, called the mixed or undetermined state. Often this is used as a "Does Not Apply" state.

Styles

This class supports the following styles:

- wxCHK_2STATE: Create a 2-state checkbox. This is the default.
- wxCHK_3STATE: Create a 3-state checkbox. Not implemented in wxGTK1.
- wxCHK_ALLOW_3RD_STATE_FOR_USER: By default a user can't set a 3-state checkbox to the third state. It can only be done from code. Using this flags allows the user to set the checkbox to the third state by clicking.
- wxALIGN_RIGHT: Makes the text appear on the left of the checkbox.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_CHECKBOX(id, func): Process a `wxEVT_CHECKBOX` event, when the checkbox is clicked.

Library: wxCore

Category: Controls

See also

wxRadioButton, wxCommandEvent

Public Member Functions

- wxCheckBox ()

  Default constructor.
- wxCheckBox (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxCheckBoxNameStr)

  Constructor, creating and showing a checkbox.
- virtual ~wxCheckBox ()

  Destructor, destroying the checkbox.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxCheckBoxNameStr)

  Creates the checkbox for two-step construction.
- virtual bool GetValue () const

  Gets the state of a 2-state checkbox.
- wxCheckBoxState Get3StateValue () const

  Gets the state of a 3-state checkbox.
- bool Is3State () const

  Returns whether or not the checkbox is a 3-state checkbox.
- bool Is3rdStateAllowedForUser () const

  Returns whether or not the user can set the checkbox to the third state.
- bool IsChecked () const

  This is just a maybe more readable synonym for GetValue(): just as the latter, it returns true if the checkbox is checked and false otherwise.
- virtual void SetValue (bool state)

  Sets the checkbox to the given state.
- void Set3StateValue (wxCheckBoxState state)

  Sets the checkbox to the given state.

Additional Inherited Members

21.83.2 Constructor & Destructor Documentation

wxCheckBox::wxCheckBox ( )

Default constructor.

See also

Create(), wxValidator

wxCheckBox::wxCheckBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxCheckBoxNameStr`)

Constructor, creating and showing a checkbox.

Parameters

parent	Parent window. Must not be NULL.
id	Checkbox identifier. The value wxID_ANY indicates a default value.
label	Text to be displayed next to the checkbox.
pos	Checkbox position. If wxDefaultPosition is specified then a default position is chosen.
size	Checkbox size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxCheckBox.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxCheckBox::~wxCheckBox ( ) [virtual]

Destructor, destroying the checkbox.

21.83.3 Member Function Documentation

bool wxCheckBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxCheckBoxNameStr)

Creates the checkbox for two-step construction.

See wxCheckBox() for details.

wxCheckBoxState wxCheckBox::Get3StateValue ( ) const

Gets the state of a 3-state checkbox.

Asserts when the function is used with a 2-state checkbox.

virtual bool wxCheckBox::GetValue ( ) const [virtual]

Gets the state of a 2-state checkbox.

Returns

Returns true if it is checked, false otherwise.

bool wxCheckBox::Is3rdStateAllowedForUser ( ) const

Returns whether or not the user can set the checkbox to the third state.

Returns

true if the user can set the third state of this checkbox, false if it can only be set programmatically or if it's a 2-state checkbox.

bool wxCheckBox::Is3State ( ) const

Returns whether or not the checkbox is a 3-state checkbox.

Returns

true if this checkbox is a 3-state checkbox, false if it's a 2-state checkbox.

bool wxCheckBox::IsChecked ( ) const

This is just a maybe more readable synonym for GetValue(): just as the latter, it returns true if the checkbox is checked and false otherwise.

void wxCheckBox::Set3StateValue (wxCheckBoxState *state*)

Sets the checkbox to the given state.

This does not cause a `wxEVT_CHECKBOX` event to get emitted.

Asserts when the checkbox is a 2-state checkbox and setting the state to wxCHK_UNDETERMINED.

virtual void wxCheckBox::SetValue (bool *state*) [virtual]

Sets the checkbox to the given state.

This does not cause a `wxEVT_CHECKBOX` event to get emitted.

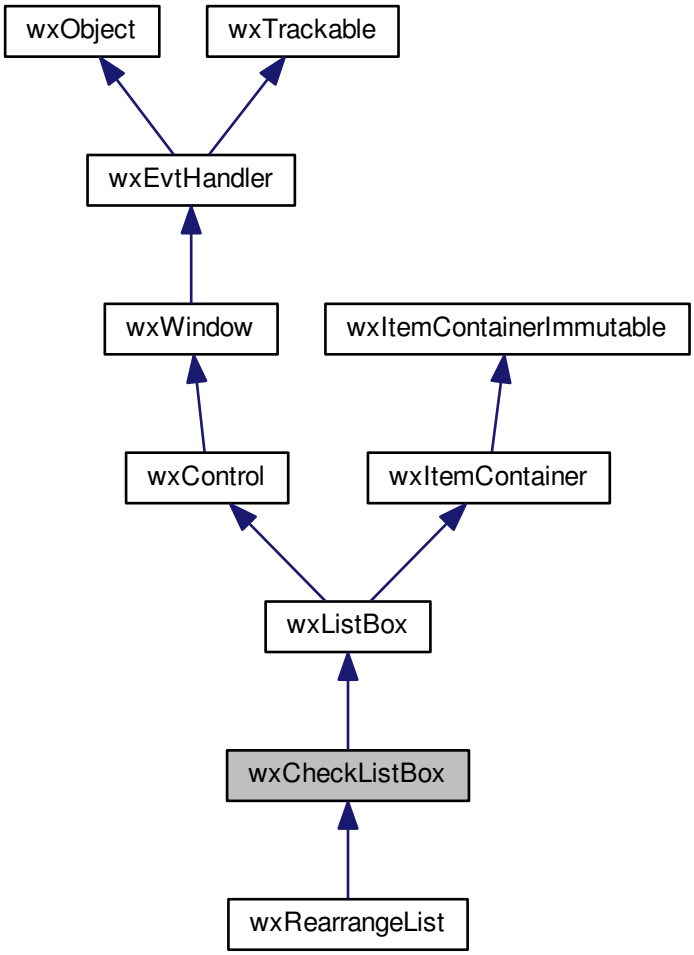
Parameters

state	If true, the check is on, otherwise it is off.

21.84 wxCheckListBox Class Reference

`#include <wx/checklst.h>`

Inheritance diagram for wxCheckListBox:



21.84.1 Detailed Description

A wxCheckListBox is like a wxListBox, but allows items to be checked or unchecked.

When using this class under Windows wxWidgets must be compiled with wxUSE_OWNER_DRAWN set to 1.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_CHECKLISTBOX(id, func): Process a `wxEVT_CHECKLISTBOX` event, when an item in the check list box is checked or unchecked. wxCommandEvent::GetInt() will contain the index of the item that was checked or unchecked. wxCommandEvent::IsChecked() is not valid! Use wxCheckListBox::IsChecked() instead.

Library: wxCore

Category: Controls

See also

wxListBox, wxChoice, wxComboBox, wxListCtrl, wxCommandEvent

Public Member Functions

- wxCheckListBox ()

  Default constructor.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int nStrings=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListBoxNameStr)
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListBoxNameStr)
- virtual ~wxCheckListBox ()

  Destructor, destroying the list box.
- void Check (unsigned int item, bool check=true)

  Checks the given item.
- bool IsChecked (unsigned int item) const

  Returns true if the given item is checked, false otherwise.
- unsigned int GetCheckedItems (wxArrayInt &checkedItems) const

  Return the indices of the checked items.

- wxCheckListBox (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name="listBox")

  Constructor, creating and showing a list box.
- wxCheckListBox (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name="listBox")

  Constructor, creating and showing a list box.

Additional Inherited Members

21.84.2 Constructor & Destructor Documentation

wxCheckListBox::wxCheckListBox ( )

Default constructor.

wxCheckListBox::wxCheckListBox (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "listBox")

Constructor, creating and showing a list box.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
n	Number of strings with which to initialise the control.
choices	An array of strings with which to initialise the control.
style	Window style. See wxCheckListBox.
validator	Window validator.
name	Window name.

wxPerl Note: Not supported by wxPerl.

wxCheckListBox::wxCheckListBox (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "listBox")

Constructor, creating and showing a list box.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
choices	An array of strings with which to initialise the control.
style	Window style. See wxCheckListBox.
validator	Window validator.
name	Window name.

wxPerl Note: Use an array reference for the *choices* parameter.

virtual wxCheckListBox::~wxCheckListBox ( ) [virtual]

Destructor, destroying the list box.

21.84.3 Member Function Documentation

void wxCheckListBox::Check (unsigned int *item*, bool *check* = true)

Checks the given item.

Note that calling this method does not result in a wxEVT_CHECKLISTBOX event being emitted.

Parameters

item	Index of item to check.
check	true if the item is to be checked, false otherwise.

bool wxCheckListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *nStrings* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxListBoxNameStr)

bool wxCheckListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxListBoxNameStr)

unsigned int wxCheckListBox::GetCheckedItems (wxArrayInt & *checkedItems*) const

Return the indices of the checked items.

Parameters

checkedItems	A reference to the array that is filled with the indices of the checked items.

Returns

The number of checked items.

See also

Check(), IsChecked()

Since

2.9.5

bool wxCheckListBox::IsChecked (unsigned int *item*) const

Returns true if the given item is checked, false otherwise.

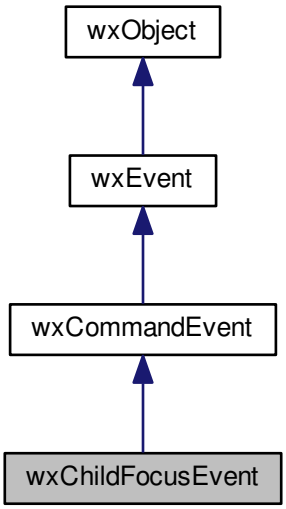
Parameters

item	Index of item whose check status is to be returned.

21.85 wxChildFocusEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxChildFocusEvent:



21.85.1 Detailed Description

A child focus event is sent to a (parent-)window when one of its child windows gains focus, so that the window could restore the focus back to its corresponding child if it loses it now and regains later.

Notice that child window is the direct child of the window receiving event. Use wxWindow::FindFocus() to retrieve the window which is actually getting focus.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxChildFocusEvent& event)

Event macros:

- EVT_CHILD_FOCUS(func): Process a `wxEVT_CHILD_FOCUS` event.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxChildFocusEvent (wxWindow *win=NULL)

  Constructor.
- wxWindow * GetWindow () const

  Returns the direct child which receives the focus, or a (grand-)parent of the control receiving the focus.

Additional Inherited Members

21.85.2 Constructor & Destructor Documentation

wxChildFocusEvent::wxChildFocusEvent (wxWindow * *win* = `NULL`)

Constructor.

Parameters

win	The direct child which is (or which contains the window which is) receiving the focus.

21.85.3 Member Function Documentation

wxWindow* wxChildFocusEvent::GetWindow ( ) const

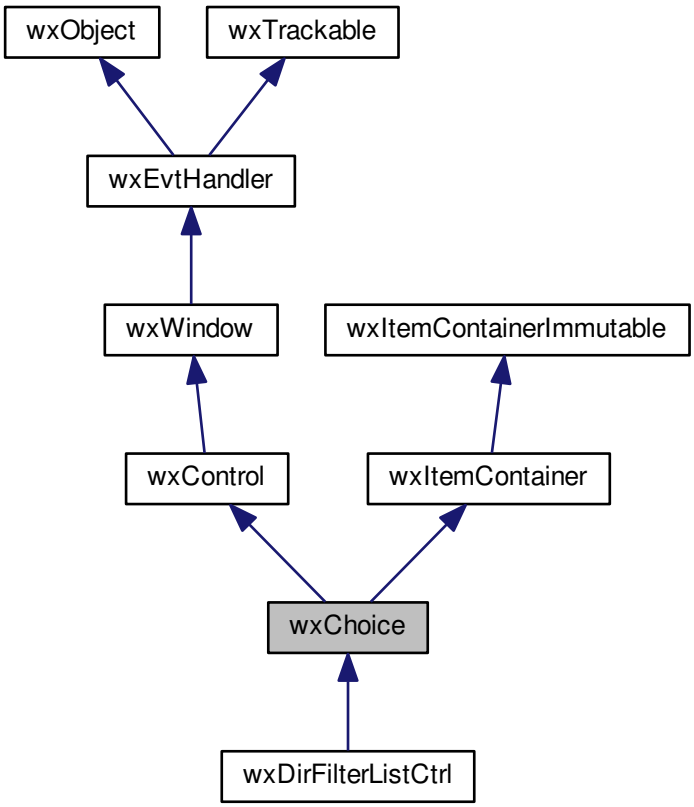
Returns the direct child which receives the focus, or a (grand-)parent of the control receiving the focus.

To get the actually focused control use wxWindow::FindFocus.

21.86 wxChoice Class Reference

`#include <wx/choice.h>`

Inheritance diagram for wxChoice:



21.86.1 Detailed Description

A choice item is used to select one of a list of strings.

Unlike a wxListBox, only the selection is visible until the user pulls down the menu of choices.

Styles

This class supports the following styles:

- wxCB_SORT: Sorts the entries alphabetically.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_CHOICE(id, func): Process a `wxEVT_CHOICE` event, when an item on the list is selected.

Library: wxCore

Category: Controls

See also

wxListBox, wxComboBox, wxCommandEvent

Public Member Functions

- wxChoice ()

  Default constructor.
- wxChoice (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxChoiceNameStr)

  Constructor, creating and showing a choice.
- wxChoice (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxChoiceNameStr)

  Constructor, creating and showing a choice.
- virtual ~wxChoice ()

  Destructor, destroying the choice item.
- virtual int GetColumns () const

  Gets the number of columns in this choice item.
- virtual int GetCurrentSelection () const

  Unlike wxControlWithItems::GetSelection() which only returns the accepted selection value (the selection in the control once the user closes the dropdown list), this function returns the current selection.
- virtual void SetColumns (int n=1)

  Sets the number of columns in this choice item.
- virtual bool IsSorted () const
- virtual unsigned int GetCount () const

  Returns the number of items in the control.
- virtual int GetSelection () const

  Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.
- virtual void SetSelection (int n)

  Sets the selection to the given item n or removes the selection entirely if n == `wxNOT_FOUND`.
- virtual int FindString (const wxString &s, bool bCase=false) const

  Finds an item whose label matches the given string.
- virtual wxString GetString (unsigned int n) const

  Returns the label of the item with the given index.
- virtual void SetString (unsigned int pos, const wxString &s)

  Sets the label for the given item.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxChoiceNameStr)

  Creates the choice for two-step construction.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxChoiceNameStr)

  Creates the choice for two-step construction.

Additional Inherited Members

21.86.2 Constructor & Destructor Documentation

wxChoice::wxChoice ( )

Default constructor.

See also

Create(), wxValidator

wxChoice::wxChoice (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxChoiceNameStr)

Constructor, creating and showing a choice.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the choice is sized appropriately.
n	Number of strings with which to initialise the choice control.
choices	An array of strings with which to initialise the choice control.
style	Window style. See wxChoice.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

wxPerl Note: Not supported by wxPerl.

wxChoice::wxChoice (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxChoiceNameStr)

Constructor, creating and showing a choice.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position.

size	Window size. If wxDefaultSize is specified then the choice is sized appropriately.
choices	An array of strings with which to initialise the choice control.
style	Window style. See wxChoice.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

wxPerl Note: Use an array reference for the *choices* parameter.

virtual wxChoice::~wxChoice ( ) [virtual]

Destructor, destroying the choice item.

21.86.3 Member Function Documentation

bool wxChoice::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxChoiceNameStr)

Creates the choice for two-step construction.

See wxChoice().

bool wxChoice::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxChoiceNameStr)

Creates the choice for two-step construction.

See wxChoice().

virtual int wxChoice::FindString (const wxString & *string*, bool *caseSensitive* = false) const [virtual]

Finds an item whose label matches the given string.

Parameters

string	String to find.
caseSensitive	Whether search is case sensitive (default is not).

Returns

The zero-based position of the item, or wxNOT_FOUND if the string was not found.

Reimplemented from wxItemContainerImmutable.

virtual int wxChoice::GetColumns ( ) const [virtual]

Gets the number of columns in this choice item.

Remarks

This is implemented for GTK and Motif only and always returns 1 for the other platforms.

virtual unsigned int wxChoice::GetCount ( ) const `[virtual]`

Returns the number of items in the control.

See also

IsEmpty()

Implements wxItemContainerImmutable.

virtual int wxChoice::GetCurrentSelection ( ) const `[virtual]`

Unlike wxControlWithItems::GetSelection() which only returns the accepted selection value (the selection in the control once the user closes the dropdown list), this function returns the current selection.

That is, while the dropdown list is shown, it returns the currently selected item in it. When it is not shown, its result is the same as for the other function.

Since

2.6.2. In older versions, wxControlWithItems::GetSelection() itself behaved like this.

virtual int wxChoice::GetSelection ( ) const `[virtual]`

Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

Returns

The position of the current selection.

Remarks

This method can be used with single selection list boxes only, you should use wxListBox::GetSelections() for the list boxes with wxLB_MULTIPLE style.

See also

SetSelection(), GetStringSelection()

Implements wxItemContainerImmutable.

virtual wxString wxChoice::GetString (unsigned int *n*) const `[virtual]`

Returns the label of the item with the given index.

Parameters

n	The zero-based index.

Returns

The label of the item or an empty string if the position was invalid.

Implements wxItemContainerImmutable.

virtual bool wxChoice::IsSorted ( ) const `[virtual]`

virtual void wxChoice::SetColumns (int *n* = `1`) `[virtual]`

Sets the number of columns in this choice item.

Parameters

n	Number of columns.

Remarks

This is implemented for GTK and Motif only and doesn't do anything under other platforms.

virtual void wxChoice::SetSelection (int *n*) [virtual]

Sets the selection to the given item *n* or removes the selection entirely if *n* == wxNOT_FOUND.

Note that this does not cause any command events to be emitted nor does it deselect any other items in the controls which support multiple selections.

Parameters

n	The string position to select, starting from zero.

See also

SetString(), SetStringSelection()

Implements wxItemContainerImmutable.

virtual void wxChoice::SetString (unsigned int *n*, const wxString & *string*) [virtual]

Sets the label for the given item.

Parameters

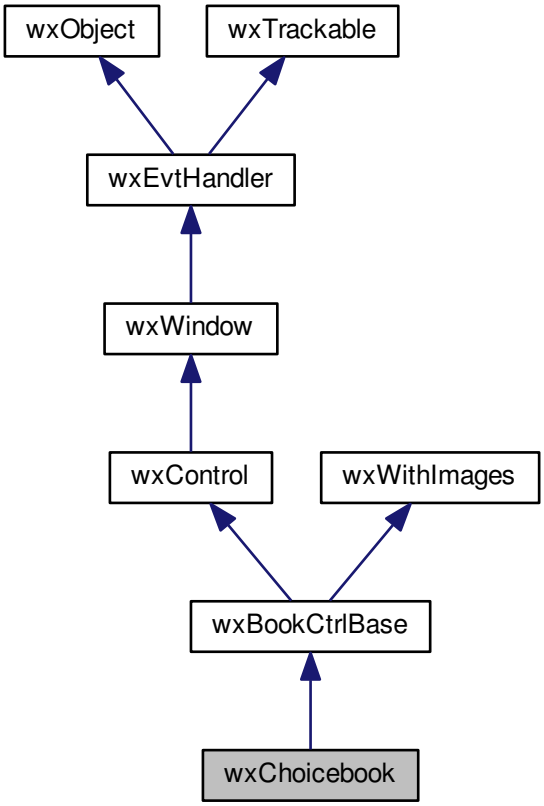
n	The zero-based item index.
string	The label to set.

Implements wxItemContainerImmutable.

21.87 wxChoicebook Class Reference

```
#include <wx/choicebk.h>
```

Inheritance diagram for wxChoicebook:



21.87.1 Detailed Description

wxChoicebook is a class similar to wxNotebook, but uses a wxChoice control to show the labels instead of the tabs.

For usage documentation of this class, please refer to the base abstract class wxBookCtrl. You can also use the Notebook Sample to see wxChoicebook in action.

wxChoicebook allows the use of wxBookCtrlBase::GetControlSizer(), allowing a program to add other controls next to the choice control. This is particularly useful when screen space is restricted, as it often is when wxChoicebook is being employed.

Styles

This class supports the following styles:

- wxCHB_DEFAULT: Choose the default location for the labels depending on the current platform (left everywhere except Mac where it is top).
- wxCHB_TOP: Place labels above the page area.
- wxCHB_LEFT: Place labels on the left side.
- wxCHB_RIGHT: Place labels on the right side.

- wxCHB_BOTTOM: Place labels below the page area.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxBookCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_CHOICEBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_CHOICEBOOK_PAGE_CHANGED` event.
- EVT_CHOICEBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_CHOICEBOOK_PAGE_CHANGING` event. This event can be vetoed (using wxNotifyEvent::Veto()).

Library: wxCore

Category: Book Controls

See also

wxBookCtrl Overview, wxNotebook, Notebook Sample

Public Member Functions

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Create the choicebook control that has already been constructed with the default constructor.
- wxChoice * GetChoiceCtrl () const

  Returns the wxChoice associated with the control.

- wxChoicebook ()

  Constructs a choicebook control.
- wxChoicebook (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Constructs a choicebook control.

Additional Inherited Members

21.87.2 Constructor & Destructor Documentation

wxChoicebook::wxChoicebook ( )

Constructs a choicebook control.

wxChoicebook::wxChoicebook (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Constructs a choicebook control.

21.87.3 Member Function Documentation

bool wxChoicebook::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Create the choicebook control that has already been constructed with the default constructor.

wxChoice* wxChoicebook::GetChoiceCtrl ( ) const

Returns the wxChoice associated with the control.

21.88 wxClassInfo Class Reference

`#include <wx/object.h>`

21.88.1 Detailed Description

This class stores meta-information about classes.

Instances of this class are not generally defined directly by an application, but indirectly through use of macros such as wxDECLARE_DYNAMIC_CLASS and wxIMPLEMENT_DYNAMIC_CLASS.

Library: wxBase

Category: Runtime Type Information (RTTI)

See also

wxClassInfo, wxObject

Public Member Functions

- wxClassInfo (const wxChar *className, const wxClassInfo *baseClass1, const wxClassInfo *baseClass2, int size, wxObjectConstructorFn fn)

  Constructs a wxClassInfo object.
- wxObject * CreateObject () const

  Creates an object of the appropriate kind.
- const wxChar * GetBaseClassName1 () const

  Returns the name of the first base class (NULL if none).
- const wxChar * GetBaseClassName2 () const

  Returns the name of the second base class (NULL if none).
- const wxChar * GetClassName () const

  Returns the string form of the class name.
- int GetSize () const

  Returns the size of the class.
- bool IsDynamic () const

  Returns true if this class info can create objects of the associated class.
- bool IsKindOf (const wxClassInfo *info) const

  Returns true if this class is a kind of (inherits from) the given class.

Static Public Member Functions

- static wxClassInfo * FindClass (const wxString &className)

  Finds the wxClassInfo object for a class with the given name.

21.88.2 Constructor & Destructor Documentation

wxClassInfo::wxClassInfo (const wxChar * *className*, const wxClassInfo * *baseClass1*, const wxClassInfo * *baseClass2*, int *size*, wxObjectConstructorFn *fn*)

Constructs a wxClassInfo object.

The supplied macros implicitly construct objects of this class, so there is no need to create such objects explicitly in an application.

21.88.3 Member Function Documentation

wxObject* wxClassInfo::CreateObject ( ) const

Creates an object of the appropriate kind.

Returns

NULL if the class has not been declared dynamically creatable (typically, this happens for abstract classes).

static wxClassInfo* wxClassInfo::FindClass (const wxString & *className*) [static]

Finds the wxClassInfo object for a class with the given *name*.

const wxChar* wxClassInfo::GetBaseClassName1 ( ) const

Returns the name of the first base class (NULL if none).

const wxChar* wxClassInfo::GetBaseClassName2 ( ) const

Returns the name of the second base class (NULL if none).

const wxChar* wxClassInfo::GetClassName ( ) const

Returns the string form of the class name.

int wxClassInfo::GetSize ( ) const

Returns the size of the class.

bool wxClassInfo::IsDynamic ( ) const

Returns true if this class info can create objects of the associated class.

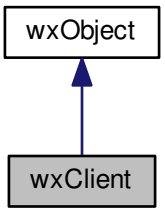
bool wxClassInfo::IsKindOf (const wxClassInfo * *info*) const

Returns true if this class is a kind of (inherits from) the given class.

21.89 wxClient Class Reference

`#include <wx/ipc.h>`

Inheritance diagram for wxClient:



21.89.1 Detailed Description

A wxClient object represents the client part of a client-server DDE-like (Dynamic Data Exchange) conversation.

The actual DDE-based implementation using wxDDEClient is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPClient, which has the same API.

To create a client which can communicate with a suitable server, you need to derive a class from wxConnection and another from wxClient. The custom wxConnection class will intercept communications in a 'conversation' with a server, and the custom wxClient is required so that a user-overridden wxClient::OnMakeConnection member can return a wxConnection of the required class, when a connection is made.

Look at the IPC sample and the Interprocess Communication for an example of how to do this.

Library: wxBase

Category: Interprocess Communication

See also

wxServer, wxConnection, Interprocess Communication

Public Member Functions

- wxClient ()

  Constructs a client object.
- wxConnectionBase * MakeConnection (const wxString &host, const wxString &service, const wxString &topic)

  Tries to make a connection with a server by host (machine name under UNIX - use 'localhost' for same machine; ignored when using native DDE in Windows), service name and topic string.
- wxConnectionBase * OnMakeConnection ()

  Called by MakeConnection(), by default this simply returns a new wxConnection object.
- bool ValidHost (const wxString &host)

  Returns true if this is a valid host name, false otherwise.

Additional Inherited Members

21.89.2 Constructor & Destructor Documentation

wxClient::wxClient ( )

Constructs a client object.

21.89.3 Member Function Documentation

wxConnectionBase* wxClient::MakeConnection (const wxString & *host,* const wxString & *service,* const wxString & *topic*)

Tries to make a connection with a server by host (machine name under UNIX - use 'localhost' for same machine; ignored when using native DDE in Windows), service name and topic string.

If the server allows a connection, a wxConnection object will be returned. The type of wxConnection returned can be altered by overriding the OnMakeConnection() member to return your own derived connection object.

Under Unix, the service name may be either an integer port identifier in which case an Internet domain socket will be used for the communications, or a valid file name (which shouldn't exist and will be deleted afterwards) in which case a Unix domain socket is created.

Note

Using Internet domain sockets is extremely insecure for IPC as there is absolutely no access control for them, use Unix domain sockets whenever possible!

wxConnectionBase* wxClient::OnMakeConnection ( )

Called by MakeConnection(), by default this simply returns a new wxConnection object.

Override this method to return a wxConnection descendant customised for the application.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as wxConnection::OnAdvise. You may also want to store application-specific data in instances of the new class.

bool wxClient::ValidHost (const wxString & *host*)

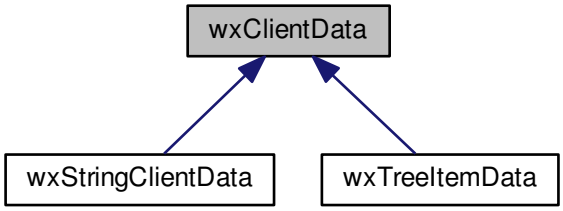
Returns true if this is a valid host name, false otherwise.

This always returns true under MS Windows.

21.90 wxClientData Class Reference

`#include <wx/clntdata.h>`

Inheritance diagram for wxClientData:



21.90.1 Detailed Description

All classes deriving from wxEvtHandler (such as all controls and wxApp) can hold arbitrary data which is here referred to as "client data".

This is useful e.g. for scripting languages which need to handle shadow objects for most of wxWidgets' classes and which store a handle to such a shadow class as client data in that class. This data can either be of type void - in which case the data *container* does not take care of freeing the data again or it is of type wxClientData or its derivatives. In that case the container (e.g. a control) will free the memory itself later. Note that you *must* not assign both void data and data derived from the wxClientData class to a container.

Some controls can hold various items and these controls can additionally hold client data for each item. This is the case for wxChoice, wxComboBox and wxListBox. wxTreeCtrl has a specialized class wxTreeItemData for each item in the tree.

If you want to add client data to your own classes, you may use the mix-in class wxClientDataContainer.

Library: wxBase

Category: Containers

See also

wxEvtHandler, wxTreeItemData, wxStringClientData, wxClientDataContainer

Public Member Functions

- wxClientData ()

  Constructor.
- virtual ~wxClientData ()

  Virtual destructor.

21.90.2 Constructor & Destructor Documentation

wxClientData::wxClientData ( )

Constructor.

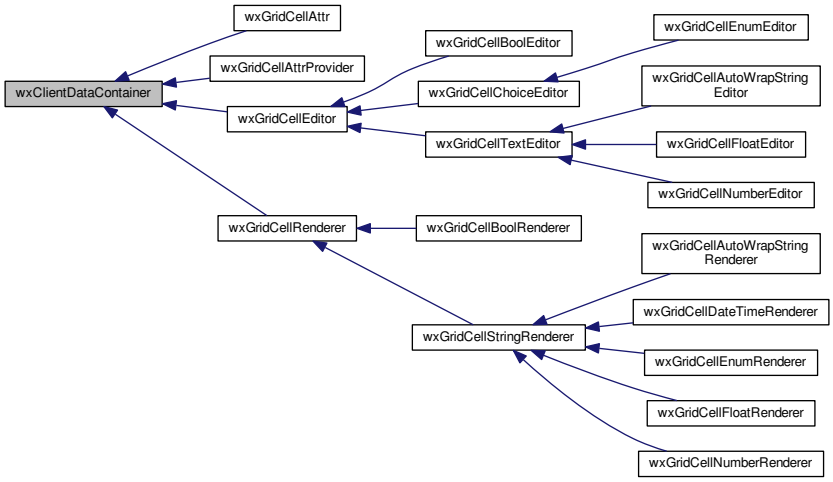
virtual wxClientData::~wxClientData ( ) [virtual]

Virtual destructor.

21.91 wxClientDataContainer Class Reference

`#include <wx/clntdata.h>`

Inheritance diagram for wxClientDataContainer:



21.91.1 Detailed Description

This class is a mixin that provides storage and management of "client data".

This data can either be of type void - in which case the data *container* does not take care of freeing the data again or it is of type wxClientData or its derivatives. In that case the container will free the memory itself later. Note that you *must* not assign both void data and data derived from the wxClientData class to a container.

Note

This functionality is currently duplicated in wxEvtHandler in order to avoid having more than one vtable in that class hierarchy.

Library: wxBase

Category: Containers

See also

wxEvtHandler, wxClientData

Public Member Functions

- wxClientDataContainer ()

Default constructor.

- virtual ~wxClientDataContainer ()

  Destructor.

- void * GetClientData () const

  Get the untyped client data.

- wxClientData * GetClientObject () const

  Get a pointer to the client data object.

- void SetClientData (void *data)

  Set the untyped client data.

- void SetClientObject (wxClientData *data)

  Set the client data object.

21.91.2 Constructor & Destructor Documentation

wxClientDataContainer::wxClientDataContainer ( )

Default constructor.

virtual wxClientDataContainer::~wxClientDataContainer ( ) [virtual]

Destructor.

21.91.3 Member Function Documentation

void* wxClientDataContainer::GetClientData ( ) const

Get the untyped client data.

wxClientData* wxClientDataContainer::GetClientObject ( ) const

Get a pointer to the client data object.

void wxClientDataContainer::SetClientData (void * *data*)

Set the untyped client data.

void wxClientDataContainer::SetClientObject (wxClientData * *data*)

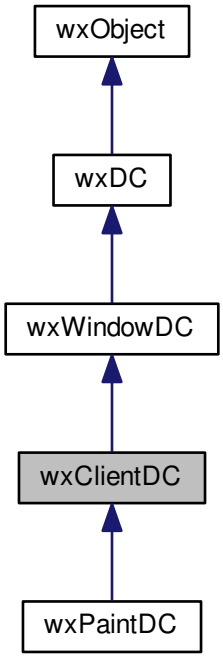
Set the client data object.

Any previous object will be deleted.

21.92 wxClientDC Class Reference

`#include <wx/dcclient.h>`

Inheritance diagram for wxClientDC:



21.92.1 Detailed Description

A wxClientDC must be constructed if an application wishes to paint on the client area of a window from outside an EVT_PAINT() handler.

This should normally be constructed as a temporary stack object; don't store a wxClientDC object.

To draw on a window from within an EVT_PAINT() handler, construct a wxPaintDC object instead.

To draw on the whole window including decorations, construct a wxWindowDC object (Windows only).

A wxClientDC object is initialized to use the same font and colours as the window it is associated with.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxMemoryDC, wxPaintDC, wxWindowDC, wxScreenDC

Public Member Functions

- wxClientDC (wxWindow *window)

  Constructor.

Additional Inherited Members

21.92.2 Constructor & Destructor Documentation

wxClientDC::wxClientDC (wxWindow * *window*)

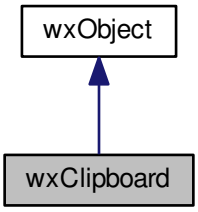
Constructor.

Pass a pointer to the window on which you wish to paint.

21.93 wxClipboard Class Reference

`#include <wx/clipbrd.h>`

Inheritance diagram for wxClipboard:



21.93.1 Detailed Description

A class for manipulating the clipboard.

To use the clipboard, you call member functions of the global wxTheClipboard object.

See the wxDataObject Overview for further information.

Call wxClipboard::Open() to get ownership of the clipboard. If this operation returns true, you now own the clipboard. Call wxClipboard::SetData() to put data on the clipboard, or wxClipboard::GetData() to retrieve data from the clipboard. Call wxClipboard::Close() to close the clipboard and relinquish ownership. You should keep the clipboard open only momentarily.

For example:

```
// Write some text to the clipboard
if (wxTheClipboard->Open())
{
    // This data objects are held by the clipboard,
    // so do not delete them in the app.
    wxTheClipboard->SetData( new wxTextDataObject("Some text") );
    wxTheClipboard->Close();
}

// Read some text
if (wxTheClipboard->Open())
{
    if (wxTheClipboard->IsSupported( wxDF_TEXT ))
    {
        wxTextDataObject data;
        wxTheClipboard->GetData( data );
        wxMessageBox( data.GetText() );
    }
```

```
    wxTheClipboard->Close();
}
```

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject Overview, wxDataObject

Public Member Functions

- wxClipboard ()

  Default constructor.
- virtual ~wxClipboard ()

  Destructor.
- virtual bool AddData (wxDataObject *data)

  Call this function to add the data object to the clipboard.
- virtual void Clear ()

  Clears the global clipboard object and the system's clipboard if possible.
- virtual void Close ()

  Call this function to close the clipboard, having opened it with Open().
- virtual bool Flush ()

  Flushes the clipboard: this means that the data which is currently on clipboard will stay available even after the application exits (possibly eating memory), otherwise the clipboard will be emptied on exit.
- virtual bool GetData (wxDataObject &data)

  Call this function to fill data with data on the clipboard, if available in the required format.
- virtual bool IsOpened () const

  Returns true if the clipboard has been opened.
- virtual bool IsSupported (const wxDataFormat &format)

  *Returns true if there is data which matches the data format of the given data object currently **available** on the clipboard.*
- bool IsUsingPrimarySelection () const

  Returns true if we are using the primary selection, false if clipboard one.
- virtual bool Open ()

  Call this function to open the clipboard before calling SetData() and GetData().
- virtual bool SetData (wxDataObject *data)

  Call this function to set the data object to the clipboard.
- virtual void UsePrimarySelection (bool primary=false)

  On platforms supporting it (all X11-based ports), wxClipboard uses the CLIPBOARD X11 selection by default.

Static Public Member Functions

- static wxClipboard * Get ()

  Returns the global instance (wxTheClipboard) of the clipboard object.

Additional Inherited Members

21.93.2 Constructor & Destructor Documentation

wxClipboard::wxClipboard ( )

Default constructor.

virtual wxClipboard::~wxClipboard ( ) [virtual]

Destructor.

21.93.3 Member Function Documentation

virtual bool wxClipboard::AddData (wxDataObject * *data*) [virtual]

Call this function to add the data object to the clipboard.

You may call this function repeatedly after having cleared the clipboard using Clear().

After this function has been called, the clipboard owns the data, so do not delete the data explicitly.

See also

SetData()

virtual void wxClipboard::Clear ( ) [virtual]

Clears the global clipboard object and the system's clipboard if possible.

virtual void wxClipboard::Close ( ) [virtual]

Call this function to close the clipboard, having opened it with Open().

virtual bool wxClipboard::Flush ( ) [virtual]

Flushes the clipboard: this means that the data which is currently on clipboard will stay available even after the application exits (possibly eating memory), otherwise the clipboard will be emptied on exit.

Currently this method is not implemented in X11-based ports, i.e. wxGTK, wxX11 and wxMotif and always returns false there.

Returns

false if the operation is unsuccessful for any reason.

static wxClipboard* wxClipboard::Get ( ) [static]

Returns the global instance (wxTheClipboard) of the clipboard object.

virtual bool wxClipboard::GetData (wxDataObject & *data*) [virtual]

Call this function to fill *data* with data on the clipboard, if available in the required format.

Returns true on success.

virtual bool wxClipboard::IsOpened ( ) const `[virtual]`

Returns true if the clipboard has been opened.

virtual bool wxClipboard::IsSupported (const wxDataFormat & *format*) `[virtual]`

Returns true if there is data which matches the data format of the given data object currently **available** on the clipboard.

Todo The name of this function is misleading. This should be renamed to something that more accurately indicates what it does.

bool wxClipboard::IsUsingPrimarySelection ( ) const

Returns true if we are using the primary selection, false if clipboard one.

See also

UsePrimarySelection()

virtual bool wxClipboard::Open ( ) `[virtual]`

Call this function to open the clipboard before calling SetData() and GetData().

Call Close() when you have finished with the clipboard. You should keep the clipboard open for only a very short time.

Returns

true on success. This should be tested (as in the sample shown above).

virtual bool wxClipboard::SetData (wxDataObject * *data*) `[virtual]`

Call this function to set the data object to the clipboard.

This function will clear all previous contents in the clipboard, so calling it several times does not make any sense.

After this function has been called, the clipboard owns the data, so do not delete the data explicitly.

See also

AddData()

virtual void wxClipboard::UsePrimarySelection (bool *primary* = `false`) `[virtual]`

On platforms supporting it (all X11-based ports), wxClipboard uses the CLIPBOARD X11 selection by default.

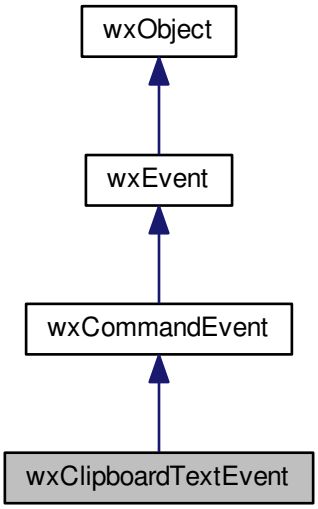
When this function is called with true, all subsequent clipboard operations will use PRIMARY selection until this function is called again with false.

On the other platforms, there is no PRIMARY selection and so all clipboard operations will fail. This allows to implement the standard X11 handling of the clipboard which consists in copying data to the CLIPBOARD selection only when the user explicitly requests it (i.e. by selecting the "Copy" menu command) but putting the currently selected text into the PRIMARY selection automatically, without overwriting the normal clipboard contents with the currently selected text on the other platforms.

21.94 wxClipboardTextEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxClipboardTextEvent:



21.94.1 Detailed Description

This class represents the events generated by a control (typically a wxTextCtrl but other windows can generate these events as well) when its content gets copied or cut to, or pasted from the clipboard.

There are three types of corresponding events `wxEVT_TEXT_COPY`, `wxEVT_TEXT_CUT` and `wxEVT_TEXT_PASTE`.

If any of these events is processed (without being skipped) by an event handler, the corresponding operation doesn't take place which allows to prevent the text from being copied from or pasted to a control. It is also possible to examine the clipboard contents in the PASTE event handler and transform it in some way before inserting in a control – for example, changing its case or removing invalid characters.

Finally notice that a CUT event is always preceded by the COPY event which makes it possible to only process the latter if it doesn't matter if the text was copied or cut.

Note

These events are currently only generated by wxTextCtrl in wxGTK and wxOSX but are also generated by wxComboBox without wxCB_READONLY style in wxMSW.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxClipboardTextEvent& event)

Event macros:

- EVT_TEXT_COPY(id, func): Some or all of the controls content was copied to the clipboard.

- EVT_TEXT_CUT(id, func): Some or all of the controls content was cut (i.e. copied and deleted).
- EVT_TEXT_PASTE(id, func): Clipboard content was pasted into the control.

Library: wxCore

Category: Events

See also

wxClipboard

Public Member Functions

- wxClipboardTextEvent (wxEventType commandType=wxEVT_NULL, int id=0)

  Constructor.

Additional Inherited Members

21.94.2 Constructor & Destructor Documentation

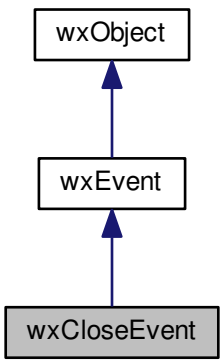
wxClipboardTextEvent::wxClipboardTextEvent (wxEventType *commandType* = wxEVT_NULL, int *id* = 0)

Constructor.

21.95 wxCloseEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxCloseEvent:



21.95.1 Detailed Description

This event class contains information about window and session close events.

The handler function for EVT_CLOSE is called when the user has tried to close a a frame or dialog box using the window manager (X) or system menu (Windows). It can also be invoked by the application itself programmatically, for example by calling the wxWindow::Close function.

You should check whether the application is forcing the deletion of the window using wxCloseEvent::CanVeto. If this is false, you *must* destroy the window using wxWindow::Destroy.

If the return value is true, it is up to you whether you respond by destroying the window.

If you don't destroy the window, you should call wxCloseEvent::Veto to let the calling code know that you did not destroy the window. This allows the wxWindow::Close function to return true or false depending on whether the close instruction was honoured or not.

Example of a wxCloseEvent handler:

```
void MyFrame::OnClose(wxCloseEvent& event)
{
    if ( event.CanVeto() && m_bFileNotSaved )
    {
        if ( wxMessageBox("The file has not been saved... continue closing?",
                          "Please confirm",
                          wxICON_QUESTION | wxYES_NO) !=
      wxYES )
        {
            event.Veto();
            return;
        }
    }

    Destroy();  // you may also do:  event.Skip();
                // since the default event handler does call Destroy(), too
}
```

The EVT_END_SESSION event is slightly different as it is sent by the system when the user session is ending (e.g. because of log out or shutdown) and so all windows are being forcefully closed. At least under MSW, after the handler for this event is executed the program is simply killed by the system. Because of this, the default handler for this event provided by wxWidgets calls all the usual cleanup code (including wxApp::OnExit()) so that it could still be executed and exit()s the process itself, without waiting for being killed. If this behaviour is for some reason undesirable, make sure that you define a handler for this event in your wxApp-derived class and do not call `event.Skip()` in it (but be aware that the system will still kill your application).

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCloseEvent& event)

Event macros:

- EVT_CLOSE(func): Process a `wxEVT_CLOSE_WINDOW` command event, supplying the member function. This event applies to wxFrame and wxDialog classes.
- EVT_QUERY_END_SESSION(func): Process a `wxEVT_QUERY_END_SESSION` session event, supplying the member function. This event can be handled in wxApp-derived class only.
- EVT_END_SESSION(func): Process a `wxEVT_END_SESSION` session event, supplying the member function. This event can be handled in wxApp-derived class only.

Library: wxCore

Category: Events

See also

wxWindow::Close, Window Deletion

Public Member Functions

- wxCloseEvent (wxEventType commandEventType=wxEVT_NULL, int id=0)

  Constructor.
- bool CanVeto () const

  Returns true if you can veto a system shutdown or a window close event.
- bool GetLoggingOff () const

  Returns true if the user is just logging off or false if the system is shutting down.
- void SetCanVeto (bool canVeto)

  Sets the 'can veto' flag.
- void SetLoggingOff (bool loggingOff)

  Sets the 'logging off' flag.
- void Veto (bool veto=true)

  Call this from your event handler to veto a system shutdown or to signal to the calling application that a window close did not happen.
- bool GetVeto () const

  Returns whether the Veto flag was set.

Additional Inherited Members

21.95.2 Constructor & Destructor Documentation

wxCloseEvent::wxCloseEvent (wxEventType *commandEventType* = wxEVT_NULL, int *id* = 0)

Constructor.

21.95.3 Member Function Documentation

bool wxCloseEvent::CanVeto ( ) const

Returns true if you can veto a system shutdown or a window close event.

Vetoing a window close event is not possible if the calling code wishes to force the application to exit, and so this function must be called to check this.

bool wxCloseEvent::GetLoggingOff ( ) const

Returns true if the user is just logging off or false if the system is shutting down.

This method can only be called for end session and query end session events, it doesn't make sense for close window event.

bool wxCloseEvent::GetVeto ( ) const

Returns whether the Veto flag was set.

void wxCloseEvent::SetCanVeto (bool *canVeto*)

Sets the 'can veto' flag.

void wxCloseEvent::SetLoggingOff (bool *loggingOff*)

Sets the 'logging off' flag.

void wxCloseEvent::Veto (bool *veto* = `true`)

Call this from your event handler to veto a system shutdown or to signal to the calling application that a window close did not happen.

You can only veto a shutdown if CanVeto() returns true.

21.96 wxCmdLineArg Class Reference

`#include <wx/cmdline.h>`

21.96.1 Detailed Description

The interface wxCmdLineArg provides information for an instance of argument passed on command line.

Example of use:

```
wxCmdLineParser parser;

for (wxCmdLineArgs::const_iterator itarg=parser.GetArguments().begin();
                                   itarg!=parser.GetArguments().end();
                                   ++itarg)
{
    wxString optionName;
    switch (itarg->GetKind())
    {
    case wxCMD_LINE_SWITCH:
        if (itarg->IsNegated()) {
        }
        else {
        }
        break;

    case wxCMD_LINE_OPTION:
        // assuming that all the options have a short name
        optionName = itarg->GetShortName();

        switch (itarg->GetType()) {
            case wxCMD_LINE_VAL_NUMBER:
                // do something with itarg->GetLongVal();
                break;

            case wxCMD_LINE_VAL_DOUBLE:
                // do something with itarg->GetDoubleVal();
                break;

            case wxCMD_LINE_VAL_DATE:
                // do something with itarg->GetDateVal();
                break;

            case wxCMD_LINE_VAL_STRING:
                // do something with itarg->GetStrVal();
                break;
        }
        break;

    case wxCMD_LINE_PARAM:
        // do something with itarg->GetStrVal();
        break;
    }
}
```

With C++11, the for loop could be written:

```
for (const auto &arg : parser.GetArguments()) {
    // working on arg as with *itarg above
}
```

Since

3.1.0

Public Member Functions

- virtual ~wxCmdLineArg ()
- virtual const wxDateTime & GetDateVal () const =0

  Returns the command line argument value as a wxDateTime.
- virtual double GetDoubleVal () const =0

  Returns the command line argument value as a double.
- virtual wxCmdLineEntryType GetKind () const =0

  Returns the command line argument entry kind.
- virtual long GetLongVal () const =0

  Returns the command line argument value as a long.
- virtual wxString GetLongName () const =0

  Returns the command line argument long name if any.
- virtual wxString GetShortName () const =0

  Returns the command line argument short name if any.
- virtual const wxString & GetStrVal () const =0

  Returns the command line argument value as a string.
- virtual wxCmdLineParamType GetType () const =0

  Returns the command line argument parameter type.
- virtual bool IsNegated () const =0

  Returns true if the switch was negated.

21.96.2 Constructor & Destructor Documentation

virtual wxCmdLineArg::~wxCmdLineArg ( ) [inline], [virtual]

21.96.3 Member Function Documentation

virtual const wxDateTime& wxCmdLineArg::GetDateVal ( ) const [pure virtual]

Returns the command line argument value as a wxDateTime.

Note

This call works only for `wxCMD_LINE_VAL_DATE` options

virtual double wxCmdLineArg::GetDoubleVal ( ) const [pure virtual]

Returns the command line argument value as a double.

Note

This call works only for `wxCMD_LINE_VAL_DOUBLE` options

virtual wxCmdLineEntryType wxCmdLineArg::GetKind ( ) const `[pure virtual]`

Returns the command line argument entry kind.

Note

Parameters can only be retrieved as strings, with GetStrVal()

See also

wxCmdLineEntryType, GetType()

virtual wxString wxCmdLineArg::GetLongName ( ) const `[pure virtual]`

Returns the command line argument long name if any.

Note

This call makes sense only for options and switches

virtual long wxCmdLineArg::GetLongVal ( ) const `[pure virtual]`

Returns the command line argument value as a long.

Note

This call works only for `wxCMD_LINE_VAL_NUMBER` options

virtual wxString wxCmdLineArg::GetShortName ( ) const `[pure virtual]`

Returns the command line argument short name if any.

Note

This call makes sense only for options and switches

virtual const wxString& wxCmdLineArg::GetStrVal ( ) const `[pure virtual]`

Returns the command line argument value as a string.

Note

This call works only for `wxCMD_LINE_VAL_STRING` options and parameters

virtual wxCmdLineParamType wxCmdLineArg::GetType ( ) const `[pure virtual]`

Returns the command line argument parameter type.

Note

This call makes sense only for options (i.e. GetKind() == `wxCMD_LINE_OPTION`).

See also

wxCmdLineParamType, GetKind()

virtual bool wxCmdLineArg::IsNegated ( ) const `[pure virtual]`

Returns true if the switch was negated.

Note

This call works only for switches.

21.97 wxCmdLineArgs Class Reference

`#include <wx/cmdline.h>`

21.97.1 Detailed Description

An ordered collection of wxCmdLineArg providing an iterator to enumerate the arguments passed on command line.

See also

wxCmdLineParser::GetArguments()

Since

3.1.0

Public Member Functions

- const_iterator begin () const
- const_iterator end () const
- size_t size () const

  Returns the number of command line arguments in this collection.

21.97.2 Member Function Documentation

const_iterator wxCmdLineArgs::begin ( ) const

const_iterator wxCmdLineArgs::end ( ) const

size_t wxCmdLineArgs::size ( ) const

Returns the number of command line arguments in this collection.

21.98 wxCmdLineEntryDesc Struct Reference

`#include <wx/cmdline.h>`

21.98.1 Detailed Description

The structure wxCmdLineEntryDesc is used to describe a command line switch, option or parameter.

An array of such structures should be passed to wxCmdLineParser::SetDesc().

Note that the meanings of parameters of the wxCmdLineParser::AddXXX() functions are the same as of the corresponding fields in this structure.

Public Attributes

- wxCmdLineEntryType kind

  The kind of this program argument.
- const char ∗ shortName

  The usual, short, name of the switch or the option.
- const char ∗ longName

  The long name for this program argument (may be empty if the option has no long name).
- const char ∗ description

  This description is used by the wxCmdLineParser::Usage() method to construct a help message explaining the syntax of the program.
- wxCmdLineParamType type

  The type associated with this option (ignored if `kind != wxCMD_LINE_OPTION`).
- int flags

  A combination of one or more wxCmdLineEntryFlags enum values.

21.98.2 Member Data Documentation

const char∗ wxCmdLineEntryDesc::description

This description is used by the wxCmdLineParser::Usage() method to construct a help message explaining the syntax of the program.

int wxCmdLineEntryDesc::flags

A combination of one or more wxCmdLineEntryFlags enum values.

wxCmdLineEntryType wxCmdLineEntryDesc::kind

The kind of this program argument.

See wxCmdLineEntryType for more info.

const char∗ wxCmdLineEntryDesc::longName

The long name for this program argument (may be empty if the option has no long name).

It may contain only letters, digits and the underscores. This field is unused if `kind == wxCMD_LINE_PARAM`.

const char∗ wxCmdLineEntryDesc::shortName

The usual, short, name of the switch or the option.

It may contain only letters, digits and the underscores. This field is unused if `kind == wxCMD_LINE_PARAM`.

wxCmdLineParamType wxCmdLineEntryDesc::type

The type associated with this option (ignored if `kind != wxCMD_LINE_OPTION`).

See wxCmdLineParamType for more info.

21.99 wxCmdLineParser Class Reference

`#include <wx/cmdline.h>`

21.99.1 Detailed Description

wxCmdLineParser is a class for parsing the command line.

It has the following features:

- distinguishes options, switches and parameters
- allows option grouping
- allows both short and long options
- automatically generates the usage message from the command line description
- checks types of the options values (number, date, ...).

To use it you should follow these steps:

1. Construct an object of this class giving it the command line to parse and optionally its description or use the `AddXXX()` functions later.
2. Call Parse().
3. Use Found() to retrieve the results.

You can also use wxApp's default command line processing just overriding wxAppConsole::OnInitCmdLine() and wxAppConsole::OnCmdLineParsed().

In the documentation below the following terminology is used:

- **switch:** a boolean option which can be given or not, but which doesn't have any value. We use the word *switch* to distinguish such boolean options from more generic options like those described below. For example, `"-v"` might be a switch meaning "enable verbose mode".
- **option:** a switch with a value associated to it. For example, `"-o filename"` might be an option for specifying the name of the output file.
- **parameter:** a required program argument.

21.99.2 Construction

Before Parse() can be called, the command line parser object must have the command line to parse and also the rules saying which switches, options and parameters are valid - this is called command line description in what follows.

You have complete freedom of choice as to when specify the required information, the only restriction is that it must be done before calling Parse().

To specify the command line to parse you may use either one of constructors accepting it (wxCmdLineParser(int, char**) or wxCmdLineParser(const wxString&) usually) or, if you use the default constructor, you can do it later by calling SetCmdLine().

The same holds for command line description: it can be specified either in the constructor (with or without the command line itself) or constructed later using either SetDesc() or combination of AddSwitch(), AddOption(), AddParam() and AddUsageText() methods.

Using constructors or SetDesc() uses a (usually const static) table containing the command line description. If you want to decide which options to accept during the run-time, using one of the AddXXX() functions above might be preferable.

21.99.3 Customization

wxCmdLineParser has several global options which may be changed by the application. All of the functions described in this section should be called before Parse().

First global option is the support for long (also known as GNU-style) options. The long options are the ones which start with two dashes and look like "\--verbose", i.e. they generally are complete words and not some abbreviations of them. As long options are used by more and more applications, they are enabled by default, but may be disabled with DisableLongOptions().

Another global option is the set of characters which may be used to start an option (otherwise, the word on the command line is assumed to be a parameter). Under Unix, "-" is always used, but Windows has at least two common choices for this: "-" and "/". Some programs also use "+". The default is to use what suits most the current platform, but may be changed with SetSwitchChars() method.

Finally, SetLogo() can be used to show some application-specific text before the explanation given by Usage() function.

21.99.4 Parsing the Command Line

After the command line description was constructed and the desired options were set, you can finally call Parse() method. It returns 0 if the command line was correct and was parsed, -1 if the help option was specified (this is a separate case as, normally, the program will terminate after this) or a positive number if there was an error during the command line parsing.

In the latter case, the appropriate error message and usage information are logged by wxCmdLineParser itself using the standard wxWidgets logging functions.

21.99.5 Getting Results

After calling Parse() (and if it returned 0), you may access the results of parsing using one of overloaded Found() methods.

For a simple switch, you will simply call Found to determine if the switch was given or not, for an option or a parameter, you will call a version of Found() which also returns the associated value in the provided variable. All Found() functions return true if the switch or option were found in the command line or false if they were not specified.

Library: wxBase

Category: Application and Process Management

See also

wxApp::argc, wxApp::argv, Console Program Sample

Public Member Functions

- wxCmdLineParser ()

  Default constructor, you must use SetCmdLine() later.
- wxCmdLineParser (int argc, char **argv)

  Constructor which specifies the command line to parse.
- wxCmdLineParser (int argc, wchar_t **argv)

  Constructor which specifies the command line to parse.
- wxCmdLineParser (const wxString &cmdline)

  Constructor which specify the command line to parse in Windows format.

- wxCmdLineParser (const wxCmdLineEntryDesc *desc)

  Specifies the command line description but not the command line.
- wxCmdLineParser (const wxCmdLineEntryDesc *desc, int argc, char **argv)

  Specifies both the command line (in Unix format) and the command line description.
- wxCmdLineParser (const wxCmdLineEntryDesc *desc, const wxString &cmdline)

  Specifies both the command line (in Windows format) and the command line description.
- ~wxCmdLineParser ()

  Frees resources allocated by the object.
- void AddLongOption (const wxString &lng, const wxString &desc=wxEmptyString, wxCmdLineParamType type=wxCMD_LINE_VAL_STRING, int flags=0)

  Adds an option with only long form.
- void AddLongSwitch (const wxString &lng, const wxString &desc=wxEmptyString, int flags=0)

  Adds a switch with only long form.
- void AddOption (const wxString &name, const wxString &lng=wxEmptyString, const wxString &desc=wxEmptyString, wxCmdLineParamType type=wxCMD_LINE_VAL_STRING, int flags=0)

  Add an option name with an optional long name lng (no long name if it is empty, which is default) taking a value of the given type (string by default) to the command line description.
- void AddParam (const wxString &desc=wxEmptyString, wxCmdLineParamType type=wxCMD_LINE_VAL_STRING, int flags=0)

  Add a parameter of the given type to the command line description.
- void AddSwitch (const wxString &name, const wxString &lng=wxEmptyString, const wxString &desc=wxEmptyString, int flags=0)

  Add a switch name with an optional long name lng (no long name if it is empty, which is default), description desc and flags flags to the command line description.
- void AddUsageText (const wxString &text)

  Add a string text to the command line description shown by Usage().
- bool AreLongOptionsEnabled () const

  Returns true if long options are enabled, otherwise false.
- void DisableLongOptions ()

  Identical to EnableLongOptions(false).
- void EnableLongOptions (bool enable=true)

  Enable or disable support for the long options.
- bool Found (const wxString &name) const

  Returns true if the given switch was found, false otherwise.
- wxCmdLineSwitchState FoundSwitch (const wxString &name) const

  Returns whether the switch was found on the command line and whether it was negated.
- bool Found (const wxString &name, wxString *value) const

  Returns true if an option taking a string value was found and stores the value in the provided pointer (which should not be NULL).
- bool Found (const wxString &name, long *value) const

  Returns true if an option taking an integer value was found and stores the value in the provided pointer (which should not be NULL).
- bool Found (const wxString &name, double *value) const

  Returns true if an option taking a float value was found and stores the value in the provided pointer (which should not be NULL).
- bool Found (const wxString &name, wxDateTime *value) const

  Returns true if an option taking a date value was found and stores the value in the provided pointer (which should not be NULL).
- wxString GetParam (size_t n=0) const

  Returns the value of Nth parameter (as string only).
- size_t GetParamCount () const

  Returns the number of parameters found.

- wxCmdLineArgs GetArguments () const

  Returns the collection of arguments.
- int Parse (bool giveUsage=true)

  Parse the command line, return 0 if ok, -1 if "-h" or "\--help" option was encountered and the help message was given or a positive value if a syntax error occurred.
- void SetDesc (const wxCmdLineEntryDesc *desc)

  Constructs the command line description.
- void SetLogo (const wxString &logo)

  The logo is some extra text which will be shown by Usage() method.
- void SetSwitchChars (const wxString &switchChars)

  switchChars contains all characters with which an option or switch may start.
- void Usage () const

  Give the standard usage message describing all program options.
- wxString GetUsageString () const

  Return the string containing the program usage description.

- void SetCmdLine (int argc, char **argv)

  Set the command line to parse after using one of the constructors which don't do it.
- void SetCmdLine (int argc, wchar_t **argv)

  Set the command line to parse after using one of the constructors which don't do it.
- void SetCmdLine (const wxString &cmdline)

  Set the command line to parse after using one of the constructors which don't do it.

Static Public Member Functions

- static wxArrayString ConvertStringToArgs (const wxString &cmdline, wxCmdLineSplitType flags=wxCMD_LINE_SPLIT_DOS)

  Breaks down the string containing the full command line in words.

21.99.6 Constructor & Destructor Documentation

wxCmdLineParser::wxCmdLineParser ( )

Default constructor, you must use SetCmdLine() later.

wxCmdLineParser::wxCmdLineParser (int *argc*, char ** *argv*)

Constructor which specifies the command line to parse.

This is the traditional (Unix) command line format. The parameters *argc* and *argv* have the same meaning as the typical `main()` function.

This constructor is available in both ANSI and Unicode modes because under some platforms the command line arguments are passed as ASCII strings even to Unicode programs.

wxCmdLineParser::wxCmdLineParser (int *argc*, wchar_t ** *argv*)

Constructor which specifies the command line to parse.

This is the traditional (Unix) command line format.

The parameters *argc* and *argv* have the same meaning as the typical `main()` function.

This constructor is only available in Unicode build.

wxCmdLineParser::wxCmdLineParser (const wxString & *cmdline*)

Constructor which specify the command line to parse in Windows format.

The parameter cmdline has the same meaning as the corresponding parameter of `WinMain()`.

wxCmdLineParser::wxCmdLineParser (const wxCmdLineEntryDesc * *desc*)

Specifies the command line description but not the command line.

You must use SetCmdLine() later.

wxCmdLineParser::wxCmdLineParser (const wxCmdLineEntryDesc * *desc*, int *argc*, char ** *argv*)

Specifies both the command line (in Unix format) and the command line description.

wxCmdLineParser::wxCmdLineParser (const wxCmdLineEntryDesc * *desc*, const wxString & *cmdline*)

Specifies both the command line (in Windows format) and the command line description.

wxCmdLineParser::~wxCmdLineParser ( )

Frees resources allocated by the object.

Note

This destructor is not virtual, don't use this class polymorphically.

21.99.7 Member Function Documentation

void wxCmdLineParser::AddLongOption (const wxString & *lng*, const wxString & *desc* = wxEmptyString, wxCmdLineParamType *type* = wxCMD_LINE_VAL_STRING, int *flags* = 0)

Adds an option with only long form.

This is just a convenient wrapper for AddOption() passing an empty string as short option name.

Since

2.9.3

void wxCmdLineParser::AddLongSwitch (const wxString & *lng*, const wxString & *desc* = wxEmptyString, int *flags* = 0)

Adds a switch with only long form.

This is just a convenient wrapper for AddSwitch() passing an empty string as short switch name.

Since

2.9.3

void wxCmdLineParser::AddOption (const wxString & *name*, const wxString & *lng* = wxEmptyString, const wxString & *desc* = wxEmptyString, wxCmdLineParamType *type* = wxCMD_LINE_VAL_STRING, int *flags* = 0)

Add an option *name* with an optional long name *lng* (no long name if it is empty, which is default) taking a value of the given type (string by default) to the command line description.

void wxCmdLineParser::AddParam (const wxString & *desc* = wxEmptyString, wxCmdLineParamType *type* = wxCMD_LINE_VAL_STRING, int *flags* = 0)

Add a parameter of the given *type* to the command line description.

void wxCmdLineParser::AddSwitch (const wxString & *name*, const wxString & *lng* = wxEmptyString, const wxString & *desc* = wxEmptyString, int *flags* = 0)

Add a switch *name* with an optional long name *lng* (no long name if it is empty, which is default), description *desc* and flags *flags* to the command line description.

void wxCmdLineParser::AddUsageText (const wxString & *text*)

Add a string *text* to the command line description shown by Usage().

Since

2.9.0

bool wxCmdLineParser::AreLongOptionsEnabled ( ) const

Returns true if long options are enabled, otherwise false.

See also

EnableLongOptions()

static wxArrayString wxCmdLineParser::ConvertStringToArgs (const wxString & *cmdline*, wxCmdLineSplitType *flags* = wxCMD_LINE_SPLIT_DOS) [static]

Breaks down the string containing the full command line in words.

Words are separated by whitespace and double quotes can be used to preserve the spaces inside the words.

By default, this function uses Windows-like word splitting algorithm, i.e. single quotes have no special meaning and backslash can't be used to escape spaces neither. With wxCMD_LINE_SPLIT_UNIX flag Unix semantics is used, i.e. both single and double quotes can be used and backslash can be used to escape all the other special characters.

void wxCmdLineParser::DisableLongOptions ( )

Identical to EnableLongOptions(false).

void wxCmdLineParser::EnableLongOptions (bool *enable* = true)

Enable or disable support for the long options.

As long options are not (yet) POSIX-compliant, this option allows to disable them.

See also

Customization and AreLongOptionsEnabled()

bool wxCmdLineParser::Found (const wxString & *name*) const

Returns true if the given switch was found, false otherwise.

bool wxCmdLineParser::Found (const wxString & *name*, wxString * *value*) const

Returns true if an option taking a string value was found and stores the value in the provided pointer (which should not be NULL).

bool wxCmdLineParser::Found (const wxString & *name*, long * *value*) const

Returns true if an option taking an integer value was found and stores the value in the provided pointer (which should not be NULL).

bool wxCmdLineParser::Found (const wxString & *name*, double * *value*) const

Returns true if an option taking a float value was found and stores the value in the provided pointer (which should not be NULL).

bool wxCmdLineParser::Found (const wxString & *name*, wxDateTime * *value*) const

Returns true if an option taking a date value was found and stores the value in the provided pointer (which should not be NULL).

wxCmdLineSwitchState wxCmdLineParser::FoundSwitch (const wxString & *name*) const

Returns whether the switch was found on the command line and whether it was negated.

This method can be used for any kind of switch but is especially useful for switches that can be negated, i.e. were added with wxCMD_LINE_SWITCH_NEGATABLE flag, as otherwise Found() is simpler to use.

However Found() doesn't allow to distinguish between switch specified normally, i.e. without dash following it, and negated switch, i.e. with the following dash. This method will return `wxCMD_SWITCH_ON` or `wxCMD_SWITCH_OFF` depending on whether the switch was negated or not. And if the switch was not found at all, `wxCMD_SWITCH_NOT_FOUND` is returned.

Since

2.9.2

wxCmdLineArgs wxCmdLineParser::GetArguments ( ) const

Returns the collection of arguments.

Note

The returned object just refers to the command line parser. The command line parser must live longer than it.

See also

wxCmdLineArgs

Since

3.1.0

wxString wxCmdLineParser::GetParam (size_t *n* = 0) const

Returns the value of Nth parameter (as string only).

size_t wxCmdLineParser::GetParamCount ( ) const

Returns the number of parameters found.

This function makes sense mostly if you had used `wxCMD_LINE_PARAM_MULTIPLE` flag.

wxString wxCmdLineParser::GetUsageString ( ) const

Return the string containing the program usage description.

Call Usage() to directly show this string to the user.

int wxCmdLineParser::Parse (bool *giveUsage* = `true`)

Parse the command line, return 0 if ok, -1 if `"-h"` or `"\--help"` option was encountered and the help message was given or a positive value if a syntax error occurred.

Parameters

giveUsage	If true (default), the usage message is given if a syntax error was encountered while parsing the command line or if help was requested. If false, only error messages about possible syntax errors are given, use Usage to show the usage message from the caller if needed.

void wxCmdLineParser::SetCmdLine (int *argc*, char ** *argv*)

Set the command line to parse after using one of the constructors which don't do it.

void wxCmdLineParser::SetCmdLine (int *argc*, wchar_t ** *argv*)

Set the command line to parse after using one of the constructors which don't do it.

void wxCmdLineParser::SetCmdLine (const wxString & *cmdline*)

Set the command line to parse after using one of the constructors which don't do it.

void wxCmdLineParser::SetDesc (const wxCmdLineEntryDesc * *desc*)

Constructs the command line description.

Take the command line description from the wxCMD_LINE_NONE terminated table.

Example of usage:

```
static const wxCmdLineEntryDesc cmdLineDesc[] =
{
    { wxCMD_LINE_SWITCH, "v", "verbose", "be verbose" },
    { wxCMD_LINE_SWITCH, "q", "quiet",   "be quiet" },

    { wxCMD_LINE_OPTION, "o", "output",  "output file" },
    { wxCMD_LINE_OPTION, "i", "input",   "input dir" },
    { wxCMD_LINE_OPTION, "s", "size",    "output block size",
      wxCMD_LINE_VAL_NUMBER },
    { wxCMD_LINE_OPTION, "d", "date",    "output file date",
      wxCMD_LINE_VAL_DATE },
```

```
    { wxCMD_LINE_PARAM,  NULL, NULL, "input file",
      wxCMD_LINE_VAL_STRING, wxCMD_LINE_PARAM_MULTIPLE },

    { wxCMD_LINE_NONE }
};

wxCmdLineParser parser;

parser.SetDesc(cmdLineDesc);
```

void wxCmdLineParser::SetLogo (const wxString & *logo*)

The *logo* is some extra text which will be shown by Usage() method.

void wxCmdLineParser::SetSwitchChars (const wxString & *switchChars*)

switchChars contains all characters with which an option or switch may start.

Default is "-" for Unix, "-/" for Windows.

void wxCmdLineParser::Usage ( ) const

Give the standard usage message describing all program options.

It will use the options and parameters descriptions specified earlier, so the resulting message will not be helpful to the user unless the descriptions were indeed specified.

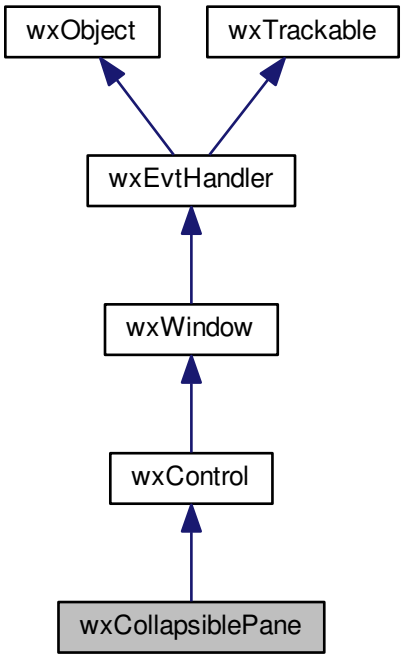
See also

SetLogo()

21.100 wxCollapsiblePane Class Reference

```
#include <wx/collpane.h>
```

Inheritance diagram for wxCollapsiblePane:



21.100.1 Detailed Description

A collapsible pane is a container with an embedded button-like control which can be used by the user to collapse or expand the pane's contents.

Once constructed you should use the GetPane() function to access the pane and add your controls inside it (i.e. use the returned pointer from GetPane() as parent for the controls which must go in the pane, **not** the wxCollapsiblePane itself!).

Note that because of its nature of control which can dynamically (and drastically) change its size at run-time under user-input, when putting wxCollapsiblePane inside a wxSizer you should be careful to add it with a proportion value of zero; this is because otherwise all other windows with non-null proportion values will automatically resize each time the user expands or collapse the pane window usually resulting in a weird, flickering effect.

Usage sample:

```
wxCollapsiblePane *collpane = new wxCollapsiblePane(this,
      wxID_ANY, "Details:");

// add the pane with a zero proportion value to the 'sz' sizer which contains it
sz->Add(collpane, 0, wxGROW|wxALL, 5);

// now add a test label in the collapsible pane using a sizer to layout it:
wxWindow *win = collpane->GetPane();
wxSizer *paneSz = new wxBoxSizer(wxVERTICAL);
paneSz->Add(new wxStaticText(win, wxID_ANY, "test!"), 1,
      wxGROW|wxALL, 2);
win->SetSizer(paneSz);
paneSz->SetSizeHints(win);
```

It is only available if `wxUSE_COLLPANE` is set to 1 (the default).

Styles

This class supports the following styles:

- wxCP_DEFAULT_STYLE: The default style. It includes wxTAB_TRAVERSAL and wxBORDER_NONE.
- wxCP_NO_TLW_RESIZE: By default wxCollapsiblePane resizes the top level window containing it when its own size changes. This allows to easily implement dialogs containing an optionally shown part, for example, and so is the default behaviour but can be inconvenient in some specific cases – use this flag to disable this automatic parent resizing then.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCollapsiblePaneEvent& event) or void handlerFuncName(wxNavigationKeyEvent& event)

Event macros for events emitted by this class:

- EVT_COLLAPSIBLEPANE_CHANGED(id, func): The user expanded or collapsed the collapsible pane.
- EVT_NAVIGATION_KEY(func): Process a navigation key event.

Library: wxCore

Category: Controls

See also

wxPanel, wxCollapsiblePaneEvent

Public Member Functions

- wxCollapsiblePane ()

  Default constructor.
- wxCollapsiblePane (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxCollapsiblePaneNameStr)

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxCollapsiblePaneNameStr)
- virtual void Collapse (bool collapse=true)

  Collapses or expands the pane window.
- void Expand ()

  Same as calling Collapse(false).
- virtual wxWindow * GetPane () const

  Returns a pointer to the pane window.
- virtual bool IsCollapsed () const

  Returns true if the pane window is currently hidden.
- bool IsExpanded () const

  Returns true if the pane window is currently shown.

Additional Inherited Members

21.100.2 Constructor & Destructor Documentation

wxCollapsiblePane::wxCollapsiblePane ( )

Default constructor.

wxCollapsiblePane::wxCollapsiblePane (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxCollapsiblePaneNameStr`)

Initializes the object and calls Create() with all the parameters.

21.100.3 Member Function Documentation

virtual void wxCollapsiblePane::Collapse (bool *collapse* = `true`) `[virtual]`

Collapses or expands the pane window.

bool wxCollapsiblePane::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxCollapsiblePaneNameStr`)

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
label	The initial label shown in the button which allows the user to expand or collapse the pane window.
pos	Initial position.
size	Initial size.
style	The window style, see wxCP_* flags.
validator	Validator which can be used for additional date checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

void wxCollapsiblePane::Expand ( )

Same as calling Collapse(false).

virtual wxWindow* wxCollapsiblePane::GetPane ( ) const `[virtual]`

Returns a pointer to the pane window.

Add controls to the returned wxWindow to make them collapsible.

virtual bool wxCollapsiblePane::IsCollapsed ( ) const `[virtual]`

Returns true if the pane window is currently hidden.

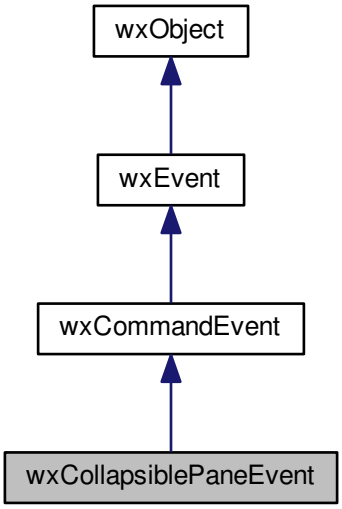
bool wxCollapsiblePane::IsExpanded ( ) const

Returns true if the pane window is currently shown.

21.101 wxCollapsiblePaneEvent Class Reference

`#include <wx/collpane.h>`

Inheritance diagram for wxCollapsiblePaneEvent:



21.101.1 Detailed Description

This event class is used for the events generated by wxCollapsiblePane.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCollapsiblePaneEvent& event)

Event macros:

- EVT_COLLAPSIBLEPANE_CHANGED(id, func): The user expanded or collapsed the collapsible pane.

Library: wxCore

Category: Events

See also

wxCollapsiblePane

Public Member Functions

- wxCollapsiblePaneEvent (wxObject *generator, int id, bool collapsed)

  The constructor is not normally used by the user code.
- bool GetCollapsed () const

  Returns true if the pane has been collapsed.
- void SetCollapsed (bool collapsed)

  Sets this as a collapsed pane event (if collapsed is true) or as an expanded pane event (if collapsed is false).

Additional Inherited Members

21.101.2 Constructor & Destructor Documentation

wxCollapsiblePaneEvent::wxCollapsiblePaneEvent (wxObject * *generator,* int *id,* bool *collapsed*)

The constructor is not normally used by the user code.

21.101.3 Member Function Documentation

bool wxCollapsiblePaneEvent::GetCollapsed ( ) const

Returns true if the pane has been collapsed.

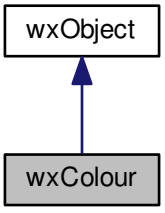
void wxCollapsiblePaneEvent::SetCollapsed (bool *collapsed*)

Sets this as a collapsed pane event (if *collapsed* is true) or as an expanded pane event (if *collapsed* is false).

21.102 wxColour Class Reference

`#include <wx/colour.h>`

Inheritance diagram for wxColour:



21.102.1 Detailed Description

A colour is an object representing a combination of Red, Green, and Blue (RGB) intensity values, and is used to determine drawing colours.

See the entry for wxColourDatabase for how a pointer to a predefined, named colour may be returned instead of creating a new colour.

Valid RGB values are in the range 0 to 255.

You can retrieve the current system colour settings with wxSystemSettings.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers:

- wxNullColour - An empty, invalid colour.
- wxTransparentColour - Valid but fully transparent colour (new in 2.9.1).
- wxBLACK
- wxBLUE
- wxCYAN
- wxGREEN
- wxYELLOW
- wxLIGHT_GREY
- wxRED
- wxWHITE

See also

wxColourDatabase, wxPen, wxBrush, wxColourDialog, wxSystemSettings

Public Member Functions

- wxColour ()

  Default constructor.
- wxColour (unsigned char red, unsigned char green, unsigned char blue, unsigned char alpha=wxALPHA_OPAQUE)
- wxColour (const wxString &colourName)
- wxColour (unsigned long colRGB)
- wxColour (const wxColour &colour)

  Copy constructor.
- virtual unsigned char Alpha () const

  Returns the alpha value, on platforms where alpha is not yet supported, this always returns wxALPHA_OPAQUE.
- virtual unsigned char Blue () const

  Returns the blue intensity.
- virtual wxString GetAsString (long flags=wxC2S_NAME|wxC2S_CSS_SYNTAX) const

  Converts this colour to a wxString using the given flags.
- wxIntPtr GetPixel () const

  Returns a pixel value which is platform-dependent.
- virtual unsigned char Green () const

  Returns the green intensity.
- virtual bool IsOk () const

Returns true if the colour object is valid (the colour has been initialised with RGB values).

- virtual unsigned char Red () const

  Returns the red intensity.
- bool operator!= (const wxColour &colour) const

  Tests the inequality of two colours by comparing individual red, green, blue colours and alpha values.
- wxColour & operator= (const wxColour &colour)

  Assignment operator, using a colour name to be found in the colour database.
- bool operator== (const wxColour &colour) const

  Tests the equality of two colours by comparing individual red, green, blue colours and alpha values.
- wxColour & MakeDisabled (unsigned char brightness=255)

  Make a disabled version of this colour.
- wxColour ChangeLightness (int ialpha) const

  wxColour wrapper for ChangeLightness(r,g,b,ialpha).

- void SetRGB (wxUint32 colRGB)

  Sets the RGB or RGBA colour values from a single 32 bit value.
- void SetRGBA (wxUint32 colRGBA)

  Sets the RGB or RGBA colour values from a single 32 bit value.

- wxUint32 GetRGB () const

  Gets the RGB or RGBA colour values as a single 32 bit value.
- wxUint32 GetRGBA () const

  Gets the RGB or RGBA colour values as a single 32 bit value.

- void Set (unsigned char red, unsigned char green, unsigned char blue, unsigned char alpha=wxALPHA_OPAQUE)

  Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).
- void Set (unsigned long RGB)

  Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).
- bool Set (const wxString &str)

  Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).

Static Public Member Functions

- static void MakeMono (unsigned char *r, unsigned char *g, unsigned char *b, bool on)

  Assign 0 or 255 to rgb out parameters.
- static void MakeDisabled (unsigned char *r, unsigned char *g, unsigned char *b, unsigned char brightness=255)

  Create a disabled (dimmed) colour from (in/out) rgb parameters.
- static void MakeGrey (unsigned char *r, unsigned char *g, unsigned char *b)

  Create a grey colour from (in/out) rgb parameters using integer arithmetic.
- static void MakeGrey (unsigned char *r, unsigned char *g, unsigned char *b, double weight_r, double weight_g, double weight_b)

  Create a grey colour from (in/out) rgb parameters using floating point arithmetic.
- static unsigned char AlphaBlend (unsigned char fg, unsigned char bg, double alpha)

  Blend colour, taking alpha into account.
- static void ChangeLightness (unsigned char *r, unsigned char *g, unsigned char *b, int ialpha)

  ChangeLightness() is a utility function that simply darkens or lightens a color, based on the specified percentage ialpha of 0 would be completely black, 200 completely white an ialpha of 100 returns the same colour.

Additional Inherited Members

21.102.2 Constructor & Destructor Documentation

wxColour::wxColour ( )

Default constructor.

wxColour::wxColour (unsigned char *red,* unsigned char *green,* unsigned char *blue,* unsigned char *alpha =* wxALPHA_OPAQUE)

Parameters

red	The red value.
green	The green value.
blue	The blue value.
alpha	The alpha value. Alpha values range from 0 (wxALPHA_TRANSPARENT) to 255 (wxALP↩ HA_OPAQUE).

wxColour::wxColour (const wxString & *colourName*)

Parameters

colourName	The colour name.

wxColour::wxColour (unsigned long *colRGB*)

Parameters

colRGB	A packed RGB value.

wxColour::wxColour (const wxColour & *colour*)

Copy constructor.

21.102.3 Member Function Documentation

virtual unsigned char wxColour::Alpha ( ) const `[virtual]`

Returns the alpha value, on platforms where alpha is not yet supported, this always returns wxALPHA_OPAQUE.

static unsigned char wxColour::AlphaBlend (unsigned char *fg,* unsigned char *bg,* double *alpha*) `[static]`

Blend colour, taking alpha into account.

Since

2.9.0

virtual unsigned char wxColour::Blue ( ) const `[virtual]`

Returns the blue intensity.

static void wxColour::ChangeLightness (unsigned char * *r*, unsigned char * *g*, unsigned char * *b*, int *ialpha*) [static]

ChangeLightness() is a utility function that simply darkens or lightens a color, based on the specified percentage ialpha of 0 would be completely black, 200 completely white an ialpha of 100 returns the same colour.

Since

2.9.0

wxColour wxColour::ChangeLightness (int *ialpha*) const

wxColour wrapper for ChangeLightness(r,g,b,ialpha).

Since

2.9.0

virtual wxString wxColour::GetAsString (long *flags* = wxC2S_NAME|wxC2S_CSS_SYNTAX) const [virtual]

Converts this colour to a wxString using the given flags.

The supported flags are `wxC2S_NAME`, to obtain the colour name (e.g. wxColour(255,0,0) == "red"), `wxC2S_CSS_SYNTAX`, to obtain the colour in the "rgb(r,g,b)" or "rgba(r,g,b,a)" syntax (e.g. wxColour(255,0,0,85) == "rgba(255,0,0,0.333)"), and `wxC2S_HTML_SYNTAX`, to obtain the colour as "#" followed by 6 hexadecimal digits (e.g. wxColour(255,0,0) == "#FF0000").

This function never fails and always returns a non-empty string but asserts if the colour has alpha channel (i.e. is non opaque) but `wxC2S_CSS_SYNTAX` (which is the only one supporting alpha) is not specified in flags.

Since

2.7.0

wxIntPtr wxColour::GetPixel ( ) const

Returns a pixel value which is platform-dependent.

On Windows, a COLORREF is returned. On X, an allocated pixel value is returned. If the pixel is invalid (on X, unallocated), `-1` is returned.

wxUint32 wxColour::GetRGB ( ) const

Gets the RGB or RGBA colour values as a single 32 bit value.

The returned value is of the same form as expected by SetRGB() and SetRGBA().

Notice that GetRGB() returns the value with 0 as its highest byte independently of the value actually returned by Alpha(). So for a fully opaque colour, the return value of GetRGBA() is `0xFFBBGGRR` while that of GetRGB() is `0x00BBGGRR`.

Since

2.9.1

wxUint32 wxColour::GetRGBA ( ) const

Gets the RGB or RGBA colour values as a single 32 bit value.

The returned value is of the same form as expected by SetRGB() and SetRGBA().

Notice that GetRGB() returns the value with 0 as its highest byte independently of the value actually returned by Alpha(). So for a fully opaque colour, the return value of GetRGBA() is `0xFFBBGGRR` while that of GetRGB() is `0x00BBGGRR`.

Since

2.9.1

virtual unsigned char wxColour::Green ( ) const `[virtual]`

Returns the green intensity.

virtual bool wxColour::IsOk ( ) const `[virtual]`

Returns true if the colour object is valid (the colour has been initialised with RGB values).

static void wxColour::MakeDisabled (unsigned char * *r*, unsigned char * *g*, unsigned char * *b*, unsigned char *brightness* = `255`) `[static]`

Create a disabled (dimmed) colour from (in/out) rgb parameters.

Since

2.9.0

wxColour& wxColour::MakeDisabled (unsigned char *brightness* = `255`)

Make a disabled version of this colour.

This method modifies the object in place and returns the object itself.

Since

2.9.5

static void wxColour::MakeGrey (unsigned char * *r*, unsigned char * *g*, unsigned char * *b*) `[static]`

Create a grey colour from (in/out) rgb parameters using integer arithmetic.

Since

2.9.0

static void wxColour::MakeGrey (unsigned char * *r*, unsigned char * *g*, unsigned char * *b*, double *weight_r*, double *weight_g*, double *weight_b*) `[static]`

Create a grey colour from (in/out) rgb parameters using floating point arithmetic.

Defaults to using the standard ITU-T BT.601 when converting to YUV, where every pixel equals (R * *weight_r*) + (G * *weight_g*) + (B * *weight_b*).

Since

2.9.0

static void wxColour::MakeMono (unsigned char * *r,* unsigned char * *g,* unsigned char * *b,* bool *on*) [static]

Assign 0 or 255 to rgb out parameters.

Since

2.9.0

bool wxColour::operator!= (const wxColour & *colour*) const

Tests the inequality of two colours by comparing individual red, green, blue colours and alpha values.

wxColour& wxColour::operator= (const wxColour & *colour*)

Assignment operator, using a colour name to be found in the colour database.

See also

wxColourDatabase

bool wxColour::operator== (const wxColour & *colour*) const

Tests the equality of two colours by comparing individual red, green, blue colours and alpha values.

virtual unsigned char wxColour::Red ( ) const [virtual]

Returns the red intensity.

void wxColour::Set (unsigned char *red,* unsigned char *green,* unsigned char *blue,* unsigned char *alpha* = wxALPHA_OPAQUE)

Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).

When using third form, Set() accepts: colour names (those listed in wxColourDatabase), the CSS-like "rgb(r,g,b)" or "rgba(r,g,b,a)" syntax (case insensitive) and the HTML-like syntax: "#" followed by 6 hexadecimal digits for red, green, blue components.

Returns true if the conversion was successful, false otherwise.

Since

2.7.0

void wxColour::Set (unsigned long *RGB*)

Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).

When using third form, Set() accepts: colour names (those listed in wxColourDatabase), the CSS-like "`rgb(r,g,b)`" or "`rgba(r,g,b,a)`" syntax (case insensitive) and the HTML-like syntax: "`#`" followed by 6 hexadecimal digits for red, green, blue components.

Returns true if the conversion was successful, false otherwise.

Since

2.7.0

bool wxColour::Set (const wxString & *str*)

Sets the RGB intensity values using the given values (first overload), extracting them from the packed long (second overload), using the given string (third overload).

When using third form, Set() accepts: colour names (those listed in wxColourDatabase), the CSS-like "`rgb(r,g,b)`" or "`rgba(r,g,b,a)`" syntax (case insensitive) and the HTML-like syntax: "`#`" followed by 6 hexadecimal digits for red, green, blue components.

Returns true if the conversion was successful, false otherwise.

Since

2.7.0

void wxColour::SetRGB (wxUint32 *colRGB*)

Sets the RGB or RGBA colour values from a single 32 bit value.

The arguments *colRGB* and *colRGBA* should be of the form 0x00BBGGRR and 0xAABBGGRR respectively where `0xRR`, `0xGG`, `0xBB` and `0xAA` are the values of the red, blue, green and alpha components.

Notice the right-to-left order of components!

See also

GetRGB(), GetRGBA()

Since

2.9.1

void wxColour::SetRGBA (wxUint32 *colRGBA*)

Sets the RGB or RGBA colour values from a single 32 bit value.

The arguments *colRGB* and *colRGBA* should be of the form 0x00BBGGRR and 0xAABBGGRR respectively where `0xRR`, `0xGG`, `0xBB` and `0xAA` are the values of the red, blue, green and alpha components.

Notice the right-to-left order of components!

See also

GetRGB(), GetRGBA()

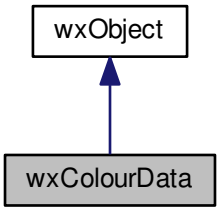
Since

2.9.1

21.103 wxColourData Class Reference

`#include <wx/colourdata.h>`

Inheritance diagram for wxColourData:



21.103.1 Detailed Description

This class holds a variety of information related to colour dialogs.

Library: wxCore

Category: Common Dialogs, Data Structures

See also

wxColour, wxColourDialog, wxColourDialog Overview

Public Types

- enum { NUM_CUSTOM = 16 }

  number of custom colours we store

Public Member Functions

- wxColourData ()

  Constructor.
- virtual ~wxColourData ()

  Destructor.
- bool GetChooseFull () const

  Under Windows, determines whether the Windows colour dialog will display the full dialog with custom colour selection controls.
- wxColour & GetColour ()

Gets the current colour associated with the colour dialog.

- wxColour GetCustomColour (int i) const

  Returns custom colours associated with the colour dialog.

- void SetChooseFull (bool flag)

  Under Windows, tells the Windows colour dialog to display the full dialog with custom colour selection controls.

- void SetColour (const wxColour &colour)

  Sets the default colour for the colour dialog.

- void SetCustomColour (int i, const wxColour &colour)

  Sets custom colours for the colour dialog.

- wxString ToString () const

  Converts the colours saved in this class in a string form, separating the various colours with a comma.

- bool FromString (const wxString &str)

  Decodes the given string, which should be in the same format returned by ToString(), and sets the internal colours.

- wxColourData & operator= (const wxColourData &data)

  Assignment operator for the colour data.

Additional Inherited Members

21.103.2 Member Enumeration Documentation

anonymous enum

number of custom colours we store

Enumerator

NUM_CUSTOM

21.103.3 Constructor & Destructor Documentation

wxColourData::wxColourData ( )

Constructor.

Initializes the custom colours to `wxNullColour`, the *data* colour setting to black, and the *choose* full setting to true.

virtual wxColourData::~wxColourData ( ) [virtual]

Destructor.

21.103.4 Member Function Documentation

bool wxColourData::FromString (const wxString & *str*)

Decodes the given string, which should be in the same format returned by ToString(), and sets the internal colours.

bool wxColourData::GetChooseFull ( ) const

Under Windows, determines whether the Windows colour dialog will display the full dialog with custom colour selection controls.

Has no meaning under other platforms.

The default value is true.

wxColour& wxColourData::GetColour ( )

Gets the current colour associated with the colour dialog.

The default colour is black.

wxColour wxColourData::GetCustomColour (int *i*) const

Returns custom colours associated with the colour dialog.

Parameters

i	An integer between 0 and 15, being any of the 15 custom colours that the user has saved. The default custom colours are invalid colours.

wxColourData& wxColourData::operator= (const wxColourData & *data*)

Assignment operator for the colour data.

void wxColourData::SetChooseFull (bool *flag*)

Under Windows, tells the Windows colour dialog to display the full dialog with custom colour selection controls.

Under other platforms, has no effect.

The default value is true.

void wxColourData::SetColour (const wxColour & *colour*)

Sets the default colour for the colour dialog.

The default colour is black.

void wxColourData::SetCustomColour (int *i,* const wxColour & *colour*)

Sets custom colours for the colour dialog.

Parameters

i	An integer between 0 and 15 for whatever custom colour you want to set. The default custom colours are invalid colours.
colour	The colour to set

wxString wxColourData::ToString ( ) const

Converts the colours saved in this class in a string form, separating the various colours with a comma.

21.104 wxColourDatabase Class Reference

```
#include <wx/gdicmn.h>
```

21.104.1 Detailed Description

wxWidgets maintains a database of standard RGB colours for a predefined set of named colours.

The application may add to this set if desired by using AddColour() and may use it to look up colours by names using Find() or find the names for the standard colour using FindName().

There is one predefined, global instance of this class called wxTheColourDatabase.

The standard database contains at least the following colours:

AQUAMARINE	FIREBRICK	MEDIUM FOREST	RED
BLACK	FOREST GREEN	GREEN	SALMON
BLUE	GOLD	MEDIUM GOLDENROD	SEA GREEN
BLUE VIOLET	GOLDENROD	MEDIUM ORCHID	SIENNA
BROWN	GREY	MEDIUM SEA GREEN	SKY BLUE
CADET BLUE	GREEN	MEDIUM SLATE BLUE	SLATE BLUE
CORAL	GREEN YELLOW	MEDIUM SPRING	SPRING GREEN
CORNFLOWER BLUE	INDIAN RED	GREEN	STEEL BLUE
CYAN	KHAKI	MEDIUM TURQUOISE	TAN
DARK GREY	LIGHT BLUE	MEDIUM VIOLET RED	THISTLE
DARK GREEN	LIGHT GREY	MIDNIGHT BLUE	TURQUOISE
DARK OLIVE GREEN	LIGHT STEEL BLUE	NAVY	VIOLET
DARK ORCHID	LIME GREEN	ORANGE	VIOLET RED
DARK SLATE BLUE	MAGENTA	ORANGE RED	WHEAT
DARK SLATE GREY	MAROON	ORCHID	WHITE
DARK TURQUOISE	MEDIUM AQUAMARINE	PALE GREEN	YELLOW
DIM GREY		PINK	YELLOW GREEN
	MEDIUM BLUE	PLUM	
		PURPLE	

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxColour

Public Member Functions

- wxColourDatabase ()

  Constructs the colour database.
- void AddColour (const wxString &colourName, const wxColour &colour)

  Adds a colour to the database.
- wxColour Find (const wxString &colourName) const

  Finds a colour given the name.
- wxString FindName (const wxColour &colour) const

  Finds a colour name given the colour.

21.104.2 Constructor & Destructor Documentation

wxColourDatabase::wxColourDatabase ( )

Constructs the colour database.

It will be initialized at the first use.

21.104.3 Member Function Documentation

void wxColourDatabase::AddColour (const wxString & *colourName,* const wxColour & *colour*)

Adds a colour to the database.

If a colour with the same name already exists, it is replaced.

wxColour wxColourDatabase::Find (const wxString & *colourName*) const

Finds a colour given the name.

Returns an invalid colour object (that is, wxColour::IsOk() will return false) if the colour wasn't found in the database.

wxString wxColourDatabase::FindName (const wxColour & *colour*) const

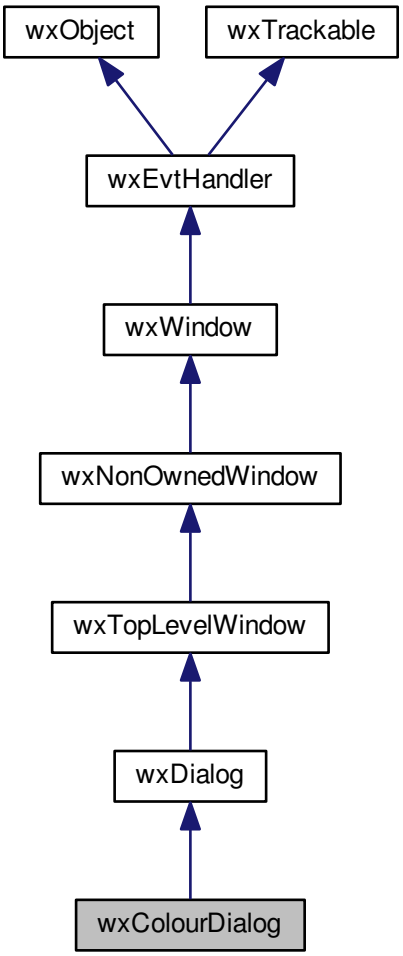
Finds a colour name given the colour.

Returns an empty string if the colour is not found in the database.

21.105 wxColourDialog Class Reference

```
#include <wx/colordlg.h>
```

Inheritance diagram for wxColourDialog:



21.105.1 Detailed Description

This class represents the colour chooser dialog.

Library: wxCore

Category: Common Dialogs

See also

wxColourDialog Overview, wxColour, wxColourData, wxGetColourFromUser()

Public Member Functions

- wxColourDialog (wxWindow *parent, wxColourData *data=NULL)

Constructor.

- virtual ~wxColourDialog ()

  Destructor.

- bool Create (wxWindow *parent, wxColourData *data=NULL)

  Same as wxColourDialog().

- wxColourData & GetColourData ()

  Returns the colour data associated with the colour dialog.

- virtual int ShowModal ()

  Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

Additional Inherited Members

21.105.2 Constructor & Destructor Documentation

wxColourDialog::wxColourDialog (wxWindow * *parent,* wxColourData * *data* = NULL)

Constructor.

Pass a parent window, and optionally a pointer to a block of colour data, which will be copied to the colour dialog's colour data.

Custom colours from colour data object will be used in the dialog's colour palette. Invalid entries in custom colours list will be ignored on some platforms(GTK) or replaced with white colour on platforms where custom colours palette has fixed size (MSW).

See also

wxColourData

virtual wxColourDialog::~wxColourDialog ( ) [virtual]

Destructor.

21.105.3 Member Function Documentation

bool wxColourDialog::Create (wxWindow * *parent,* wxColourData * *data* = NULL)

Same as wxColourDialog().

wxColourData& wxColourDialog::GetColourData ( )

Returns the colour data associated with the colour dialog.

virtual int wxColourDialog::ShowModal ( ) [virtual]

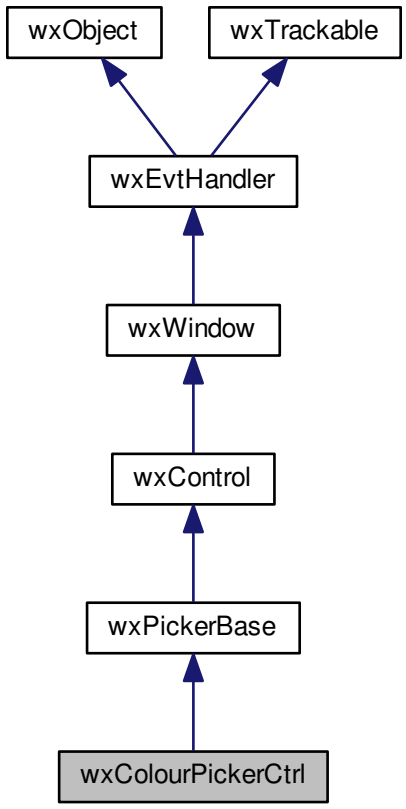
Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

Reimplemented from wxDialog.

21.106 wxColourPickerCtrl Class Reference

`#include <wx/clrpicker.h>`

Inheritance diagram for wxColourPickerCtrl:



21.106.1 Detailed Description

This control allows the user to select a colour.

The generic implementation is a button which brings up a wxColourDialog when clicked. Native implementation may differ but this is usually a (small) widget which give access to the colour-chooser dialog. It is only available if `wxUSE_COLOURPICKERCTRL` is set to 1 (the default).

Styles

This class supports the following styles:

- wxCLRP_DEFAULT_STYLE: The default style: 0.
- wxCLRP_USE_TEXTCTRL: Creates a text control to the left of the picker button which is completely managed by the wxColourPickerCtrl and which can be used by the user to specify a colour (see SetColour). The text control is automatically synchronized with button's value. Use functions defined in wxPickerBase to modify the text control.

- wxCLRP_SHOW_LABEL: Shows the colour in HTML form (AABBCC) as colour button label (instead of no label at all).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxColourPickerEvent& event)

Event macros for events emitted by this class:

- EVT_COLOURPICKER_CHANGED(id, func): The user changed the colour selected in the control either using the button or using text control (see `wxCLRP_USE_TEXTCTRL`; note that in this case the event is fired only if the user's input is valid, i.e. recognizable).

Library: wxCore

Category: Picker Controls

See also

wxColourDialog, wxColourPickerEvent

Public Member Functions

- wxColourPickerCtrl ()
- wxColourPickerCtrl (wxWindow *parent, wxWindowID id, const wxColour &colour=*wxBLACK, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCLRP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxColourPickerCtrlNameStr)

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxColour &colour=*wxBLACK, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCLRP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxColourPickerCtrlNameStr)

  Creates a colour picker with the given arguments.
- wxColour GetColour () const

  Returns the currently selected colour.
- void SetColour (const wxColour &col)

  Sets the currently selected colour.
- void SetColour (const wxString &colname)

  Sets the currently selected colour.

Additional Inherited Members

21.106.2 Constructor & Destructor Documentation

wxColourPickerCtrl::wxColourPickerCtrl ( )

wxColourPickerCtrl::wxColourPickerCtrl (wxWindow * *parent*, wxWindowID *id*, const wxColour & *colour* = *wxBLACK, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCLRP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxColourPickerCtrlNameStr`)

Initializes the object and calls Create() with all the parameters.

21.106.3 Member Function Documentation

bool wxColourPickerCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxColour & *colour* = *wxBLACK, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCLRP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxColourPickerCtrlNameStr)

Creates a colour picker with the given arguments.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
colour	The initial colour shown in the control.
pos	Initial position.
size	Initial size.
style	The window style, see wxCRLP_* flags.
validator	Validator which can be used for additional date checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

wxColour wxColourPickerCtrl::GetColour ( ) const

Returns the currently selected colour.

void wxColourPickerCtrl::SetColour (const wxColour & *col*)

Sets the currently selected colour.

See wxColour::Set().

void wxColourPickerCtrl::SetColour (const wxString & *colname*)

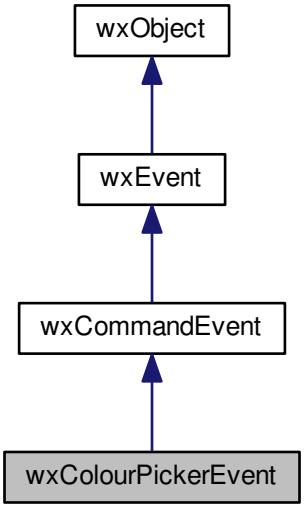
Sets the currently selected colour.

See wxColour::Set().

21.107 wxColourPickerEvent Class Reference

`#include <wx/clrpicker.h>`

Inheritance diagram for wxColourPickerEvent:



21.107.1 Detailed Description

This event class is used for the events generated by wxColourPickerCtrl.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxColourPickerEvent& event)

Event macros:

- EVT_COLOURPICKER_CHANGED(id, func): Generated whenever the selected colour changes.

Library: wxCore

Category: Events

See also

wxColourPickerCtrl

Public Member Functions

- wxColourPickerEvent ()
- wxColourPickerEvent (wxObject *generator, int id, const wxColour &colour)

  The constructor is not normally used by the user code.
- wxColour GetColour () const

Retrieve the colour the user has just selected.

- void SetColour (const wxColour &pos)

Set the colour associated with the event.

Additional Inherited Members

21.107.2 Constructor & Destructor Documentation

wxColourPickerEvent::wxColourPickerEvent ( )

wxColourPickerEvent::wxColourPickerEvent (wxObject * *generator,* int *id,* const wxColour & *colour*)

The constructor is not normally used by the user code.

21.107.3 Member Function Documentation

wxColour wxColourPickerEvent::GetColour ( ) const

Retrieve the colour the user has just selected.

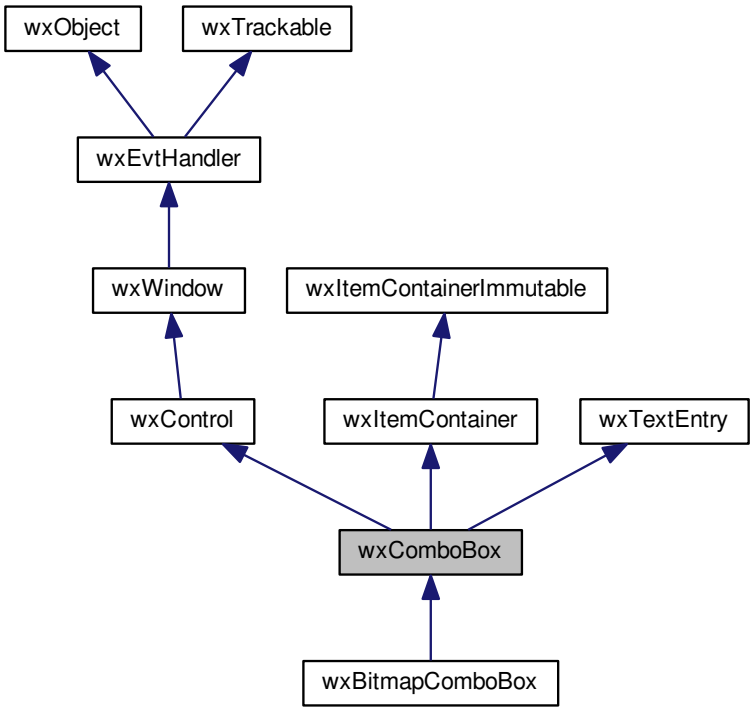
void wxColourPickerEvent::SetColour (const wxColour & *pos*)

Set the colour associated with the event.

21.108 wxComboBox Class Reference

`#include <wx/combobox.h>`

Inheritance diagram for wxComboBox:



21.108.1 Detailed Description

A combobox is like a combination of an edit control and a listbox.

It can be displayed as static list with editable or read-only text field; or a drop-down list with text field; or a drop-down list without a text field depending on the platform and presence of wxCB_READONLY style.

A combobox permits a single selection only. Combobox items are numbered from zero.

If you need a customized combobox, have a look at wxComboCtrl, wxOwnerDrawnComboBox, wxComboPopup and the ready-to-use wxBitmapComboBox.

Please refer to wxTextEntry documentation for the description of methods operating with the text entry part of the combobox and to wxItemContainer for the methods operating with the list of strings. Notice that at least under MSW wxComboBox doesn't behave correctly if it contains strings differing in case only so portable programs should avoid adding such strings to this control.

Styles

This class supports the following styles:

- wxCB_SIMPLE: Creates a combobox with a permanently displayed list. Windows only.
- wxCB_DROPDOWN: Creates a combobox with a drop-down list. MSW and Motif only.
- wxCB_READONLY: A combobox with this style behaves like a wxChoice (and may look in the same way as well, although this is platform-dependent), i.e. it allows the user to choose from the list of options but doesn't allow to enter a value not present in the list.

- wxCB_SORT: Sorts the entries in the list alphabetically.
- wxTE_PROCESS_ENTER: The control will generate the event `wxEVT_TEXT_ENTER` (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_COMBOBOX(id, func): Process a `wxEVT_COMBOBOX` event, when an item on the list is selected. Note that calling GetValue() returns the new value of selection.
- EVT_TEXT(id, func): Process a `wxEVT_TEXT` event, when the combobox text changes.
- EVT_TEXT_ENTER(id, func): Process a `wxEVT_TEXT_ENTER` event, when RETURN is pressed in the combobox (notice that the combobox must have been created with wxTE_PROCESS_ENTER style to receive this event).
- EVT_COMBOBOX_DROPDOWN(id, func): Process a `wxEVT_COMBOBOX_DROPDOWN` event, which is generated when the list box part of the combo box is shown (drops down). Notice that this event is only supported by wxMSW, wxGTK with GTK+ 2.10 or later, and wxOSX/Cocoa.
- EVT_COMBOBOX_CLOSEUP(id, func): Process a `wxEVT_COMBOBOX_CLOSEUP` event, which is generated when the list box of the combo box disappears (closes up). This event is only generated for the same platforms as `wxEVT_COMBOBOX_DROPDOWN` above. Also note that only wxMSW and wxOSX/Cocoa support adding or deleting items in this event.

Library: wxCore

Category: Controls

See also

wxListBox, wxTextCtrl, wxChoice, wxCommandEvent

Public Member Functions

- wxComboBox ()

  Default constructor.
- wxComboBox (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Constructor, creating and showing a combobox.
- wxComboBox (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Constructor, creating and showing a combobox.
- virtual ~wxComboBox ()

  Destructor, destroying the combobox.
- virtual int GetCurrentSelection () const

  Returns the item being selected right now.
- virtual long GetInsertionPoint () const

Same as wxTextEntry::GetInsertionPoint().

- bool IsEmpty () const

  IsEmpty() is not available in this class.

- bool IsListEmpty () const

  Returns true if the list of combobox choices is empty.

- bool IsTextEmpty () const

  Returns true if the text of the combobox is empty.

- virtual void SetSelection (long from, long to)

  Same as wxTextEntry::SetSelection().

- virtual void SetValue (const wxString &text)

  Sets the text for the combobox text field.

- virtual void Popup ()

  Shows the list box portion of the combo box.

- virtual void Dismiss ()

  Hides the list box portion of the combo box.

- virtual int GetSelection () const

  Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

- virtual void GetSelection (long *from, long *to) const

  Gets the current selection span.

- virtual void SetSelection (int n)

  Sets the selection to the given item n or removes the selection entirely if n == `wxNOT_FOUND`.

- virtual int FindString (const wxString &s, bool bCase=false) const

  Finds an item whose label matches the given string.

- virtual wxString GetString (unsigned int n) const

  Returns the label of the item with the given index.

- virtual wxString GetStringSelection () const

  Gets the text currently selected in the control.

- virtual void SetString (unsigned int n, const wxString &text)

  Changes the text of the specified combobox item.

- virtual unsigned int GetCount () const

  Returns the number of items in the control.

- bool Create (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=(const wxString *) NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Creates the combobox for two-step construction.

- bool Create (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Creates the combobox for two-step construction.

Additional Inherited Members

21.108.2 Constructor & Destructor Documentation

wxComboBox::wxComboBox ( )

Default constructor.

wxComboBox::wxComboBox (wxWindow ∗ ***parent*****, wxWindowID** ***id*****,** const **wxString** & ***value*** **= wxEmptyString**, const **wxPoint** & ***pos*** **= wxDefaultPosition**, const **wxSize** & ***size*** **= wxDefaultSize**, int ***n*** **=** `0`, const **wxString** ***choices[]*** **=** `NULL`, long ***style*** **=** `0`, const **wxValidator** & ***validator*** **= wxDefaultValidator**, const **wxString** & ***name*** **=** `wxComboBoxNameStr` **)**

Constructor, creating and showing a combobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial selection string. An empty string indicates no selection. Notice that for the controls with `wxCB_READONLY` style this string must be one of the valid choices if it is not empty.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
n	Number of strings with which to initialise the control.
choices	An array of strings with which to initialise the control.
style	Window style. See wxComboBox.
validator	Window validator.
name	Window name.

wxPerl Note: Not supported by wxPerl.

See also

Create(), wxValidator

wxComboBox::wxComboBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxComboBoxNameStr`)

Constructor, creating and showing a combobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial selection string. An empty string indicates no selection.
pos	Window position.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
choices	An array of strings with which to initialise the control.
style	Window style. See wxComboBox.
validator	Window validator.
name	Window name.

wxPerl Note: Use an array reference for the *choices* parameter.

See also

Create(), wxValidator

virtual wxComboBox::~wxComboBox ( ) `[virtual]`

Destructor, destroying the combobox.

21.108.3 Member Function Documentation

bool wxComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = `0`, const wxString *choices[]* = `(const wxString *) NULL`, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxComboBoxNameStr`)

Creates the combobox for two-step construction.

Derived classes should call or replace this function. See wxComboBox() for further details.

bool wxComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxComboBoxNameStr)

Creates the combobox for two-step construction.

Derived classes should call or replace this function. See wxComboBox() for further details.

virtual void wxComboBox::Dismiss ( ) [virtual]

Hides the list box portion of the combo box.

Currently this method is implemented in wxMSW, wxGTK and wxOSX/Cocoa.

Notice that calling this function will generate a wxEVT_COMBOBOX_CLOSEUP event except under wxOSX where generation of this event is not supported at all.

Since

2.9.1

virtual int wxComboBox::FindString (const wxString & *string*, bool *caseSensitive* = false) const [virtual]

Finds an item whose label matches the given string.

Parameters

string	String to find.
caseSensitive	Whether search is case sensitive (default is not).

Returns

The zero-based position of the item, or wxNOT_FOUND if the string was not found.

Reimplemented from wxItemContainerImmutable.

virtual unsigned int wxComboBox::GetCount ( ) const [virtual]

Returns the number of items in the control.

See also

IsEmpty()

Implements wxItemContainerImmutable.

virtual int wxComboBox::GetCurrentSelection ( ) const [virtual]

Returns the item being selected right now.

This function does the same things as wxChoice::GetCurrentSelection() and returns the item currently selected in the dropdown list if it's open or the same thing as wxControlWithItems::GetSelection() otherwise.

virtual long wxComboBox::GetInsertionPoint ( ) const [virtual]

Same as wxTextEntry::GetInsertionPoint().

Note

Under wxMSW, this function always returns 0 if the combobox doesn't have the focus.

Reimplemented from wxTextEntry.

virtual int wxComboBox::GetSelection ( ) const `[virtual]`

Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

Returns

The position of the current selection.

Remarks

This method can be used with single selection list boxes only, you should use wxListBox::GetSelections() for the list boxes with wxLB_MULTIPLE style.

See also

SetSelection(), GetStringSelection()

Implements wxItemContainerImmutable.

virtual void wxComboBox::GetSelection (long * *from*, long * *to*) const `[virtual]`

Gets the current selection span.

If the returned values are equal, there was no selection. Please note that the indices returned may be used with the other wxTextCtrl methods but don't necessarily represent the correct indices into the string returned by GetValue() for multiline controls under Windows (at least,) you should use GetStringSelection() to get the selected text.

Parameters

from	The returned first position.
to	The returned last position.

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (from, to).

Reimplemented from wxTextEntry.

virtual wxString wxComboBox::GetString (unsigned int *n*) const `[virtual]`

Returns the label of the item with the given index.

Parameters

n	The zero-based index.

Returns

The label of the item or an empty string if the position was invalid.

Implements wxItemContainerImmutable.

virtual wxString wxComboBox::GetStringSelection ( ) const `[virtual]`

Gets the text currently selected in the control.

If there is no selection, the returned string is empty.

Reimplemented from wxTextEntry.

bool wxComboBox::IsEmpty ( ) const `[virtual]`

IsEmpty() is not available in this class.

This method is documented here only to notice that it can't be used with this class because of the ambiguity between the methods with the same name inherited from wxItemContainer and wxTextEntry base classes.

Because of this, any attempt to call it results in a compilation error and you should use either IsListEmpty() or IsTextEmpty() depending on what exactly do you want to test.

Reimplemented from wxTextEntry.

bool wxComboBox::IsListEmpty ( ) const

Returns true if the list of combobox choices is empty.

Use this method instead of (not available in this class) IsEmpty() to test if the list of items is empty.

Since

2.9.3

bool wxComboBox::IsTextEmpty ( ) const

Returns true if the text of the combobox is empty.

Use this method instead of (not available in this class) IsEmpty() to test if the text currently entered into the combobox is empty.

Since

2.9.3

virtual void wxComboBox::Popup ( ) `[virtual]`

Shows the list box portion of the combo box.

Currently this method is implemented in wxMSW, wxGTK and wxOSX/Cocoa.

Notice that calling this function will generate a `wxEVT_COMBOBOX_DROPDOWN` event except under wxOSX where generation of this event is not supported at all.

Since

2.9.1

virtual void wxComboBox::SetSelection (long *from*, long *to*) `[virtual]`

Same as wxTextEntry::SetSelection().

Reimplemented from wxTextEntry.

virtual void wxComboBox::SetSelection (int *n*) `[virtual]`

Sets the selection to the given item *n* or removes the selection entirely if *n* == `wxNOT_FOUND`.

Note that this does not cause any command events to be emitted nor does it deselect any other items in the controls which support multiple selections.

Parameters

n	The string position to select, starting from zero.

See also

SetString(), SetStringSelection()

Implements wxItemContainerImmutable.

virtual void wxComboBox::SetString (unsigned int *n*, const wxString & *text*) [virtual]

Changes the text of the specified combobox item.

Notice that if the item is the currently selected one, i.e. if its text is displayed in the text part of the combobox, then the text is also replaced with the new *text*.

Implements wxItemContainerImmutable.

virtual void wxComboBox::SetValue (const wxString & *text*) [virtual]

Sets the text for the combobox text field.

Notice that this method will generate a `wxEVT_TEXT` event, use wxTextEntry::ChangeValue() if this is undesirable.

Note

For a combobox with `wxCB_READONLY` style the string must be in the combobox choices list, otherwise the call to SetValue() is ignored. This is case insensitive.

Parameters

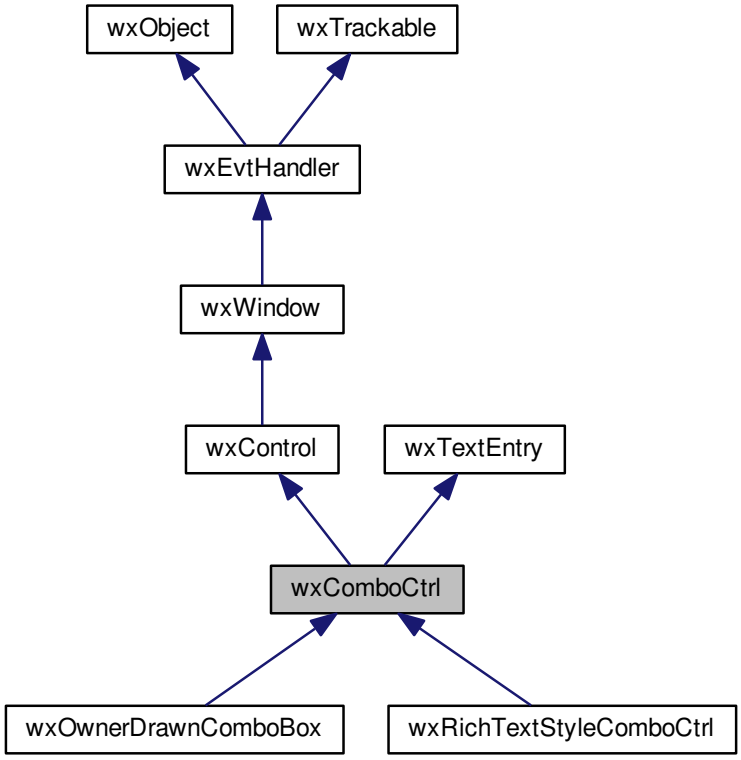
text	The text to set.

Reimplemented from wxTextEntry.

21.109 wxComboCtrl Class Reference

```
#include <wx/combo.h>
```

Inheritance diagram for wxComboCtrl:



21.109.1 Detailed Description

A combo control is a generic combobox that allows totally custom popup.

In addition it has other customization features. For instance, position and size of the dropdown button can be changed.

21.109.2 Setting Custom Popup for wxComboCtrl

wxComboCtrl needs to be told somehow which control to use and this is done by SetPopupControl(). However, we need something more than just a wxControl in this method as, for example, we need to call SetStringValue("initial text value") and wxControl doesn't have such method. So we also need a wxComboPopup which is an interface which must be implemented by a control to be usable as a popup.

We couldn't derive wxComboPopup from wxControl as this would make it impossible to have a class deriving from a wxWidgets control and from it, so instead it is just a mix-in.

Here's a minimal sample of wxListView popup:

```
#include <wx/combo.h>
#include <wx/listctrl.h>

class wxListViewComboPopup : public wxListView, public wxComboPopup
{
public:
    // Initialize member variables
    virtual void Init()
```

```
    {
        m_value = -1;
    }

    // Create popup control
    virtual bool Create(wxWindow* parent)
    {
        return wxListView::Create(parent,1,wxPoint(0,0),
      wxDefaultSize);
    }

    // Return pointer to the created control
    virtual wxWindow *GetControl() { return this; }

    // Translate string into a list selection
    virtual void SetStringValue(const wxString& s)
    {
        int n = wxListView::FindItem(-1,s);
        if ( n >= 0 && n < wxListView::GetItemCount() )
            wxListView::Select(n);
    }

    // Get list selection as a string
    virtual wxString GetStringValue() const
    {
        if ( m_value >= 0 )
        return wxListView::GetItemText(m_value);
        return wxEmptyString;
    }

    // Do mouse hot-tracking (which is typical in list popups)
    void OnMouseMove(wxMouseEvent& event)
    {
        // TODO: Move selection to cursor
    }

    // On mouse left up, set the value and close the popup
    void OnMouseClick(wxMouseEvent& WXUNUSED(event))
    {
        m_value = wxListView::GetFirstSelected();

        // TODO: Send event as well

        Dismiss();
    }

protected:

    int m_value; // current item index

private:
    wxDECLARE_EVENT_TABLE();
};

wxBEGIN_EVENT_TABLE(wxListViewComboPopup, wxListView)
    EVT_MOTION(wxListViewComboPopup::OnMouseMove)
    EVT_LEFT_UP(wxListViewComboPopup::OnMouseClick)
wxEND_EVENT_TABLE()
```

Here's how you would create and populate it in a dialog constructor:

```
wxComboCtrl* comboCtrl = new wxComboCtrl(this, wxID_ANY,
      wxEmptyString);

wxListViewComboPopup* popupCtrl = new wxListViewComboPopup();

// It is important to call SetPopupControl() as soon as possible
comboCtrl->SetPopupControl(popupCtrl);

// Populate using wxListView methods
popupCtrl->InsertItem(popupCtrl->GetItemCount(), "First Item");
popupCtrl->InsertItem(popupCtrl->GetItemCount(), "Second Item");
popupCtrl->InsertItem(popupCtrl->GetItemCount(), "Third Item");
```

Styles

This class supports the following styles:

- wxCB_READONLY: Text will not be editable.
- wxCB_SORT: Sorts the entries in the list alphabetically.

- wxTE_PROCESS_ENTER: The control will generate the event `wxEVT_TEXT_ENTER` (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls). Windows only.
- wxCC_SPECIAL_DCLICK: Double-clicking triggers a call to popup's OnComboDoubleClick. Actual behaviour is defined by a derived class. For instance, wxOwnerDrawnComboBox will cycle an item. This style only applies if wxCB_READONLY is used as well.
- wxCC_STD_BUTTON: Drop button will behave more like a standard push button.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_TEXT(id, func): Process a `wxEVT_TEXT` event, when the text changes.
- EVT_TEXT_ENTER(id, func): Process a `wxEVT_TEXT_ENTER` event, when RETURN is pressed in the combo control.
- EVT_COMBOBOX_DROPDOWN(id, func): Process a `wxEVT_COMBOBOX_DROPDOWN` event, which is generated when the popup window is shown (drops down).
- EVT_COMBOBOX_CLOSEUP(id, func): Process a `wxEVT_COMBOBOX_CLOSEUP` event, which is generated when the popup window of the combo control disappears (closes up). You should avoid adding or deleting items in this event.

Library: wxCore

Category: Controls

See also

wxComboBox, wxChoice, wxOwnerDrawnComboBox, wxComboPopup, wxCommandEvent

Public Member Functions

- wxComboCtrl ()

  Default constructor.
- wxComboCtrl (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Constructor, creating and showing a combo control.
- virtual ~wxComboCtrl ()

  Destructor, destroying the combo control.
- virtual void Copy ()

  Copies the selected text to the clipboard.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Creates the combo control for two-step construction.
- virtual void Cut ()

  Copies the selected text to the clipboard and removes the selection.

- virtual void Dismiss ()

  Dismisses the popup window.
- void EnablePopupAnimation (bool enable=true)

  Enables or disables popup animation, if any, depending on the value of the argument.
- virtual bool IsKeyPopupToggle (const wxKeyEvent &event) const

  Returns true if given key combination should toggle the popup.
- virtual void PrepareBackground (wxDC &dc, const wxRect &rect, int flags) const

  Prepare background of combo control or an item in a dropdown list in a way typical on platform.
- bool ShouldDrawFocus () const

  Returns true if focus indicator should be drawn in the control.
- const wxBitmap & GetBitmapDisabled () const

  Returns disabled button bitmap that has been set with SetButtonBitmaps().
- const wxBitmap & GetBitmapHover () const

  Returns button mouse hover bitmap that has been set with SetButtonBitmaps().
- const wxBitmap & GetBitmapNormal () const

  Returns default button bitmap that has been set with SetButtonBitmaps().
- const wxBitmap & GetBitmapPressed () const

  Returns depressed button bitmap that has been set with SetButtonBitmaps().
- wxSize GetButtonSize ()

  Returns current size of the dropdown button.
- int GetCustomPaintWidth () const

  Returns custom painted area in control.
- virtual wxString GetHint () const

  Returns the current hint string.
- virtual long GetInsertionPoint () const

  Returns the insertion point for the combo control's text field.
- virtual long GetLastPosition () const

  Returns the last position in the combo control text field.
- wxPoint GetMargins () const

  Returns the margins used by the control.
- wxComboPopup * GetPopupControl ()

  Returns current popup interface that has been set with SetPopupControl().
- wxWindow * GetPopupWindow () const

  Returns popup window containing the popup control.
- wxTextCtrl * GetTextCtrl () const

  Get the text control which is part of the combo control.
- wxCoord GetTextIndent () const

  Returns actual indentation in pixels.
- const wxRect & GetTextRect () const

  Returns area covered by the text field (includes everything except borders and the dropdown button).
- virtual wxString GetValue () const

  Returns text representation of the current value.
- virtual void HidePopup (bool generateEvent=false)

  Dismisses the popup window.
- bool IsPopupShown () const

  Returns true if the popup is currently shown.
- bool IsPopupWindowState (int state) const

  Returns true if the popup window is in the given state.
- virtual void OnButtonClick ()

  Implement in a derived class to define what happens on dropdown button click.
- virtual void Paste ()

Pastes text from the clipboard to the text field.

- virtual void Popup ()

  Shows the popup portion of the combo control.

- virtual void Remove (long from, long to)

  Removes the text between the two positions in the combo control text field.

- virtual void Replace (long from, long to, const wxString &text)

  Replaces the text between two positions with the given text, in the combo control text field.

- void SetButtonBitmaps (const wxBitmap &bmpNormal, bool pushButtonBg=false, const wxBitmap &bmpPressed=wxNullBitmap, const wxBitmap &bmpHover=wxNullBitmap, const wxBitmap &bmpDisabled=wxNullBitmap)

  Sets custom dropdown button graphics.

- void SetButtonPosition (int width=-1, int height=-1, int side=wxRIGHT, int spacingX=0)

  Sets size and position of dropdown button.

- void SetCustomPaintWidth (int width)

  Set width, in pixels, of custom painted area in control without `wxCB_READONLY` style.

- virtual bool SetHint (const wxString &hint)

  Sets a hint shown in an empty unfocused combo control.

- virtual void SetInsertionPoint (long pos)

  Sets the insertion point in the text field.

- virtual void SetInsertionPointEnd ()

  Sets the insertion point at the end of the combo control text field.

- void SetPopupAnchor (int anchorSide)

  Set side of the control to which the popup will align itself.

- void SetPopupControl (wxComboPopup *popup)

  Set popup interface class derived from wxComboPopup.

- void SetPopupExtents (int extLeft, int extRight)

  Extends popup size horizontally, relative to the edges of the combo control.

- void SetPopupMaxHeight (int height)

  Sets preferred maximum height of the popup.

- void SetPopupMinWidth (int width)

  Sets minimum width of the popup.

- virtual void SetSelection (long from, long to)

  Selects the text between the two positions, in the combo control text field.

- void SetText (const wxString &value)

  Sets the text for the text field without affecting the popup.

- void SetTextCtrlStyle (int style)

  Set a custom window style for the embedded wxTextCtrl.

- void SetTextIndent (int indent)

  This will set the space in pixels between left edge of the control and the text, regardless whether control is read-only or not.

- virtual void SetValue (const wxString &value)

  Sets the text for the combo control text field.

- void SetValueByUser (const wxString &value)

  Changes value of the control as if user had done it by selecting an item from a combo box drop-down list.

- virtual void ShowPopup ()

  Show the popup.

- virtual void Undo ()

  Undoes the last edit in the text field.

- void UseAltPopupWindow (bool enable=true)

  Enable or disable usage of an alternative popup window, which guarantees ability to focus the popup control, and allows common native controls to function normally.

- bool SetMargins (const wxPoint &pt)

  Attempts to set the control margins.
- bool SetMargins (wxCoord left, wxCoord top=-1)

  Attempts to set the control margins.

Static Public Member Functions

- static int GetFeatures ()

  Returns features supported by wxComboCtrl.

Protected Types

- enum {
  ShowBelow = 0x0000,
  ShowAbove = 0x0001,
  CanDeferShow = 0x0002 }

  Flags for DoShowPopup() and AnimateShow().

Protected Member Functions

- virtual bool AnimateShow (const wxRect &rect, int flags)

  This member function is not normally called in application code.
- virtual void DoSetPopupControl (wxComboPopup *popup)

  This member function is not normally called in application code.
- virtual void DoShowPopup (const wxRect &rect, int flags)

  This member function is not normally called in application code.

Additional Inherited Members

21.109.3 Member Enumeration Documentation

anonymous enum `[protected]`

Flags for DoShowPopup() and AnimateShow().

Enumerator

ShowBelow Show popup below the control.

ShowAbove Show popup above the control.

CanDeferShow Can only return true from AnimateShow() if this is set.

21.109.4 Constructor & Destructor Documentation

wxComboCtrl::wxComboCtrl ( )

Default constructor.

wxComboCtrl::wxComboCtrl (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxComboBoxNameStr`)

Constructor, creating and showing a combo control.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial selection string. An empty string indicates no selection.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxComboCtrl.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxComboCtrl::~wxComboCtrl ( ) [virtual]

Destructor, destroying the combo control.

21.109.5 Member Function Documentation

virtual bool wxComboCtrl::AnimateShow (const wxRect & *rect*, int *flags*) [protected], [virtual]

This member function is not normally called in application code.

Instead, it can be implemented in a derived class to create a custom popup animation.

The parameters are the same as those for DoShowPopup().

Returns

true if animation finishes before the function returns, false otherwise. In the latter case you need to manually call DoShowPopup() after the animation ends.

virtual void wxComboCtrl::Copy ( ) [virtual]

Copies the selected text to the clipboard.

Reimplemented from wxTextEntry.

bool wxComboCtrl::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxComboBoxNameStr)

Creates the combo control for two-step construction.

Derived classes should call or replace this function. See wxComboCtrl() for further details.

virtual void wxComboCtrl::Cut ( ) [virtual]

Copies the selected text to the clipboard and removes the selection.

Reimplemented from wxTextEntry.

virtual void wxComboCtrl::Dismiss ( ) [virtual]

Dismisses the popup window.

Notice that calling this function will generate a `wxEVT_COMBOBOX_CLOSEUP` event.

Since

2.9.2

virtual void wxComboCtrl::DoSetPopupControl (wxComboPopup * *popup*) [protected], [virtual]

This member function is not normally called in application code.

Instead, it can be implemented in a derived class to return default wxComboPopup, in case *popup* is NULL.

Note

If you have implemented OnButtonClick() to do something else than show the popup, then DoSetPopupControl() must always set *popup* to NULL.

virtual void wxComboCtrl::DoShowPopup (const wxRect & *rect,* int *flags*) [protected], [virtual]

This member function is not normally called in application code.

Instead, it must be called in a derived class to make sure popup is properly shown after a popup animation has finished (but only if AnimateShow() did not finish the animation within its function scope).

Parameters

rect	Position to show the popup window at, in screen coordinates.
flags	Combination of any of the following: wxComboCtrl::ShowAbove, and wxComboCtrl::CanDeferShow.

void wxComboCtrl::EnablePopupAnimation (bool *enable* = `true`)

Enables or disables popup animation, if any, depending on the value of the argument.

const wxBitmap& wxComboCtrl::GetBitmapDisabled ( ) const

Returns disabled button bitmap that has been set with SetButtonBitmaps().

Returns

A reference to the disabled state bitmap.

const wxBitmap& wxComboCtrl::GetBitmapHover ( ) const

Returns button mouse hover bitmap that has been set with SetButtonBitmaps().

Returns

A reference to the mouse hover state bitmap.

const wxBitmap& wxComboCtrl::GetBitmapNormal ( ) const

Returns default button bitmap that has been set with SetButtonBitmaps().

Returns

A reference to the normal state bitmap.

const wxBitmap& wxComboCtrl::GetBitmapPressed ( ) const

Returns depressed button bitmap that has been set with SetButtonBitmaps().

Returns

A reference to the depressed state bitmap.

wxSize wxComboCtrl::GetButtonSize ( )

Returns current size of the dropdown button.

int wxComboCtrl::GetCustomPaintWidth ( ) const

Returns custom painted area in control.

See also

SetCustomPaintWidth().

static int wxComboCtrl::GetFeatures ( ) [static]

Returns features supported by wxComboCtrl.

If needed feature is missing, you need to instead use wxGenericComboCtrl, which however may lack a native look and feel (but otherwise sports identical API).

Returns

Value returned is a combination of the flags defined in wxComboCtrlFeatures.

virtual wxString wxComboCtrl::GetHint ( ) const [virtual]

Returns the current hint string.

See SetHint() for more information about hints.

Since

2.9.1

Reimplemented from wxTextEntry.

virtual long wxComboCtrl::GetInsertionPoint ( ) const `[virtual]`

Returns the insertion point for the combo control's text field.

Note

Under Windows, this function always returns 0 if the combo control doesn't have the focus.

Reimplemented from wxTextEntry.

virtual long wxComboCtrl::GetLastPosition ( ) const `[virtual]`

Returns the last position in the combo control text field.

Reimplemented from wxTextEntry.

wxPoint wxComboCtrl::GetMargins ( ) const

Returns the margins used by the control.

The `x` field of the returned point is the horizontal margin and the `y` field is the vertical one.

Remarks

If given margin cannot be accurately determined, its value will be set to -1.

See also

SetMargins()

Since

2.9.1

wxComboPopup* wxComboCtrl::GetPopupControl ( )

Returns current popup interface that has been set with SetPopupControl().

wxWindow* wxComboCtrl::GetPopupWindow ( ) const

Returns popup window containing the popup control.

wxTextCtrl* wxComboCtrl::GetTextCtrl ( ) const

Get the text control which is part of the combo control.

wxCoord wxComboCtrl::GetTextIndent ( ) const

Returns actual indentation in pixels.

Deprecated Use GetMargins() instead.

const wxRect& wxComboCtrl::GetTextRect ( ) const

Returns area covered by the text field (includes everything except borders and the dropdown button).

virtual wxString wxComboCtrl::GetValue ( ) const `[virtual]`

Returns text representation of the current value.

For writable combo control it always returns the value in the text field.

Reimplemented from wxTextEntry.

virtual void wxComboCtrl::HidePopup (bool *generateEvent* = `false`) `[virtual]`

Dismisses the popup window.

Parameters

generateEvent	Set this to true in order to generate `wxEVT_COMBOBOX_CLOSEUP` event.

Deprecated Use Dismiss() instead.

virtual bool wxComboCtrl::IsKeyPopupToggle (const wxKeyEvent & *event*) const `[virtual]`

Returns true if given key combination should toggle the popup.

bool wxComboCtrl::IsPopupShown ( ) const

Returns true if the popup is currently shown.

bool wxComboCtrl::IsPopupWindowState (int *state*) const

Returns true if the popup window is in the given state.

Possible values are:

wxComboCtrl::Hidden	Popup window is hidden.
wxComboCtrl::Animating	Popup window is being shown, but the popup animation has not yet finished.
wxComboCtrl::Visible	Popup window is fully visible.

virtual void wxComboCtrl::OnButtonClick ( ) `[virtual]`

Implement in a derived class to define what happens on dropdown button click.

Default action is to show the popup.

Note

If you implement this to do something else than show the popup, you must then also implement DoSetPopupControl() to always return NULL.

virtual void wxComboCtrl::Paste ( ) `[virtual]`

Pastes text from the clipboard to the text field.

Reimplemented from wxTextEntry.

virtual void wxComboCtrl::Popup ( ) [virtual]

Shows the popup portion of the combo control.

Notice that calling this function will generate a `wxEVT_COMBOBOX_DROPDOWN` event.

Since

2.9.2

virtual void wxComboCtrl::PrepareBackground (wxDC & *dc*, const wxRect & *rect*, int *flags*) const [virtual]

Prepare background of combo control or an item in a dropdown list in a way typical on platform.

This includes painting the focus/disabled background and setting the clipping region.

Unless you plan to paint your own focus indicator, you should always call this in your wxComboPopup::PaintComboControl implementation. In addition, it sets pen and text colour to what looks good and proper against the background.

flags: wxRendererNative flags: wxCONTROL_ISSUBMENU: is drawing a list item instead of combo control wxCONTROL_SELECTED: list item is selected wxCONTROL_DISABLED: control/item is disabled

virtual void wxComboCtrl::Remove (long *from*, long *to*) [virtual]

Removes the text between the two positions in the combo control text field.

Parameters

from	The first position.
to	The last position.

Reimplemented from wxTextEntry.

virtual void wxComboCtrl::Replace (long *from*, long *to*, const wxString & *text*) [virtual]

Replaces the text between two positions with the given text, in the combo control text field.

Parameters

from	The first position.
to	The second position.
text	The text to insert.

Reimplemented from wxTextEntry.

void wxComboCtrl::SetButtonBitmaps (const wxBitmap & *bmpNormal*, bool *pushButtonBg* = `false`, const wxBitmap & *bmpPressed* = wxNullBitmap, const wxBitmap & *bmpHover* = wxNullBitmap, const wxBitmap & *bmpDisabled* = wxNullBitmap)

Sets custom dropdown button graphics.

Parameters

bmpNormal	Default button image.
pushButtonBg	If true, blank push button background is painted below the image.

bmpPressed	Depressed button image.
bmpHover	Button image when mouse hovers above it. This should be ignored on platforms and themes that do not generally draw different kind of button on mouse hover.
bmpDisabled	Disabled button image.

void wxComboCtrl::SetButtonPosition (int *width* = -1, int *height* = -1, int *side* = wxRIGHT, int *spacingX* = 0)

Sets size and position of dropdown button.

Parameters

width	Button width. Value = 0 specifies default.
height	Button height. Value = 0 specifies default.
side	Indicates which side the button will be placed. Value can be wxLEFT or wxRIGHT.
spacingX	Horizontal spacing around the button. Default is 0.

void wxComboCtrl::SetCustomPaintWidth (int *width*)

Set width, in pixels, of custom painted area in control without `wxCB_READONLY` style.

In read-only wxOwnerDrawnComboBox, this is used to indicate area that is not covered by the focus rectangle.

virtual bool wxComboCtrl::SetHint (const wxString & *hint*) [virtual]

Sets a hint shown in an empty unfocused combo control.

Notice that hints are known as *cue banners* under MSW or *placeholder strings* under OS X.

See also

wxTextEntry::SetHint()

Since

2.9.1

Reimplemented from wxTextEntry.

virtual void wxComboCtrl::SetInsertionPoint (long *pos*) [virtual]

Sets the insertion point in the text field.

Parameters

pos	The new insertion point.

Reimplemented from wxTextEntry.

virtual void wxComboCtrl::SetInsertionPointEnd ( ) [virtual]

Sets the insertion point at the end of the combo control text field.

Reimplemented from wxTextEntry.

bool wxComboCtrl::SetMargins (const wxPoint & *pt*)

Attempts to set the control margins.

When margins are given as wxPoint, x indicates the left and y the top margin. Use -1 to indicate that an existing value should be used.

Returns

true if setting of all requested margins was successful.

Since

2.9.1

bool wxComboCtrl::SetMargins (wxCoord *left*, wxCoord *top* = -1)

Attempts to set the control margins.

When margins are given as wxPoint, x indicates the left and y the top margin. Use -1 to indicate that an existing value should be used.

Returns

true if setting of all requested margins was successful.

Since

2.9.1

void wxComboCtrl::SetPopupAnchor (int *anchorSide*)

Set side of the control to which the popup will align itself.

Valid values are `wxLEFT`, `wxRIGHT` and 0. The default value 0 means that the most appropriate side is used (which, currently, is always `wxLEFT`).

void wxComboCtrl::SetPopupControl (wxComboPopup * *popup*)

Set popup interface class derived from wxComboPopup.

This method should be called as soon as possible after the control has been created, unless OnButtonClick() has been overridden.

void wxComboCtrl::SetPopupExtents (int *extLeft*, int *extRight*)

Extends popup size horizontally, relative to the edges of the combo control.

Parameters

extLeft	How many pixel to extend beyond the left edge of the control. Default is 0.
extRight	How many pixel to extend beyond the right edge of the control. Default is 0.

Remarks

Popup minimum width may override arguments. It is up to the popup to fully take this into account.

void wxComboCtrl::SetPopupMaxHeight (int *height*)

Sets preferred maximum height of the popup.

Remarks

Value -1 indicates the default.

void wxComboCtrl::SetPopupMinWidth (int *width*)

Sets minimum width of the popup.

If wider than combo control, it will extend to the left.

Remarks

Value -1 indicates the default. Also, popup implementation may choose to ignore this.

virtual void wxComboCtrl::SetSelection (long *from,* long *to*) `[virtual]`

Selects the text between the two positions, in the combo control text field.

Parameters

from	The first position.
to	The second position.

Reimplemented from wxTextEntry.

void wxComboCtrl::SetText (const wxString & *value*)

Sets the text for the text field without affecting the popup.

Thus, unlike SetValue(), it works equally well with combo control using `wxCB_READONLY` style.

void wxComboCtrl::SetTextCtrlStyle (int *style*)

Set a custom window style for the embedded wxTextCtrl.

Usually you will need to use this during two-step creation, just before Create(). For example:

```
wxComboCtrl* comboCtrl = new wxComboCtrl();

// Let's make the text right-aligned
comboCtrl->SetTextCtrlStyle(wxTE_RIGHT);

comboCtrl->Create(parent, wxID_ANY, wxEmptyString);
```

void wxComboCtrl::SetTextIndent (int *indent*)

This will set the space in pixels between left edge of the control and the text, regardless whether control is read-only or not.

Value -1 can be given to indicate platform default.

Deprecated Use SetMargins() instead.

virtual void wxComboCtrl::SetValue (const wxString & *value*) `[virtual]`

Sets the text for the combo control text field.

Note

For a combo control with `wxCB_READONLY` style the string must be accepted by the popup (for instance, exist in the dropdown list), otherwise the call to SetValue() is ignored.

Reimplemented from wxTextEntry.

void wxComboCtrl::SetValueByUser (const wxString & *value*)

Changes value of the control as if user had done it by selecting an item from a combo box drop-down list.

bool wxComboCtrl::ShouldDrawFocus ( ) const

Returns true if focus indicator should be drawn in the control.

virtual void wxComboCtrl::ShowPopup ( ) `[virtual]`

Show the popup.

Deprecated Use Popup() instead.

virtual void wxComboCtrl::Undo ( ) `[virtual]`

Undoes the last edit in the text field.

Windows only.

Reimplemented from wxTextEntry.

void wxComboCtrl::UseAltPopupWindow (bool *enable* = `true`)

Enable or disable usage of an alternative popup window, which guarantees ability to focus the popup control, and allows common native controls to function normally.

This alternative popup window is usually a wxDialog, and as such, when it is shown, its parent top-level window will appear as if the focus has been lost from it.

21.110 wxComboCtrlFeatures Struct Reference

`#include <wx/combo.h>`

21.110.1 Detailed Description

Features enabled for wxComboCtrl.

See also

wxComboCtrl::GetFeatures()

Public Types

- enum {
  MovableButton = 0x0001,
  BitmapButton = 0x0002,
  ButtonSpacing = 0x0004,
  TextIndent = 0x0008,
  PaintControl = 0x0010,
  PaintWritable = 0x0020,
  Borderless = 0x0040,
  All }

21.110.2 Member Enumeration Documentation

anonymous enum

Enumerator

MovableButton Button can be on either side of control.

BitmapButton Button may be replaced with bitmap.

ButtonSpacing Button can have spacing from the edge of the control.

TextIndent wxComboCtrl::SetMargins() can be used.

PaintControl Combo control itself can be custom painted.

PaintWritable A variable-width area in front of writable combo control's textctrl can be custom painted.

Borderless wxNO_BORDER window style works.

All All features.

21.111 wxComboPopup Class Reference

```
#include <wx/combo.h>
```

21.111.1 Detailed Description

In order to use a custom popup with wxComboCtrl, an interface class must be derived from wxComboPopup.

For more information on how to use it, see Setting Custom Popup for wxComboCtrl.

Library: wxCore

Category: Controls

See also

wxComboCtrl

Public Member Functions

- wxComboPopup ()
  Default constructor.
- virtual bool Create (wxWindow *parent)=0
  The derived class must implement this to create the popup control.

- virtual void DestroyPopup ()

  You only need to implement this member function if you create your popup class in non-standard way.
- void Dismiss ()

  Utility function that hides the popup.
- virtual bool FindItem (const wxString &item, wxString *trueItem=NULL)

  Implement to customize matching of value string to an item container entry.
- virtual wxSize GetAdjustedSize (int minWidth, int prefHeight, int maxHeight)

  The derived class may implement this to return adjusted size for the popup control, according to the variables given.
- wxComboCtrl * GetComboCtrl () const

  Returns pointer to the associated parent wxComboCtrl.
- virtual wxWindow * GetControl ()=0

  The derived class must implement this to return pointer to the associated control created in Create().
- virtual wxString GetStringValue () const =0

  The derived class must implement this to return string representation of the value.
- virtual void Init ()

  The derived class must implement this to initialize its internal variables.
- bool IsCreated () const

  Utility method that returns true if Create has been called.
- virtual bool LazyCreate ()

  The derived class may implement this to return true if it wants to delay call to Create() until the popup is shown for the first time.
- virtual void OnComboDoubleClick ()

  The derived class may implement this to do something when the parent wxComboCtrl gets double-clicked.
- virtual void OnComboKeyEvent (wxKeyEvent &event)

  The derived class may implement this to receive key events from the parent wxComboCtrl.
- virtual void OnDismiss ()

  The derived class may implement this to do special processing when popup is hidden.
- virtual void OnPopup ()

  The derived class may implement this to do special processing when popup is shown.
- virtual void PaintComboControl (wxDC &dc, const wxRect &rect)

  The derived class may implement this to paint the parent wxComboCtrl.
- virtual void SetStringValue (const wxString &value)

  The derived class must implement this to receive string value changes from wxComboCtrl.

Protected Attributes

- wxComboCtrl * m_combo

  Parent wxComboCtrl.

21.111.2 Constructor & Destructor Documentation

wxComboPopup::wxComboPopup ( )

Default constructor.

It is recommended that internal variables are prepared in Init() instead (because m_combo is not valid in constructor).

21.111.3 Member Function Documentation

virtual bool wxComboPopup::Create (wxWindow * *parent*) [pure virtual]

The derived class must implement this to create the popup control.

Returns

true if the call succeeded, false otherwise.

virtual void wxComboPopup::DestroyPopup ( ) [virtual]

You only need to implement this member function if you create your popup class in non-standard way.

The default implementation can handle both multiple-inherited popup control (as seen in wxComboCtrl samples) and one allocated separately in heap.

If you do completely re-implement this function, make sure it calls Destroy() for the popup control and also deletes *this* object (usually as the last thing).

void wxComboPopup::Dismiss ( )

Utility function that hides the popup.

virtual bool wxComboPopup::FindItem (const wxString & *item,* wxString * *trueItem* = NULL) [virtual]

Implement to customize matching of value string to an item container entry.

Parameters

item	String entered, usually by user or from SetValue() call.
trueItem	When item matches an entry, but the entry's string representation is not exactly the same (case mismatch, for example), then the true item string should be written back to here, if it is not a NULL pointer.

Remarks

Default implementation always return true and does not alter trueItem.

virtual wxSize wxComboPopup::GetAdjustedSize (int *minWidth,* int *prefHeight,* int *maxHeight*) [virtual]

The derived class may implement this to return adjusted size for the popup control, according to the variables given.

Parameters

minWidth	Preferred minimum width.
prefHeight	Preferred height. May be -1 to indicate no preference.
maxHeight	Max height for window, as limited by screen size.

Remarks

This function is called each time popup is about to be shown.

wxComboCtrl* wxComboPopup::GetComboCtrl ( ) const

Returns pointer to the associated parent wxComboCtrl.

virtual wxWindow* wxComboPopup::GetControl ( ) [pure virtual]

The derived class must implement this to return pointer to the associated control created in Create().

virtual wxString wxComboPopup::GetStringValue ( ) const [pure virtual]

The derived class must implement this to return string representation of the value.

virtual void wxComboPopup::Init ( ) [virtual]

The derived class must implement this to initialize its internal variables.

This method is called immediately after construction finishes. m_combo member variable has been initialized before the call.

bool wxComboPopup::IsCreated ( ) const

Utility method that returns true if Create has been called.

Useful in conjunction with LazyCreate().

virtual bool wxComboPopup::LazyCreate ( ) [virtual]

The derived class may implement this to return true if it wants to delay call to Create() until the popup is shown for the first time.

It is more efficient, but on the other hand it is often more convenient to have the control created immediately.

Remarks

Base implementation returns false.

virtual void wxComboPopup::OnComboDoubleClick ( ) [virtual]

The derived class may implement this to do something when the parent wxComboCtrl gets double-clicked.

virtual void wxComboPopup::OnComboKeyEvent (wxKeyEvent & *event*) [virtual]

The derived class may implement this to receive key events from the parent wxComboCtrl.

Events not handled should be skipped, as usual.

virtual void wxComboPopup::OnDismiss ( ) [virtual]

The derived class may implement this to do special processing when popup is hidden.

virtual void wxComboPopup::OnPopup ( ) [virtual]

The derived class may implement this to do special processing when popup is shown.

virtual void wxComboPopup::PaintComboControl (wxDC & *dc*, const wxRect & *rect*) [virtual]

The derived class may implement this to paint the parent wxComboCtrl.

Default implementation draws value as string.

virtual void wxComboPopup::SetStringValue (const wxString & *value*) `[virtual]`

The derived class must implement this to receive string value changes from wxComboCtrl.

21.111.4 Member Data Documentation

wxComboCtrl* wxComboPopup::m_combo `[protected]`

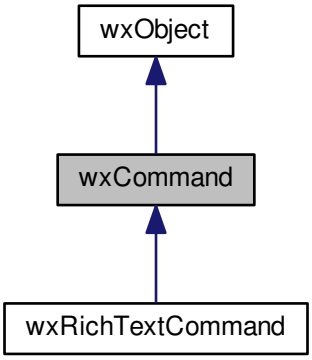
Parent wxComboCtrl.

This member variable is prepared automatically before Init() is called.

21.112 wxCommand Class Reference

`#include <wx/cmdproc.h>`

Inheritance diagram for wxCommand:



21.112.1 Detailed Description

wxCommand is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other means provided by the application to change the data or view.

Library: wxCore

Category: Document/View Framework

See also

wxCommand Overview

Public Member Functions

- wxCommand (bool canUndo=false, const wxString &name=wxEmptyString)

  Constructor.
- virtual ~wxCommand ()

  Destructor.
- virtual bool CanUndo () const

  Returns true if the command can be undone, false otherwise.
- virtual bool Do ()=0

  Override this member function to execute the appropriate action when called.
- virtual wxString GetName () const

  Returns the command name.
- virtual bool Undo ()=0

  Override this member function to un-execute a previous Do.

Additional Inherited Members

21.112.2 Constructor & Destructor Documentation

wxCommand::wxCommand (bool *canUndo* = `false`, const wxString & *name* = wxEmptyString)

Constructor.

wxCommand is an abstract class, so you will need to derive a new class and call this constructor from your own constructor.

Parameters

canUndo	Tells the command processor whether this command is undo-able. You can achieve the same functionality by overriding the CanUndo() member function (if for example the criteria for undoability is context-dependent).
name	Must be supplied for the command processor to display the command name in the application's edit menu.

virtual wxCommand::~wxCommand ( ) `[virtual]`

Destructor.

21.112.3 Member Function Documentation

virtual bool wxCommand::CanUndo ( ) const `[virtual]`

Returns true if the command can be undone, false otherwise.

virtual bool wxCommand::Do ( ) `[pure virtual]`

Override this member function to execute the appropriate action when called.

Returns

true to indicate that the action has taken place, false otherwise. Returning false will indicate to the command processor that the action is not undoable and should not be added to the command history.

Implemented in wxRichTextCommand.

virtual wxString wxCommand::GetName ( ) const [virtual]

Returns the command name.

virtual bool wxCommand::Undo ( ) [pure virtual]

Override this member function to un-execute a previous Do.

How you implement this command is totally application dependent, but typical strategies include:

- Perform an inverse operation on the last modified piece of data in the document. When redone, a copy of data stored in command is pasted back or some operation reapplied. This relies on the fact that you know the ordering of Undos; the user can never Undo at an arbitrary position in the command history.

- Restore the entire document state (perhaps using document transacting). Potentially very inefficient, but possibly easier to code if the user interface and data are complex, and an "inverse execute" operation is hard to write. The docview sample uses the first method, to remove or restore segments in the drawing.

Returns

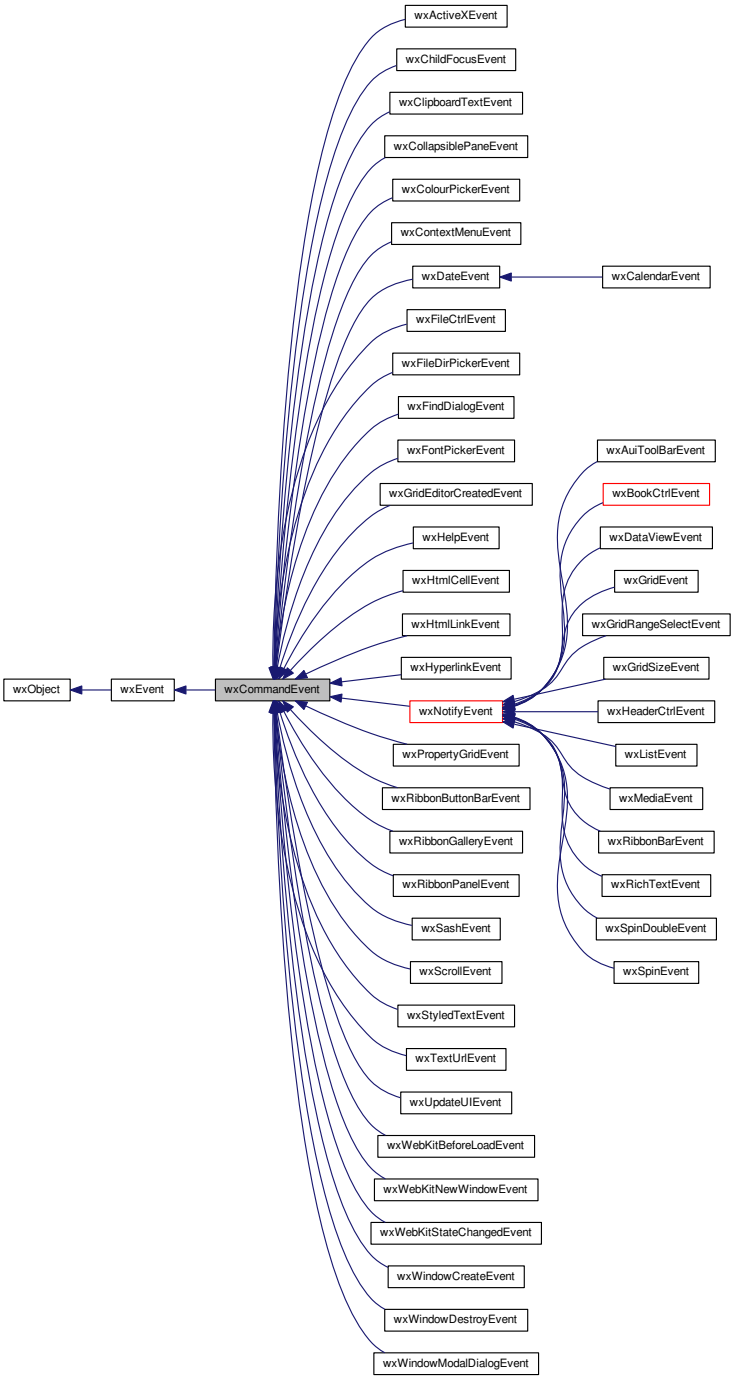
true to indicate that the action has taken place, false otherwise. Returning false will indicate to the command processor that the action is not redoable and no change should be made to the command history.

Implemented in wxRichTextCommand.

21.113 wxCommandEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxCommandEvent:



21.113.1 Detailed Description

This event class contains information about command events, which originate from a variety of simple controls.

Note that wxCommandEvents and wxCommandEvent-derived event classes by default and unlike other wxEvent-derived classes propagate upward from the source window (the window which emits the event) up to the first parent which processes the event. Be sure to read How Events Propagate Upwards.

More complex controls, such as wxTreeCtrl, have separate command event classes.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros:

- EVT_COMMAND(id, event, func): Process a command, supplying the window identifier, command event identifier, and member function.
- EVT_COMMAND_RANGE(id1, id2, event, func): Process a command for a range of window identifiers, supplying the minimum and maximum window identifiers, command event identifier, and member function.
- EVT_BUTTON(id, func): Process a `wxEVT_BUTTON` command, which is generated by a wxButton control.
- EVT_CHECKBOX(id, func): Process a `wxEVT_CHECKBOX` command, which is generated by a wxCheckBox control.
- EVT_CHOICE(id, func): Process a `wxEVT_CHOICE` command, which is generated by a wxChoice control.
- EVT_COMBOBOX(id, func): Process a `wxEVT_COMBOBOX` command, which is generated by a wxComboBox control.
- EVT_LISTBOX(id, func): Process a `wxEVT_LISTBOX` command, which is generated by a wxListBox control.
- EVT_LISTBOX_DCLICK(id, func): Process a `wxEVT_LISTBOX_DCLICK` command, which is generated by a wxListBox control.
- EVT_CHECKLISTBOX(id, func): Process a `wxEVT_CHECKLISTBOX` command, which is generated by a wxCheckListBox control.
- EVT_MENU(id, func): Process a `wxEVT_MENU` command, which is generated by a menu item.
- EVT_MENU_RANGE(id1, id2, func): Process a `wxEVT_MENU` command, which is generated by a range of menu items.
- EVT_CONTEXT_MENU(func): Process the event generated when the user has requested a popup menu to appear by pressing a special keyboard key (under Windows) or by right clicking the mouse.
- EVT_RADIOBOX(id, func): Process a `wxEVT_RADIOBOX` command, which is generated by a wxRadioBox control.
- EVT_RADIOBUTTON(id, func): Process a `wxEVT_RADIOBUTTON` command, which is generated by a wxRadioButton control.
- EVT_SCROLLBAR(id, func): Process a `wxEVT_SCROLLBAR` command, which is generated by a wxScrollBar control. This is provided for compatibility only; more specific scrollbar event macros should be used instead (see wxScrollEvent).
- EVT_SLIDER(id, func): Process a `wxEVT_SLIDER` command, which is generated by a wxSlider control.
- EVT_TEXT(id, func): Process a `wxEVT_TEXT` command, which is generated by a wxTextCtrl control.
- EVT_TEXT_ENTER(id, func): Process a `wxEVT_TEXT_ENTER` command, which is generated by a wxTextCtrl control. Note that you must use wxTE_PROCESS_ENTER flag when creating the control if you want it to generate such events.
- EVT_TEXT_MAXLEN(id, func): Process a `wxEVT_TEXT_MAXLEN` command, which is generated by a wxTextCtrl control when the user tries to enter more characters into it than the limit previously set with SetMaxLength().
- EVT_TOGGLEBUTTON(id, func): Process a `wxEVT_TOGGLEBUTTON` event.
- EVT_TOOL(id, func): Process a `wxEVT_TOOL` event (a synonym for `wxEVT_MENU`). Pass the id of the tool.

- EVT_TOOL_RANGE(id1, id2, func): Process a `wxEVT_TOOL` event for a range of identifiers. Pass the ids of the tools.
- EVT_TOOL_RCLICKED(id, func): Process a `wxEVT_TOOL_RCLICKED` event. Pass the id of the tool. (Not available on wxOSX.)
- EVT_TOOL_RCLICKED_RANGE(id1, id2, func): Process a `wxEVT_TOOL_RCLICKED` event for a range of ids. Pass the ids of the tools. (Not available on wxOSX.)
- EVT_TOOL_ENTER(id, func): Process a `wxEVT_TOOL_ENTER` event. Pass the id of the toolbar itself. The value of wxCommandEvent::GetSelection() is the tool id, or -1 if the mouse cursor has moved off a tool. (Not available on wxOSX.)
- EVT_COMMAND_LEFT_CLICK(id, func): Process a `wxEVT_COMMAND_LEFT_CLICK` command, which is generated by a control (wxMSW only).
- EVT_COMMAND_LEFT_DCLICK(id, func): Process a `wxEVT_COMMAND_LEFT_DCLICK` command, which is generated by a control (wxMSW only).
- EVT_COMMAND_RIGHT_CLICK(id, func): Process a `wxEVT_COMMAND_RIGHT_CLICK` command, which is generated by a control (wxMSW only).
- EVT_COMMAND_SET_FOCUS(id, func): Process a `wxEVT_COMMAND_SET_FOCUS` command, which is generated by a control (wxMSW only).
- EVT_COMMAND_KILL_FOCUS(id, func): Process a `wxEVT_COMMAND_KILL_FOCUS` command, which is generated by a control (wxMSW only).
- EVT_COMMAND_ENTER(id, func): Process a `wxEVT_COMMAND_ENTER` command, which is generated by a control.

Library: wxCore

Category: Events

Public Member Functions

- wxCommandEvent (wxEventType commandEventType=wxEVT_NULL, int id=0)

  Constructor.
- void * GetClientData () const

  Returns client data pointer for a listbox or choice selection event (not valid for a deselection).
- wxClientData * GetClientObject () const

  Returns client object pointer for a listbox or choice selection event (not valid for a deselection).
- long GetExtraLong () const

  Returns extra information dependent on the event objects type.
- int GetInt () const

  Returns the integer identifier corresponding to a listbox, choice or radiobox selection (only if the event was a selection, not a deselection), or a boolean value representing the value of a checkbox.
- int GetSelection () const

  Returns item index for a listbox or choice selection event (not valid for a deselection).
- wxString GetString () const

  Returns item string for a listbox or choice selection event.
- bool IsChecked () const

  This method can be used with checkbox and menu events: for the checkboxes, the method returns true for a selection event and false for a deselection one.
- bool IsSelection () const

For a listbox or similar event, returns true if it is a selection, false if it is a deselection.

- void SetClientData (void *clientData)

  Sets the client data for this event.

- void SetClientObject (wxClientData *clientObject)

  Sets the client object for this event.

- void SetExtraLong (long extraLong)

  *Sets the **m_extraLong** member.*

- void SetInt (int intCommand)

  *Sets the **m_commandInt** member.*

- void SetString (const wxString &string)

  *Sets the **m_commandString** member.*

Additional Inherited Members

21.113.2 Constructor & Destructor Documentation

wxCommandEvent::wxCommandEvent (wxEventType *commandEventType* = wxEVT_NULL, int *id* = 0)

Constructor.

21.113.3 Member Function Documentation

void* wxCommandEvent::GetClientData ( ) const

Returns client data pointer for a listbox or choice selection event (not valid for a deselection).

wxClientData* wxCommandEvent::GetClientObject ( ) const

Returns client object pointer for a listbox or choice selection event (not valid for a deselection).

long wxCommandEvent::GetExtraLong ( ) const

Returns extra information dependent on the event objects type.

If the event comes from a listbox selection, it is a boolean determining whether the event was a selection (true) or a deselection (false). A listbox deselection only occurs for multiple-selection boxes, and in this case the index and string values are indeterminate and the listbox must be examined by the application.

int wxCommandEvent::GetInt ( ) const

Returns the integer identifier corresponding to a listbox, choice or radiobox selection (only if the event was a selection, not a deselection), or a boolean value representing the value of a checkbox.

For a menu item, this method returns -1 if the item is not checkable or a boolean value (true or false) for checkable items indicating the new state of the item.

int wxCommandEvent::GetSelection ( ) const

Returns item index for a listbox or choice selection event (not valid for a deselection).

wxString wxCommandEvent::GetString ( ) const

Returns item string for a listbox or choice selection event.

If one or several items have been deselected, returns the index of the first deselected item. If some items have been selected and others deselected at the same time, it will return the index of the first selected item.

bool wxCommandEvent::IsChecked ( ) const

This method can be used with checkbox and menu events: for the checkboxes, the method returns true for a selection event and false for a deselection one.

For the menu events, this method indicates if the menu item just has become checked or unchecked (and thus only makes sense for checkable menu items).

Notice that this method cannot be used with wxCheckListBox currently.

bool wxCommandEvent::IsSelection ( ) const

For a listbox or similar event, returns true if it is a selection, false if it is a deselection.

If some items have been selected and others deselected at the same time, it will return true.

void wxCommandEvent::SetClientData (void * *clientData*)

Sets the client data for this event.

void wxCommandEvent::SetClientObject (wxClientData * *clientObject*)

Sets the client object for this event.

The client object is not owned by the event object and the event object will not delete the client object in its destructor.

The client object must be owned and deleted by another object (e.g. a control) that has longer life time than the event object.

void wxCommandEvent::SetExtraLong (long *extraLong*)

Sets the **m_extraLong** member.

void wxCommandEvent::SetInt (int *intCommand*)

Sets the **m_commandInt** member.

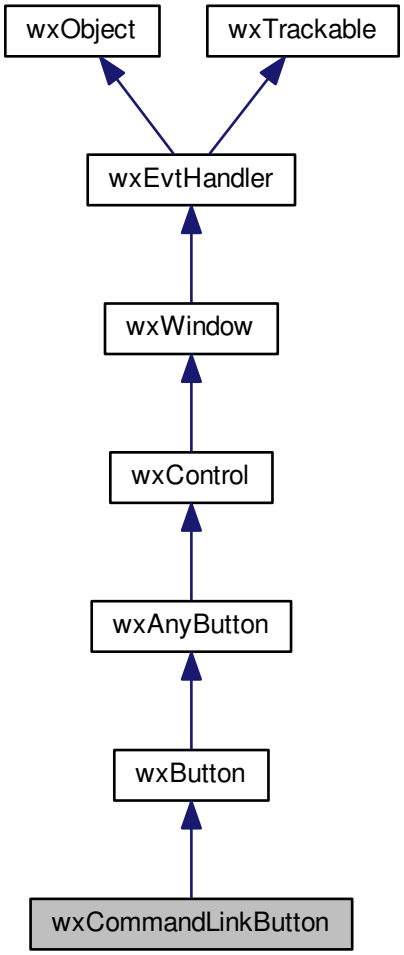
void wxCommandEvent::SetString (const wxString & *string*)

Sets the **m_commandString** member.

21.114 wxCommandLinkButton Class Reference

`#include <wx/commandlinkbutton.h>`

Inheritance diagram for wxCommandLinkButton:



21.114.1 Detailed Description

Objects of this class are similar in appearance to the normal wxButtons but are similar to the links in a web page in functionality.

Pressing such button usually results in switching to another window of the program and so they can be used as a replacement for the "Next" button in a multi-page dialog (such as wxWizard), for example.

Their advantage compared to the ordinary wxButtons is that they emphasize the action of switching the window and also that they allow to give more detailed explanation to the user because, in addition to the short button label, they also show a longer description string.

The short, title-like, part of the label is called the *main label* and the longer description is the *note*. Both of them can be set and queried independently using wxCommandLinkButton-specific methods such as SetMainLabel() or GetNote() or also via SetLabel() and GetLabel() methods inherited from wxButton. When using the latter, the main label and the note are concatenated into a single string using a new line character between them (notice that the note part can have more new lines in it).

wxCommandLinkButton generates the same event as wxButton but doesn't support any of wxButton-specific styles

nor adds any new styles of its own.

Currently this class uses native implementation under Windows Vista and later versions and a generic implementation for the other platforms and earlier Windows versions.

Since

2.9.2

Library: wxAdvanced

Category: Controls

See also

wxButton, wxBitmapButton

Public Member Functions

- wxCommandLinkButton ()

  Default constructor.
- wxCommandLinkButton (wxWindow *parent, wxWindowID id, const wxString &mainLabel=wxEmptyString, const wxString &note=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)

  Constructor really creating a command Link button.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &mainLabel=wxEmptyString, const wxString &note=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)

  Button creation function for two-step creation.
- void SetMainLabelAndNote (const wxString &mainLabel, const wxString &note)

  Sets a new main label and note for the button.
- virtual void SetLabel (const wxString &label)

  Sets the string label and note for the button.
- wxString GetLabel () const

  Returns the string label for the button.
- void SetMainLabel (const wxString &mainLabel)

  Changes the main label.
- void SetNote (const wxString &note)

  Changes the note.
- wxString GetMainLabel () const

  Returns the current main label.
- wxString GetNote () const

  Returns the currently used note.

Additional Inherited Members

21.114.2 Constructor & Destructor Documentation

wxCommandLinkButton::wxCommandLinkButton ( )

Default constructor.

Use Create() to really create the control.

wxCommandLinkButton::wxCommandLinkButton (wxWindow * *parent*, wxWindowID *id*, const wxString & *mainLabel* = wxEmptyString, const wxString & *note* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxButtonNameStr)

Constructor really creating a command Link button.

The button will be decorated with stock icons under GTK+ 2.

Parameters

parent	Parent window. Must not be NULL.
id	Button identifier. A value of wxID_ANY indicates a default value.
mainLabel	First line of text on the button, typically the label of an action that will be made when the button is pressed.
note	Second line of text describing the action performed when the button is pressed.
pos	Button position.
size	Button size. If the default size is specified then the button is sized appropriately for the text.
style	Window style. See wxButton class description.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

21.114.3 Member Function Documentation

bool wxCommandLinkButton::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *mainLabel* = wxEmptyString, const wxString & *note* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxButtonNameStr)

Button creation function for two-step creation.

For more details, see wxCommandLinkButton().

wxString wxCommandLinkButton::GetLabel ( ) const [virtual]

Returns the string label for the button.

See also

SetLabel()

Returns

A string with the main label and note concatenated together with a newline separating them.

Reimplemented from wxButton.

wxString wxCommandLinkButton::GetMainLabel ( ) const

Returns the current main label.

Returns

Main label currently displayed.

wxString wxCommandLinkButton::GetNote ( ) const

Returns the currently used note.

Returns

Note currently displayed.

virtual void wxCommandLinkButton::SetLabel (const wxString & *label*) [virtual]

Sets the string label and note for the button.

Parameters

label	The label and note to set, with the two separated by the first newline or none to set a blank note.

Reimplemented from wxButton.

void wxCommandLinkButton::SetMainLabel (const wxString & *mainLabel*)

Changes the main label.

Parameters

mainLabel	New main label to use.

void wxCommandLinkButton::SetMainLabelAndNote (const wxString & *mainLabel,* const wxString & *note*)

Sets a new main label and note for the button.

Neither of the arguments can be empty, if you need to change just the label or just the note, use SetMainLabel() or SetNote() instead of this function.

Parameters

mainLabel	New main label to use.
note	New note to use.

void wxCommandLinkButton::SetNote (const wxString & *note*)

Changes the note.

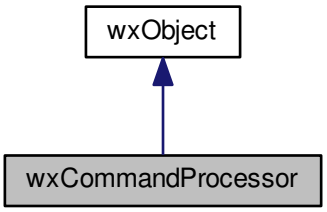
Parameters

note	New note to use.

21.115 wxCommandProcessor Class Reference

```
#include <wx/cmdproc.h>
```

Inheritance diagram for wxCommandProcessor:



21.115.1 Detailed Description

wxCommandProcessor is a class that maintains a history of wxCommands, with undo/redo functionality built-in.

Derive a new class from this if you want different behaviour.

Library: wxCore

Category: Document/View Framework

See also

wxCommandProcessor Overview, wxCommand

Public Member Functions

- wxCommandProcessor (int maxCommands=-1)

  Constructor.
- virtual ~wxCommandProcessor ()

  Destructor.
- virtual bool CanUndo () const

  Returns true if the currently-active command can be undone, false otherwise.
- virtual bool CanRedo () const

  Returns true if the currently-active command can be redone, false otherwise.
- virtual void ClearCommands ()

  Deletes all commands in the list and sets the current command pointer to NULL.
- wxList & GetCommands ()

  Returns the list of commands.
- wxCommand * GetCurrentCommand () const

  Returns the current command.
- wxMenu * GetEditMenu () const

  Returns the edit menu associated with the command processor.
- int GetMaxCommands () const

  Returns the maximum number of commands that the command processor stores.
- const wxString & GetRedoAccelerator () const

Returns the string that will be appended to the Redo menu item.

- wxString GetRedoMenuLabel () const

  Returns the string that will be shown for the redo menu item.
- const wxString & GetUndoAccelerator () const

  Returns the string that will be appended to the Undo menu item.
- wxString GetUndoMenuLabel () const

  Returns the string that will be shown for the undo menu item.
- virtual void Initialize ()

  Initializes the command processor, setting the current command to the last in the list (if any), and updating the edit menu (if one has been specified).
- virtual bool IsDirty () const

  Returns a boolean value that indicates if changes have been made since the last save operation.
- void MarkAsSaved ()

  You must call this method whenever the project is saved if you plan to use IsDirty().
- virtual bool Redo ()

  Executes (redoes) the current command (the command that has just been undone if any).
- void SetEditMenu (wxMenu *menu)

  Tells the command processor to update the Undo and Redo items on this menu as appropriate.
- virtual void SetMenuStrings ()

  Sets the menu labels according to the currently set menu and the current command state.
- void SetRedoAccelerator (const wxString &accel)

  Sets the string that will be appended to the Redo menu item.
- void SetUndoAccelerator (const wxString &accel)

  Sets the string that will be appended to the Undo menu item.
- virtual bool Submit (wxCommand *command, bool storeIt=true)

  Submits a new command to the command processor.
- virtual void Store (wxCommand *command)

  Just store the command without executing it.
- virtual bool Undo ()

  Undoes the last command executed.

Additional Inherited Members

21.115.2 Constructor & Destructor Documentation

wxCommandProcessor::wxCommandProcessor (int *maxCommands* = `-1`)

Constructor.

Parameters

maxCommands	May be set to a positive integer to limit the number of commands stored to it, otherwise (and by default) the list of commands can grow arbitrarily.

virtual wxCommandProcessor::~wxCommandProcessor ( ) `[virtual]`

Destructor.

21.115.3 Member Function Documentation

virtual bool wxCommandProcessor::CanRedo ( ) const `[virtual]`

Returns true if the currently-active command can be redone, false otherwise.

virtual bool wxCommandProcessor::CanUndo ( ) const `[virtual]`

Returns true if the currently-active command can be undone, false otherwise.

virtual void wxCommandProcessor::ClearCommands ( ) `[virtual]`

Deletes all commands in the list and sets the current command pointer to NULL.

wxList& wxCommandProcessor::GetCommands ( )

Returns the list of commands.

wxCommand* wxCommandProcessor::GetCurrentCommand ( ) const

Returns the current command.

wxMenu* wxCommandProcessor::GetEditMenu ( ) const

Returns the edit menu associated with the command processor.

int wxCommandProcessor::GetMaxCommands ( ) const

Returns the maximum number of commands that the command processor stores.

const wxString& wxCommandProcessor::GetRedoAccelerator ( ) const

Returns the string that will be appended to the Redo menu item.

wxString wxCommandProcessor::GetRedoMenuLabel ( ) const

Returns the string that will be shown for the redo menu item.

const wxString& wxCommandProcessor::GetUndoAccelerator ( ) const

Returns the string that will be appended to the Undo menu item.

wxString wxCommandProcessor::GetUndoMenuLabel ( ) const

Returns the string that will be shown for the undo menu item.

virtual void wxCommandProcessor::Initialize ( ) `[virtual]`

Initializes the command processor, setting the current command to the last in the list (if any), and updating the edit menu (if one has been specified).

virtual bool wxCommandProcessor::IsDirty ( ) const `[virtual]`

Returns a boolean value that indicates if changes have been made since the last save operation.

This only works if MarkAsSaved() is called whenever the project is saved.

void wxCommandProcessor::MarkAsSaved ( )

You must call this method whenever the project is saved if you plan to use IsDirty().

virtual bool wxCommandProcessor::Redo ( ) [virtual]

Executes (redoes) the current command (the command that has just been undone if any).

void wxCommandProcessor::SetEditMenu (wxMenu * *menu*)

Tells the command processor to update the Undo and Redo items on this menu as appropriate.

Set this to NULL if the menu is about to be destroyed and command operations may still be performed, or the command processor may try to access an invalid pointer.

virtual void wxCommandProcessor::SetMenuStrings ( ) [virtual]

Sets the menu labels according to the currently set menu and the current command state.

void wxCommandProcessor::SetRedoAccelerator (const wxString & *accel*)

Sets the string that will be appended to the Redo menu item.

void wxCommandProcessor::SetUndoAccelerator (const wxString & *accel*)

Sets the string that will be appended to the Undo menu item.

virtual void wxCommandProcessor::Store (wxCommand * *command*) [virtual]

Just store the command without executing it.

The command is stored in the history list, and the associated edit menu (if any) updated appropriately.

virtual bool wxCommandProcessor::Submit (wxCommand * *command*, bool *storeIt* = true) [virtual]

Submits a new command to the command processor.

The command processor calls wxCommand::Do() to execute the command; if it succeeds, the command is stored in the history list, and the associated edit menu (if any) updated appropriately. If it fails, the command is deleted immediately. Once Submit() has been called, the passed command should not be deleted directly by the application.

Parameters

command	The command to submit
storeIt	Indicates whether the successful command should be stored in the history list.

virtual bool wxCommandProcessor::Undo ( ) [virtual]

Undoes the last command executed.

21.116 wxCondition Class Reference

`#include <wx/thread.h>`

21.116.1 Detailed Description

wxCondition variables correspond to pthread conditions or to Win32 event objects.

They may be used in a multithreaded application to wait until the given condition becomes true which happens when the condition becomes signaled.

For example, if a worker thread is doing some long task and another thread has to wait until it is finished, the latter thread will wait on the condition object and the worker thread will signal it on exit (this example is not perfect because in this particular case it would be much better to just wxThread::Wait for the worker thread, but if there are several worker threads it already makes much more sense).

Note that a call to wxCondition::Signal may happen before the other thread calls wxCondition::Wait and, just as with the pthread conditions, the signal is then lost and so if you want to be sure that you don't miss it you must keep the mutex associated with the condition initially locked and lock it again before calling wxCondition::Signal. Of course, this means that this call is going to block until wxCondition::Wait is called by another thread.

21.116.2 Example

This example shows how a main thread may launch a worker thread which starts running and then waits until the main thread signals it to continue:

```
class MySignallingThread : public wxThread
{
public:
    MySignallingThread(wxMutex *mutex, wxCondition *condition)
    {
        m_mutex = mutex;
        m_condition = condition;
    }

    virtual ExitCode Entry()
    {
        ... do our job ...

        // tell the other(s) thread(s) that we're about to terminate: we must
        // lock the mutex first or we might signal the condition before the
        // waiting threads start waiting on it!
        wxMutexLocker lock(*m_mutex);
        m_condition->Broadcast(); // same as Signal() here -- one waiter only

        return 0;
    }

private:
    wxCondition *m_condition;
    wxMutex *m_mutex;
};

int main()
{
    wxMutex mutex;
    wxCondition condition(mutex);

    // the mutex should be initially locked
    mutex.Lock();

    // create and run the thread but notice that it won't be able to
    // exit (and signal its exit) before we unlock the mutex below
    MySignallingThread *thread = new MySignallingThread(&mutex, &condition);

    thread->Run();

    // wait for the thread termination: Wait() atomically unlocks the mutex
    // which allows the thread to continue and starts waiting
    condition.Wait();

    // now we can exit
    return 0;
}
```

Of course, here it would be much better to simply use a joinable thread and call wxThread::Wait on it, but this example does illustrate the importance of properly locking the mutex when using wxCondition.

Library: wxBase

Category: Threading

See also

wxThread, wxMutex

Public Member Functions

- wxCondition (wxMutex &mutex)
  Default and only constructor.
- ~wxCondition ()
  Destroys the wxCondition object.
- wxCondError Broadcast ()
  Broadcasts to all waiting threads, waking all of them up.
- bool IsOk () const
  Returns true if the object had been initialized successfully, false if an error occurred.
- wxCondError Signal ()
  Signals the object waking up at most one thread.
- wxCondError Wait ()
  Waits until the condition is signalled.
- template<typename Functor >
  wxCondError Wait (const Functor &predicate)
  Waits until the condition is signalled and the associated condition true.
- wxCondError WaitTimeout (unsigned long milliseconds)
  Waits until the condition is signalled or the timeout has elapsed.

21.116.3 Constructor & Destructor Documentation

wxCondition::wxCondition (wxMutex & *mutex*)

Default and only constructor.

The *mutex* must be locked by the caller before calling Wait() function. Use IsOk() to check if the object was successfully initialized.

wxCondition::~wxCondition ( )

Destroys the wxCondition object.

The destructor is not virtual so this class should not be used polymorphically.

21.116.4 Member Function Documentation

wxCondError wxCondition::Broadcast ( )

Broadcasts to all waiting threads, waking all of them up.

Note that this method may be called whether the mutex associated with this condition is locked or not.

See also

Signal()

bool wxCondition::IsOk ( ) const

Returns true if the object had been initialized successfully, false if an error occurred.

wxCondError wxCondition::Signal ( )

Signals the object waking up at most one thread.

If several threads are waiting on the same condition, the exact thread which is woken up is undefined. If no threads are waiting, the signal is lost and the condition would have to be signalled again to wake up any thread which may start waiting on it later.

Note that this method may be called whether the mutex associated with this condition is locked or not.

See also

Broadcast()

wxCondError wxCondition::Wait ( )

Waits until the condition is signalled.

This method atomically releases the lock on the mutex associated with this condition (this is why it must be locked prior to calling Wait()) and puts the thread to sleep until Signal() or Broadcast() is called. It then locks the mutex again and returns.

Note that even if Signal() had been called before Wait() without waking up any thread, the thread would still wait for another one and so it is important to ensure that the condition will be signalled after Wait() or the thread may sleep forever.

Returns

Returns wxCOND_NO_ERROR on success, another value if an error occurred.

See also

WaitTimeout()

template<typename Functor > wxCondError wxCondition::Wait (const Functor & *predicate*)

Waits until the condition is signalled and the associated condition true.

This is a convenience overload that may be used to ignore spurious awakenings while waiting for a specific condition to become true.

Equivalent to

```
while ( !predicate() )
{
    wxCondError e = Wait();
    if ( e != wxCOND_NO_ERROR )
        return e;
}
return wxCOND_NO_ERROR;
```

The predicate would typically be a C++11 lambda:

```
condvar.Wait([]{return value == 1;});
```

Since

3.0

wxCondError wxCondition::WaitTimeout (unsigned long *milliseconds*)

Waits until the condition is signalled or the timeout has elapsed.

This method is identical to Wait() except that it returns, with the return code of `wxCOND_TIMEOUT` as soon as the given timeout expires.

Parameters

milliseconds	Timeout in milliseconds

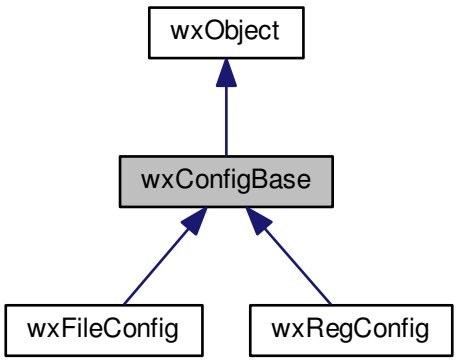
Returns

Returns wxCOND_NO_ERROR if the condition was signalled, wxCOND_TIMEOUT if the timeout elapsed before this happened or another error code from wxCondError enum.

21.117 wxConfigBase Class Reference

`#include <wx/config.h>`

Inheritance diagram for wxConfigBase:



21.117.1 Detailed Description

wxConfigBase defines the basic interface of all config classes.

It cannot be used by itself (it is an abstract base class) and you will always use one of its derivations: wxFileConfig, wxRegConfig or any other.

However, usually you don't even need to know the precise nature of the class you're working with but you would just use the wxConfigBase methods. This allows you to write the same code regardless of whether you're working with the registry under Windows or text-based config files under Unix. To make writing the portable code even easier, wxWidgets provides a typedef wxConfig which is mapped onto the native wxConfigBase implementation on the given platform: i.e. wxRegConfig under Windows and wxFileConfig otherwise.

See wxConfig Overview for a description of all features of this class.

It is highly recommended to use static functions Get() and/or Set(), so please have a look at them.

Related Include Files:

- <wx/config.h> - Let wxWidgets choose a wxConfig class for your platform.
- <wx/confbase.h> - Base config class.
- <wx/fileconf.h> - wxFileConfig class.
- <wx/msw/regconf.h> - wxRegConfig class, see also wxRegKey.

21.117.2 Example

Here is how you would typically use this class:

```
// using wxConfig instead of writing wxFileConfig or wxRegConfig enhances
// portability of the code
wxConfig *config = new wxConfig("MyAppName");

wxString str;
if ( config->Read("LastPrompt", &str) ) {
    // last prompt was found in the config file/registry and its value is
    // now in str
    // ...
}
else {
    // no last prompt...
}

// another example: using default values and the full path instead of just
// key name: if the key is not found , the value 17 is returned
long value = config->ReadLong("/LastRun/CalculatedValues/MaxValue", 17);

// at the end of the program we would save everything back
config->Write("LastPrompt", str);
config->Write("/LastRun/CalculatedValues/MaxValue", value);

// the changes will be written back automatically
delete config;
```

This basic example, of course, doesn't show all wxConfig features, such as enumerating, testing for existence and deleting the entries and groups of entries in the config file, its abilities to automatically store the default values or expand the environment variables on the fly. However, the main idea is that using this class is easy and that it should normally do what you expect it to.

Note

In the documentation of this class, the words "config file" also mean "registry hive" for wxRegConfig and, generally speaking, might mean any physical storage where a wxConfigBase-derived class stores its data.

21.117.3 Static Functions

The static functions provided deal with the "default" config object. Although its usage is not at all mandatory it may be convenient to use a global config object instead of creating and deleting the local config objects each time you need one (especially because creating a wxFileConfig object might be a time consuming operation). In this case, you may create this global config object in the very start of the program and Set() it as the default. Then, from anywhere in your program, you may access it using the Get() function. This global wxConfig object will be deleted by wxWidgets automatically if it exists. Note that this implies that if you do delete this object yourself (usually in wxApp::OnExit()) you must use Set(NULL) to prevent wxWidgets from deleting it the second time.

As it happens, you may even further simplify the procedure described above: you may forget about calling Set(). When Get() is called and there is no current object, it will create one using Create() function. To disable this behaviour DontCreateOnDemand() is provided.

Note

You should use either Set() or Get() because wxWidgets library itself would take advantage of it and could save various information in it. For example wxFontMapper or Unix version of wxFileDialog have the ability to use wxConfig class.

21.117.4 Path Management

As explained in the config overview, the config classes support a file system-like hierarchy of keys (files) and groups (directories). As in the file system case, to specify a key in the config class you must use a path to it. Config classes also support the notion of the current group, which makes it possible to use the relative paths. To clarify all this, here is an example (it is only for the sake of demonstration, it doesn't do anything sensible!):

```
wxConfig *config = new wxConfig("FooBarApp");

// right now the current path is '/'
conf->Write("RootEntry", 1);

// go to some other place: if the group(s) don't exist, they will be created
conf->SetPath("/Group/Subgroup");

// create an entry in subgroup
conf->Write("SubgroupEntry", 3);

// '..' is understood
conf->Write("../GroupEntry", 2);
conf->SetPath("..");

wxASSERT( conf->ReadLong("Subgroup/SubgroupEntry", 0) == 3 );

// use absolute path: it is allowed, too
wxASSERT( conf->ReadLong("/RootEntry", 0) == 1 );
```

It is highly recommended that you restore the path to its old value on function exit:

```
void foo(wxConfigBase *config)
{
    wxString strOldPath = config->GetPath();

    config->SetPath("/Foo/Data");
    // ...

    config->SetPath(strOldPath);
}
```

Otherwise the assert in the following example will surely fail (we suppose here that the foo() function is the same as above except that it doesn't save and restore the path):

```
void bar(wxConfigBase *config)
{
    config->Write("Test", 17);

    foo(config);

    // we're reading "/Foo/Data/Test" here! -1 will probably be returned...
    wxASSERT( config->ReadLong("Test", -1) == 17 );
}
```

Finally, the path separator in wxConfigBase and derived classes is always "/", regardless of the platform (i.e. it is not "\\" under Windows).

21.117.5 Enumeration

The enumeration functions allow you to enumerate all entries and groups in the config file. All functions here return false when there are no more items.

You must pass the same index to GetNext() and GetFirst() (don't modify it). Please note that it is not the index of the current item (you will have some great surprises with wxRegConfig if you assume this) and you shouldn't even

look at it: it is just a "cookie" which stores the state of the enumeration. It can't be stored inside the class because it would prevent you from running several enumerations simultaneously, that's why you must pass it explicitly.

Having said all this, enumerating the config entries/groups is very simple:

```
wxConfigBase *config = ...;
wxArrayString aNames;

// enumeration variables
wxString str;
long dummy;

// first enum all entries
bool bCont = config->GetFirstEntry(str, dummy);
while ( bCont ) {
    aNames.Add(str);

    bCont = config->GetNextEntry(str, dummy);
}

// ... we have all entry names in aNames...

// now all groups...
bCont = config->GetFirstGroup(str, dummy);
while ( bCont ) {
    aNames.Add(str);

    bCont = config->GetNextGroup(str, dummy);
}

// ... we have all group (and entry) names in aNames...
```

There are also functions to get the number of entries/subgroups without actually enumerating them, but you will probably never need them.

21.117.6 Key Access

The key access functions are the core of wxConfigBase class: they allow you to read and write config file data. All Read() functions take a default value which will be returned if the specified key is not found in the config file.

Currently, supported types of data are: wxString, `long`, `double`, `bool`, wxColour and any other types for which the functions wxToString() and wxFromString() are defined.

Try not to read long values into string variables and vice versa: although it just might work with wxFileConfig, you will get a system error with wxRegConfig because in the Windows registry the different types of entries are indeed used.

Final remark: the *szKey* parameter for all these functions can contain an arbitrary path (either relative or absolute), not just the key name.

Library: wxBase

Category: Application and System configuration

See also

wxConfigPathChanger

Public Types

- enum EntryType {
  Type_Unknown,
  Type_String,
  Type_Boolean,
  Type_Integer,
  Type_Float }

Public Member Functions

- wxConfigBase (const wxString &appName=wxEmptyString, const wxString &vendorName=wxEmptyString, const wxString &localFilename=wxEmptyString, const wxString &globalFilename=wxEmptyString, long style=0, const wxMBConv &conv=wxConvAuto())

  This is the default and only constructor of the wxConfigBase class, and derived classes.
- virtual ~wxConfigBase ()

  Empty but ensures that dtor of all derived classes is virtual.

Path Management

See Path Management

- virtual const wxString & GetPath () const =0

  Retrieve the current path (always as absolute path).
- virtual void SetPath (const wxString &strPath)=0

  Set current path: if the first character is '/', it is the absolute path, otherwise it is a relative path.

Enumeration

See Enumeration

- virtual bool GetFirstEntry (wxString &str, long &index) const =0

  Gets the first entry.
- virtual bool GetFirstGroup (wxString &str, long &index) const =0

  Gets the first group.
- virtual bool GetNextEntry (wxString &str, long &index) const =0

  Gets the next entry.
- virtual bool GetNextGroup (wxString &str, long &index) const =0

  Gets the next group.
- virtual size_t GetNumberOfEntries (bool bRecursive=false) const =0

  Get number of entries in the current group.
- virtual size_t GetNumberOfGroups (bool bRecursive=false) const =0

  Get number of entries/subgroups in the current group, with or without its subgroups.

Tests of Existence

- bool Exists (const wxString &strName) const
- virtual wxConfigBase::EntryType GetEntryType (const wxString &name) const

  Returns the type of the given entry or Unknown if the entry doesn't exist.
- virtual bool HasEntry (const wxString &strName) const =0
- virtual bool HasGroup (const wxString &strName) const =0

Miscellaneous Functions

- wxString GetAppName () const

  Returns the application name.
- wxString GetVendorName () const

  Returns the vendor name.

Key Access

See Key Access

- virtual bool Flush (bool bCurrentOnly=false)=0

  Permanently writes all changes (otherwise, they're only written from object's destructor).
- bool Read (const wxString &key, wxString *str) const

  Read a string from the key, returning true if the value was read.
- bool Read (const wxString &key, wxString *str, const wxString &defaultVal) const

  Read a string from the key.

- const wxString Read (const wxString &key, const wxString &defaultVal) const
  Another version of Read(), returning the string value directly.
- bool Read (const wxString &key, long *l) const
  Reads a long value, returning true if the value was found.
- bool Read (const wxString &key, long *l, long defaultVal) const
  Reads a long value, returning true if the value was found.
- bool Read (const wxString &key, double *d) const
  Reads a double value, returning true if the value was found.
- bool Read (const wxString &key, double *d, double defaultVal) const
  Reads a double value, returning true if the value was found.
- bool Read (const wxString &key, float *f) const
  Reads a float value, returning true if the value was found.
- bool Read (const wxString &key, float *f, float defaultVal) const
  Reads a float value, returning true if the value was found.
- bool Read (const wxString &key, bool *b) const
  Reads a boolean value, returning true if the value was found.
- bool Read (const wxString &key, bool *d, bool defaultVal) const
  Reads a boolean value, returning true if the value was found.
- bool Read (const wxString &key, wxMemoryBuffer *buf) const
  Reads a binary block, returning true if the value was found.
- bool Read (const wxString &key, T *value) const
  Reads a value of type T, for which function wxFromString() is defined, returning true if the value was found.
- bool Read (const wxString &key, T *value, const T &defaultVal) const
  Reads a value of type T, for which function wxFromString() is defined, returning true if the value was found.
- bool ReadBool (const wxString &key, bool defaultVal) const
  Reads a bool value from the key and returns it.
- double ReadDouble (const wxString &key, double defaultVal) const
  Reads a double value from the key and returns it.
- long ReadLong (const wxString &key, long defaultVal) const
  Reads a long value from the key and returns it.
- T ReadObject (const wxString &key, T const &defaultVal) const
  Reads a value of type T (for which the function wxFromString() must be defined) from the key and returns it.
- bool Write (const wxString &key, const wxString &value)
  Writes the wxString value to the config file and returns true on success.
- bool Write (const wxString &key, long value)
  Writes the long value to the config file and returns true on success.
- bool Write (const wxString &key, double value)
  Writes the double value to the config file and returns true on success.
- bool Write (const wxString &key, bool value)
  Writes the bool value to the config file and returns true on success.
- bool Write (const wxString &key, const wxMemoryBuffer &buf)
  Writes the wxMemoryBuffer value to the config file and returns true on success.
- bool Write (const wxString &key, T const &buf)
  Writes the specified value to the config file and returns true on success.

Rename Entries/Groups

These functions allow renaming entries or subgroups of the current group.

They will return false on error, typically because either the entry/group with the original name doesn't exist, because the entry/group with the new name already exists or because the function is not supported in this wxConfig implementation.

- virtual bool RenameEntry (const wxString &oldName, const wxString &newName)=0
  Renames an entry in the current group.
- virtual bool RenameGroup (const wxString &oldName, const wxString &newName)=0
  Renames a subgroup of the current group.

Delete Entries/Groups

These functions delete entries and/or groups of entries from the config file.

DeleteAll() is especially useful if you want to erase all traces of your program presence: for example, when you uninstall it.

- virtual bool DeleteAll ()=0
  Delete the whole underlying object (disk file, registry key, ...).
- virtual bool DeleteEntry (const wxString &key, bool bDeleteGroupIfEmpty=true)=0
  Deletes the specified entry and the group it belongs to if it was the last key in it and the second parameter is true.
- virtual bool DeleteGroup (const wxString &key)=0
  Delete the group (with all subgroups).

Options

Some aspects of wxConfigBase behaviour can be changed during run-time.

The first of them is the expansion of environment variables in the string values read from the config file: for example, if you have the following in your config file:

```
# config file for my program
UserData = $HOME/data

# the following syntax is valid only under Windows
UserData = %windir%\\data.dat
```

The call to Read("UserData") will return something like "`/home/zeitlin/data`" on linux for example.

Although this feature is very useful, it may be annoying if you read a value which contains '$' or '' symbols (% is used for environment variables expansion under Windows) which are not used for environment variable expansion. In this situation you may call SetExpandEnvVars(false) just before reading this value and SetExpandEnvVars(true) just after. Another solution would be to prefix the offending symbols with a backslash.

- bool IsExpandingEnvVars () const
  Returns true if we are expanding environment variables in key values.
- bool IsRecordingDefaults () const
  Returns true if we are writing defaults back to the config file.
- void SetExpandEnvVars (bool bDoIt=true)
  Determine whether we wish to expand environment variables in key values.
- void SetRecordDefaults (bool bDoIt=true)
  Sets whether defaults are recorded to the config file whenever an attempt to read the value which is not present in it is done.

Static Public Member Functions

- static wxConfigBase * Create ()
  Create a new config object and sets it as the current one.
- static void DontCreateOnDemand ()
  Calling this function will prevent Get() from automatically creating a new config object if the current one is NULL.
- static wxConfigBase * Get (bool CreateOnDemand=true)
  Get the current config object.
- static wxConfigBase * Set (wxConfigBase *pConfig)
  Sets the config object as the current one, returns the pointer to the previous current object (both the parameter and returned value may be NULL).

Additional Inherited Members

21.117.7 Member Enumeration Documentation

enum wxConfigBase::EntryType

Enumerator

Type_Unknown

Type_String

Type_Boolean

Type_Integer

Type_Float

21.117.8 Constructor & Destructor Documentation

wxConfigBase::wxConfigBase (const wxString & *appName* = wxEmptyString, const wxString & *vendorName* = wxEmptyString, const wxString & *localFilename* = wxEmptyString, const wxString & *globalFilename* = wxEmptyString, long *style* = 0, const wxMBConv & *conv* = wxConvAuto())

This is the default and only constructor of the wxConfigBase class, and derived classes.

Parameters

appName	The application name. If this is empty, the class will normally use wxApp::GetAppName() to set it. The application name is used in the registry key on Windows, and can be used to deduce the local filename parameter if that is missing.
vendorName	The vendor name. If this is empty, it is assumed that no vendor name is wanted, if this is optional for the current config class. The vendor name is appended to the application name for wxRegConfig.
localFilename	Some config classes require a local filename. If this is not present, but required, the application name will be used instead.
globalFilename	Some config classes require a global filename. If this is not present, but required, the application name will be used instead.
style	Can be one of wxCONFIG_USE_LOCAL_FILE and wxCONFIG_USE_GLOBAL_FILE. The style interpretation depends on the config class and is ignored by some implementations. For wxFileConfig, these styles determine whether a local or global config file is created or used: if wxCONFIG_USE_GLOBAL_FILE is used, then settings are read from the global config file and if wxCONFIG_USE_LOCAL_FILE is used, settings are read from and written to local config file (if they are both set, global file is read first, then local file, overwriting global settings). If the flag is present but the parameter is empty, the parameter will be set to a default. If the parameter is present but the style flag not, the relevant flag will be added to the style. For wxRegConfig, the GLOBAL flag refers to the HKLM key while LOCAL one is for the usual HKCU one. For wxFileConfig you can also add wxCONFIG_USE_RELATIVE_PATH by logically or'ing it to either of the _FILE options to tell wxFileConfig to use relative instead of absolute paths. On non-VMS Unix systems, the default local configuration file is "~/.appname". However, this path may be also used as user data directory (see wxStandardPaths::GetUserDataDir()) if the application has several data files. In this case wxCONFIG_USE_SUBDIR flag, which changes the default local configuration file to "~/.appname/appname" should be used. Notice that this flag is ignored if *localFilename* is provided. wxCONFIG_USE_SUBDIR is new since wxWidgets version 2.8.2. For wxFileConfig, you can also add wxCONFIG_USE_NO_ESCAPE_CHARACTERS which will turn off character escaping for the values of entries stored in the config file: for example a foo key with some backslash characters will be stored as "foo=C:\mydir" instead of the usual storage of "foo=C:\\mydir". The wxCONFIG_USE_NO_ESCAPE_CHARACTERS style can be helpful if your config file must be read or written to by a non-wxWidgets program (which might not understand the escape characters). Note, however, that if wxCONFIG_USE_NO_ESCAPE_CHARACTERS style is used, it is now your application's responsibility to ensure that there is no newline or other illegal characters in a value, before writing that value to the file.

conv	This parameter is only used by wxFileConfig when compiled in Unicode mode. It specifies the encoding in which the configuration file is written.

Remarks

By default, environment variable expansion is on and recording defaults is off.

virtual wxConfigBase::~wxConfigBase ( ) [virtual]

Empty but ensures that dtor of all derived classes is virtual.

21.117.9 Member Function Documentation

static wxConfigBase* wxConfigBase::Create ( ) [static]

Create a new config object and sets it as the current one.

This function will create the most appropriate implementation of wxConfig available for the current platform. By default this means that the system registry will be used for storing the configuration information under MSW and a file under the user home directory (see wxStandardPaths::GetUserConfigDir()) elsewhere.

If you prefer to use the configuration files everywhere, you can define `wxUSE_CONFIG_NATIVE` to 0 when compiling wxWidgets. Or you can simply always create wxFileConfig explicitly.

Finally, if you want to create a custom wxConfig subclass you may change this function behaviour by overriding wxAppTraits::CreateConfig() to create it. An example when this could be useful could be an application which could be installed either normally (in which case the default behaviour of using wxRegConfig is appropriate) or in a "portable" way in which case a wxFileConfig with a file in the program directory would be used and the choice would be done in CreateConfig() at run-time.

virtual bool wxConfigBase::DeleteAll ( ) [pure virtual]

Delete the whole underlying object (disk file, registry key, ...).

Primarily for use by uninstallation routine.

Implemented in wxFileConfig.

virtual bool wxConfigBase::DeleteEntry (const wxString & *key*, bool *bDeleteGroupIfEmpty* = `true`) [pure virtual]

Deletes the specified entry and the group it belongs to if it was the last key in it and the second parameter is true.

Implemented in wxFileConfig.

virtual bool wxConfigBase::DeleteGroup (const wxString & *key*) [pure virtual]

Delete the group (with all subgroups).

If the current path is under the group being deleted it is changed to its deepest still existing component. E.g. if the current path is `"/A/B/C/D"` and the group `C` is deleted, the path becomes `"/A/B"`.

Implemented in wxFileConfig.

static void wxConfigBase::DontCreateOnDemand ( ) [static]

Calling this function will prevent *Get()* from automatically creating a new config object if the current one is NULL.

It might be useful to call it near the program end to prevent "accidental" creation of a new config object.

bool wxConfigBase::Exists (const wxString & *strName*) const

Returns

true if either a group or an entry with a given name exists.

virtual bool wxConfigBase::Flush (bool *bCurrentOnly* = `false`) [pure virtual]

Permanently writes all changes (otherwise, they're only written from object's destructor).

Implemented in wxFileConfig.

static wxConfigBase* wxConfigBase::Get (bool *CreateOnDemand* = `true`) [static]

Get the current config object.

If there is no current object and *CreateOnDemand* is true, this creates one (using Create()) unless DontCreateOn↩Demand() was called previously.

wxString wxConfigBase::GetAppName ( ) const

Returns the application name.

virtual wxConfigBase::EntryType wxConfigBase::GetEntryType (const wxString & *name*) const [virtual]

Returns the type of the given entry or *Unknown* if the entry doesn't exist.

This function should be used to decide which version of Read() should be used because some of wxConfig implementations will complain about type mismatch otherwise: e.g., an attempt to read a string value from an integer key with wxRegConfig will fail.

virtual bool wxConfigBase::GetFirstEntry (wxString & *str,* long & *index*) const [pure virtual]

Gets the first entry.

wxPerl Note: In wxPerl this method takes no parameters and returns a 3-element list (continue_flag, string, index↩_for_getnextentry).

Implemented in wxFileConfig.

virtual bool wxConfigBase::GetFirstGroup (wxString & *str,* long & *index*) const [pure virtual]

Gets the first group.

wxPerl Note: In wxPerl this method takes no parameters and returns a 3-element list (continue_flag, string, index↩_for_getnextentry).

Implemented in wxFileConfig.

virtual bool wxConfigBase::GetNextEntry (wxString & *str,* long & *index*) const [pure virtual]

Gets the next entry.

wxPerl Note: In wxPerl this method only takes the *index* parameter and returns a 3-element list (continue_flag, string, index_for_getnextentry).

Implemented in wxFileConfig.

virtual bool wxConfigBase::GetNextGroup (wxString & *str,* long & *index*) const `[pure virtual]`

Gets the next group.

wxPerl Note: In wxPerl this method only takes the *index* parameter and returns a 3-element list (continue_flag, string, index_for_getnextentry).

Implemented in wxFileConfig.

virtual size_t wxConfigBase::GetNumberOfEntries (bool *bRecursive* = `false`) const `[pure virtual]`

Get number of entries in the current group.

Implemented in wxFileConfig.

virtual size_t wxConfigBase::GetNumberOfGroups (bool *bRecursive* = `false`) const `[pure virtual]`

Get number of entries/subgroups in the current group, with or without its subgroups.

Implemented in wxFileConfig.

virtual const wxString& wxConfigBase::GetPath ( ) const `[pure virtual]`

Retrieve the current path (always as absolute path).

Implemented in wxFileConfig.

wxString wxConfigBase::GetVendorName ( ) const

Returns the vendor name.

virtual bool wxConfigBase::HasEntry (const wxString & *strName*) const `[pure virtual]`

Returns

true if the entry by this name exists.

Implemented in wxFileConfig.

virtual bool wxConfigBase::HasGroup (const wxString & *strName*) const `[pure virtual]`

Returns

true if the group by this name exists.

Implemented in wxFileConfig.

bool wxConfigBase::IsExpandingEnvVars ( ) const

Returns true if we are expanding environment variables in key values.

bool wxConfigBase::IsRecordingDefaults ( ) const

Returns true if we are writing defaults back to the config file.

bool wxConfigBase::Read (const wxString & *key,* wxString * *str*) const

Read a string from the key, returning true if the value was read.

If the key was not found, *str* is not changed.

wxPerl Note: Not supported by wxPerl.

bool wxConfigBase::Read (const wxString & *key,* wxString * *str,* const wxString & *defaultVal*) const

Read a string from the key.

The default value is returned if the key was not found.

Returns

true if value was really read, false if the default was used.

wxPerl Note: Not supported by wxPerl.

const wxString wxConfigBase::Read (const wxString & *key,* const wxString & *defaultVal*) const

Another version of Read(), returning the string value directly.

wxPerl Note: In wxPerl, this can be called as:

- Read(key): returns the empty string if no key is found
- Read(key, default): returns the default value if no key is found

bool wxConfigBase::Read (const wxString & *key,* long * *l*) const

Reads a long value, returning true if the value was found.

If the value was not found, *l* is not changed.

wxPerl Note: Not supported by wxPerl.

bool wxConfigBase::Read (const wxString & *key,* long * *l,* long *defaultVal*) const

Reads a long value, returning true if the value was found.

If the value was not found, *defaultVal* is used instead.

wxPerl Note: In wxPerl, this can be called as:

- ReadInt(key): returns the 0 if no key is found
- ReadInt(key, default): returns the default value if no key is found

bool wxConfigBase::Read (const wxString & *key,* double * *d*) const

Reads a double value, returning true if the value was found.

If the value was not found, *d* is not changed.

wxPerl Note: Not supported by wxPerl.

bool wxConfigBase::Read (const wxString & *key,* double * *d,* double *defaultVal*) const

Reads a double value, returning true if the value was found.

If the value was not found, *defaultVal* is used instead.

wxPerl Note: In wxPerl, this can be called as:

- ReadFloat(key): returns the 0.0 if no key is found
- ReadFloat(key, default): returns the default value if no key is found

bool wxConfigBase::Read (const wxString & *key,* float * *f*) const

Reads a float value, returning true if the value was found.

If the value was not found, *f* is not changed.

Notice that the value is read as a double but must be in a valid range for floats for the function to return true.

Since

2.9.1

wxPerl Note: Not supported by wxPerl.

bool wxConfigBase::Read (const wxString & *key,* float * *f,* float *defaultVal*) const

Reads a float value, returning true if the value was found.

If the value was not found, *defaultVal* is used instead.

Notice that the value is read as a double but must be in a valid range for floats for the function to return true.

Since

2.9.1

wxPerl Note: Not supported by wxPerl.

bool wxConfigBase::Read (const wxString & *key,* bool * *b*) const

Reads a boolean value, returning true if the value was found.

If the value was not found, *b* is not changed.

Since

2.9.1

wxPerl Note: Not supported by wxPerl.

bool wxConfigBase::Read (const wxString & *key,* bool * *d,* bool *defaultVal*) const

Reads a boolean value, returning true if the value was found.

If the value was not found, *defaultVal* is used instead.

wxPerl Note: In wxPerl, this can be called as:

- ReadBool(key): returns false if no key is found
- ReadBool(key, default): returns the default value if no key is found

bool wxConfigBase::Read (const wxString & *key,* wxMemoryBuffer * *buf*) const

Reads a binary block, returning true if the value was found.

If the value was not found, *buf* is not changed.

bool wxConfigBase::Read (const wxString & *key,* T * *value*) const

Reads a value of type T, for which function wxFromString() is defined, returning true if the value was found.

If the value was not found, *value* is not changed.

bool wxConfigBase::Read (const wxString & *key,* T * *value,* const T & *defaultVal*) const

Reads a value of type T, for which function wxFromString() is defined, returning true if the value was found.

If the value was not found, *defaultVal* is used instead.

bool wxConfigBase::ReadBool (const wxString & *key,* bool *defaultVal*) const

Reads a bool value from the key and returns it.

defaultVal is returned if the key is not found.

double wxConfigBase::ReadDouble (const wxString & *key,* double *defaultVal*) const

Reads a double value from the key and returns it.

defaultVal is returned if the key is not found.

long wxConfigBase::ReadLong (const wxString & *key,* long *defaultVal*) const

Reads a long value from the key and returns it.

defaultVal is returned if the key is not found.

T wxConfigBase::ReadObject (const wxString & *key,* T const & *defaultVal*) const

Reads a value of type T (for which the function wxFromString() must be defined) from the key and returns it.

defaultVal is returned if the key is not found.

virtual bool wxConfigBase::RenameEntry (const wxString & *oldName,* const wxString & *newName*) [pure virtual]

Renames an entry in the current group.

The entries names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns

false if *oldName* doesn't exist or if *newName* already exists.

Implemented in wxFileConfig.

virtual bool wxConfigBase::RenameGroup (const wxString & *oldName,* const wxString & *newName*) [pure virtual]

Renames a subgroup of the current group.

The subgroup names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns

false if *oldName* doesn't exist or if *newName* already exists.

Implemented in wxFileConfig.

static wxConfigBase* wxConfigBase::Set (wxConfigBase * *pConfig*) [static]

Sets the config object as the current one, returns the pointer to the previous current object (both the parameter and returned value may be NULL).

void wxConfigBase::SetExpandEnvVars (bool *bDoIt* = true)

Determine whether we wish to expand environment variables in key values.

virtual void wxConfigBase::SetPath (const wxString & *strPath*) [pure virtual]

Set current path: if the first character is '/', it is the absolute path, otherwise it is a relative path.

'..' is supported. If *strPath* doesn't exist, it is created.

See also

wxConfigPathChanger

Implemented in wxFileConfig.

void wxConfigBase::SetRecordDefaults (bool *bDoIt* = true)

Sets whether defaults are recorded to the config file whenever an attempt to read the value which is not present in it is done.

If on (default is off) all default values for the settings used by the program are written back to the config file. This allows the user to see what config options may be changed and is probably useful only for wxFileConfig.

bool wxConfigBase::Write (const wxString & *key,* const wxString & *value*)

Writes the wxString value to the config file and returns true on success.

bool wxConfigBase::Write (const wxString & *key,* long *value*)

Writes the long value to the config file and returns true on success.

bool wxConfigBase::Write (const wxString & *key*, double *value*)

Writes the double value to the config file and returns true on success.

Notice that if floating point numbers are saved as strings (as is the case with the configuration files used by wxFileConfig), this function uses the C locale for writing out the number, i.e. it will always use a period as the decimal separator, irrespectively of the current locale. This behaviour is new since wxWidgets 2.9.1 as the current locale was used before, but the change should be transparent because both C and current locales are tried when reading the numbers back.

bool wxConfigBase::Write (const wxString & *key*, bool *value*)

Writes the bool value to the config file and returns true on success.

bool wxConfigBase::Write (const wxString & *key*, const wxMemoryBuffer & *buf*)

Writes the wxMemoryBuffer value to the config file and returns true on success.

bool wxConfigBase::Write (const wxString & *key*, T const & *buf*)

Writes the specified value to the config file and returns true on success.

The function wxToString() must be defined for type *T*.

21.118 wxConfigPathChanger Class Reference

`#include <wx/config.h>`

21.118.1 Detailed Description

A handy little class which changes the current path in a wxConfig object and restores it in dtor.

Declaring a local variable of this type, it's possible to work in a specific directory and ensure that the path is automatically restored when the function returns.

For example:

```
// this function loads somes settings from the given wxConfig object;
// the path selected inside it is left unchanged
bool LoadMySettings(wxConfigBase* cfg)
{
    wxConfigPathChanger changer(cfg, "/Foo/Data/SomeString");
    wxString str;
    if ( !config->Read("SomeString", &str) ) {
        wxLogError("Couldn't read SomeString!");
        return false;
            // NOTE: without wxConfigPathChanger it would be easy to forget to
            //       set the old path back into the wxConfig object before this return!
    }

    // do something useful with SomeString...

    return true;    // again: wxConfigPathChanger dtor will restore the original wxConfig path
}
```

Library: wxBase

Category: Application and System configuration

Public Member Functions

- wxConfigPathChanger (const wxConfigBase *pContainer, const wxString &strEntry)

  Changes the path of the given wxConfigBase object so that the key strEntry is accessible (for read or write).

- ~wxConfigPathChanger ()

  Restores the path selected, inside the wxConfig object passed to the ctor, to the path which was selected when the wxConfigPathChanger ctor was called.

- const wxString & Name () const

  Returns the name of the key which was passed to the ctor.

- void UpdateIfDeleted ()

  This method must be called if the original path inside the wxConfig object (i.e.

21.118.2 Constructor & Destructor Documentation

wxConfigPathChanger::wxConfigPathChanger (const wxConfigBase * *pContainer,* const wxString & *strEntry*)

Changes the path of the given wxConfigBase object so that the key *strEntry* is accessible (for read or write).

In other words, the ctor uses wxConfigBase::SetPath() with everything which precedes the last slash of *strEntry*, so that:

```
wxConfigPathChanger(wxConfigBase::Get(), "/MyProgram/SomeKeyName");
```

has the same effect of:

```
wxConfigPathChanger(wxConfigBase::Get(), "/MyProgram/");
```

wxConfigPathChanger::~wxConfigPathChanger ( )

Restores the path selected, inside the wxConfig object passed to the ctor, to the path which was selected when the wxConfigPathChanger ctor was called.

21.118.3 Member Function Documentation

const wxString& wxConfigPathChanger::Name ( ) const

Returns the name of the key which was passed to the ctor.

The "name" is just anything which follows the last slash of the string given to the ctor.

void wxConfigPathChanger::UpdateIfDeleted ( )

This method must be called if the original path inside the wxConfig object (i.e.

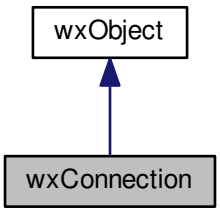
the current path at the moment of creation of this wxConfigPathChanger object) could have been deleted, thus preventing wxConfigPathChanger from restoring the not existing (any more) path.

If the original path doesn't exist any more, the path will be restored to the deepest still existing component of the old path.

21.119 wxConnection Class Reference

`#include <wx/ipc.h>`

Inheritance diagram for wxConnection:



21.119.1 Detailed Description

A wxConnection object represents the connection between a client and a server.

It is created by making a connection using a wxClient object, or by the acceptance of a connection by a wxServer object.

The bulk of a DDE-like (Dynamic Data Exchange) conversation is controlled by calling members in a **wxConnection** object or by overriding its members. The actual DDE-based implementation using wxDDEConnection is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPConnection, which has the same API.

An application should normally derive a new connection class from wxConnection, in order to override the communication event handlers to do something interesting.

Library: wxBase

Category: Interprocess Communication

See also

wxClient, wxServer, Interprocess Communication

Public Member Functions

- bool Disconnect ()

  Called by the client or server application to disconnect from the other program; it causes the OnDisconnect() message to be sent to the corresponding connection object in the other program.
- virtual bool OnAdvise (const wxString &topic, const wxString &item, const void *data, size_t size, wxIPC↩ Format format)

  Message sent to the client application when the server notifies it of a change in the data associated with the given item, using Advise().
- virtual bool OnDisconnect ()

  Message sent to the client or server application when the other application notifies it to end the connection.
- virtual bool OnExec (const wxString &topic, const wxString &data)

Message sent to the server application when the client notifies it to execute the given data, using Execute().

- virtual bool OnPoke (const wxString &topic, const wxString &item, const void ∗data, size_t size, wxIPCFormat format)

  Message sent to the server application when the client notifies it to accept the given data.
- virtual const void ∗ OnRequest (const wxString &topic, const wxString &item, size_t ∗size, wxIPCFormat format)

  Message sent to the server application when the client calls Request().
- virtual bool OnStartAdvise (const wxString &topic, const wxString &item)

  Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item.
- virtual bool OnStopAdvise (const wxString &topic, const wxString &item)

  Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item.
- const void ∗ Request (const wxString &item, size_t ∗size, wxIPCFormat format=wxIPC_TEXT)

  Called by the client application to request data from the server.
- bool StartAdvise (const wxString &item)

  Called by the client application to ask if an advise loop can be started with the server.
- bool StopAdvise (const wxString &item)

  Called by the client application to ask if an advise loop can be stopped.

- wxConnection ()

  Constructs a connection object.
- wxConnection (void ∗buffer, size_t size)

  Constructs a connection object.

- bool Advise (const wxString &item, const void ∗data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const char ∗data)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const wchar_t ∗data)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const wxString data)

  Called by the server application to advise the client of a change in the data associated with the given item.

- bool Execute (const void ∗data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the client application to execute a command on the server.
- bool Execute (const char ∗data)

  Called by the client application to execute a command on the server.
- bool Execute (const wchar_t ∗data)

  Called by the client application to execute a command on the server.
- bool Execute (const wxString data)

  Called by the client application to execute a command on the server.

- bool Poke (const wxString &item, const void ∗data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const char ∗data)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const wchar_t ∗data)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const wxString data)

  Called by the client application to poke data into the server.

Static Public Member Functions

- static bool IsTextFormat (wxIPCFormat format)

  Returns true if the format is one of the text formats.
- static wxString GetTextFromData (const void ∗data, size_t size, wxIPCFormat format)

  Returns the data in any of the text formats as string.

Additional Inherited Members

21.119.2 Constructor & Destructor Documentation

wxConnection::wxConnection ( )

Constructs a connection object.

If no user-defined connection object is to be derived from wxConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection.

However, if the user defines his or her own derived connection object, the wxServer::OnAcceptConnection and/or wxClient::OnMakeConnection members should be replaced by functions which construct the new connection object.

If the arguments of the wxConnection constructor are void then the wxConnection object manages its own connection buffer, allocating memory as needed. A programmer-supplied buffer cannot be increased if necessary, and the program will assert if it is not large enough.

The programmer-supplied buffer is included mainly for backwards compatibility.

wxConnection::wxConnection (void ∗ *buffer,* size_t *size*)

Constructs a connection object.

If no user-defined connection object is to be derived from wxConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection.

However, if the user defines his or her own derived connection object, the wxServer::OnAcceptConnection and/or wxClient::OnMakeConnection members should be replaced by functions which construct the new connection object.

If the arguments of the wxConnection constructor are void then the wxConnection object manages its own connection buffer, allocating memory as needed. A programmer-supplied buffer cannot be increased if necessary, and the program will assert if it is not large enough.

The programmer-supplied buffer is included mainly for backwards compatibility.

21.119.3 Member Function Documentation

bool wxConnection::Advise (const wxString & *item,* const void ∗ *data,* size_t *size,* wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxConnection::Advise (const wxString & *item,* const char ∗ *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxConnection::Advise (const wxString & *item,* const wchar_t * *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxConnection::Advise (const wxString & *item,* const wxString *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxConnection::Disconnect ( )

Called by the client or server application to disconnect from the other program; it causes the OnDisconnect() message to be sent to the corresponding connection object in the other program.

Returns true if successful or already disconnected. The application that calls Disconnect() must explicitly delete its side of the connection.

bool wxConnection::Execute (const void * *data,* size_t *size,* wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExec() member to be called. Returns true if successful.

bool wxConnection::Execute (const char * *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExec() member to be called. Returns true if successful.

bool wxConnection::Execute (const wchar_t * *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExec() member to be called. Returns true if successful.

bool wxConnection::Execute (const wxString *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExec() member to be called. Returns true if successful.

static wxString wxConnection::GetTextFromData (const void * *data*, size_t *size*, wxIPCFormat *format*) [static]

Returns the data in any of the text formats as string.

Parameters

data	The raw data pointer as used with any of the other methods of this class.
size	The size of the data buffer pointed to by *data*.
format	The format of the data. It must be a text one, i.e. such that IsTextFormat() returns true for it.

Returns

The string representation of the data. If the format is not text, an assertion failure is triggered and empty string is returned.

static bool wxConnection::IsTextFormat (wxIPCFormat *format*) [static]

Returns true if the format is one of the text formats.

The text formats are wxIPC_TEXT, wxIPC_UNICODETEXT and wxIPC_UTF8TEXT.

virtual bool wxConnection::OnAdvise (const wxString & *topic*, const wxString & *item*, const void * *data*, size_t *size*, wxIPCFormat *format*) [virtual]

Message sent to the client application when the server notifies it of a change in the data associated with the given item, using Advise().

virtual bool wxConnection::OnDisconnect ( ) [virtual]

Message sent to the client or server application when the other application notifies it to end the connection.

The default behaviour is to delete the connection object and return true, so applications should generally override OnDisconnect() (finally calling the inherited method as well) so that they know the connection object is no longer available.

virtual bool wxConnection::OnExec (const wxString & *topic*, const wxString & *data*) [virtual]

Message sent to the server application when the client notifies it to execute the given data, using Execute().

Note that there is no item associated with this message.

virtual bool wxConnection::OnPoke (const wxString & *topic*, const wxString & *item*, const void * *data*, size_t *size*, wxIPCFormat *format*) [virtual]

Message sent to the server application when the client notifies it to accept the given data.

virtual const void* wxConnection::OnRequest (const wxString & *topic,* const wxString & *item,* size_t * *size,* wxIPCFormat *format*) [virtual]

Message sent to the server application when the client calls Request().

The server's OnRequest() method should respond by returning a character string, or NULL to indicate no data, and setting *size.

The character string must of course persist after the call returns.

virtual bool wxConnection::OnStartAdvise (const wxString & *topic,* const wxString & *item*) [virtual]

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item.

The server can refuse to participate by returning false.

virtual bool wxConnection::OnStopAdvise (const wxString & *topic,* const wxString & *item*) [virtual]

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item.

The server can refuse to stop the advise loop by returning false, although this doesn't have much meaning in practice.

bool wxConnection::Poke (const wxString & *item,* const void * *data,* size_t *size,* wxIPCFormat *format =* wxIPC_PRIVATE)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. If size is -1 the size is computed from the string length of data.

Returns true if successful.

bool wxConnection::Poke (const wxString & *item,* const char * *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. If size is -1 the size is computed from the string length of data.

Returns true if successful.

bool wxConnection::Poke (const wxString & *item,* const wchar_t * *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. If size is -1 the size is computed from the string length of data.

Returns true if successful.

bool wxConnection::Poke (const wxString & *item,* const wxString *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. If size is -1 the size is computed from the string length of data.

Returns true if successful.

const void* wxConnection::Request (const wxString & *item*, size_t * *size*, wxIPCFormat *format* = wxIPC_TEXT)

Called by the client application to request data from the server.

Causes the server connection's OnRequest() member to be called. Size may be NULL or a pointer to a variable to receive the size of the requested item.

Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise. This buffer does not need to be deleted.

bool wxConnection::StartAdvise (const wxString & *item*)

Called by the client application to ask if an advise loop can be started with the server.

Causes the server connection's OnStartAdvise() member to be called. Returns true if the server okays it, false otherwise.

bool wxConnection::StopAdvise (const wxString & *item*)

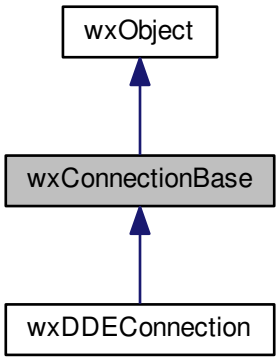
Called by the client application to ask if an advise loop can be stopped.

Causes the server connection's OnStopAdvise() member to be called. Returns true if the server okays it, false otherwise.

21.120 wxConnectionBase Class Reference

`#include <wx/ipcbase.h>`

Inheritance diagram for wxConnectionBase:



21.120.1 Detailed Description

Todo Document this class.

This class provides base, common functionality shared between wxDDEConnection, and wxTCPConnection.

Library: wxBase

Category: Interprocess Communication

See also

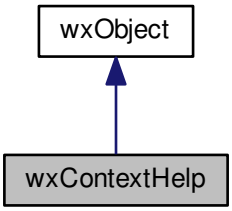
wxDDEConnection, wxTCPConnection

Additional Inherited Members

21.121 wxContextHelp Class Reference

`#include <wx/cshelp.h>`

Inheritance diagram for wxContextHelp:



21.121.1 Detailed Description

This class changes the cursor to a query and puts the application into a 'context-sensitive help mode'.

When the user left-clicks on a window within the specified window, a `wxEVT_HELP` event is sent to that control, and the application may respond to it by popping up some help.

For example:

```
wxContextHelp contextHelp(myWindow);
```

There are a couple of ways to invoke this behaviour implicitly:

- Use the wxDIALOG_EX_CONTEXTHELP style for a dialog (Windows only). This will put a question mark in the titlebar, and Windows will put the application into context-sensitive help mode automatically, with further programming.
- Create a wxContextHelpButton, whose predefined behaviour is to create a context help object. Normally you will write your application so that this button is only added to a dialog for non-Windows platforms (use wxDIALOG_EX_CONTEXTHELP on Windows).

Note that on Mac OS X, the cursor does not change when in context-sensitive help mode.

Library: wxCore

Category: Help

See also

wxHelpEvent, wxHelpController, wxContextHelpButton

Public Member Functions

- wxContextHelp (wxWindow *window=NULL, bool doNow=true)

  Constructs a context help object, calling BeginContextHelp() if doNow is true (the default).
- virtual ~wxContextHelp ()

  Destroys the context help object.
- bool BeginContextHelp (wxWindow *window)

  Puts the application into context-sensitive help mode.
- bool EndContextHelp ()

  Ends context-sensitive help mode.

Additional Inherited Members

21.121.2 Constructor & Destructor Documentation

wxContextHelp::wxContextHelp (wxWindow * *window* = `NULL`, bool *doNow* = `true`)

Constructs a context help object, calling BeginContextHelp() if *doNow* is true (the default).

If *window* is NULL, the top window is used.

virtual wxContextHelp::~wxContextHelp ( ) `[virtual]`

Destroys the context help object.

21.121.3 Member Function Documentation

bool wxContextHelp::BeginContextHelp (wxWindow * *window*)

Puts the application into context-sensitive help mode.

window is the window which will be used to catch events; if NULL, the top window will be used.

Returns true if the application was successfully put into context-sensitive help mode. This function only returns when the event loop has finished.

bool wxContextHelp::EndContextHelp ( )

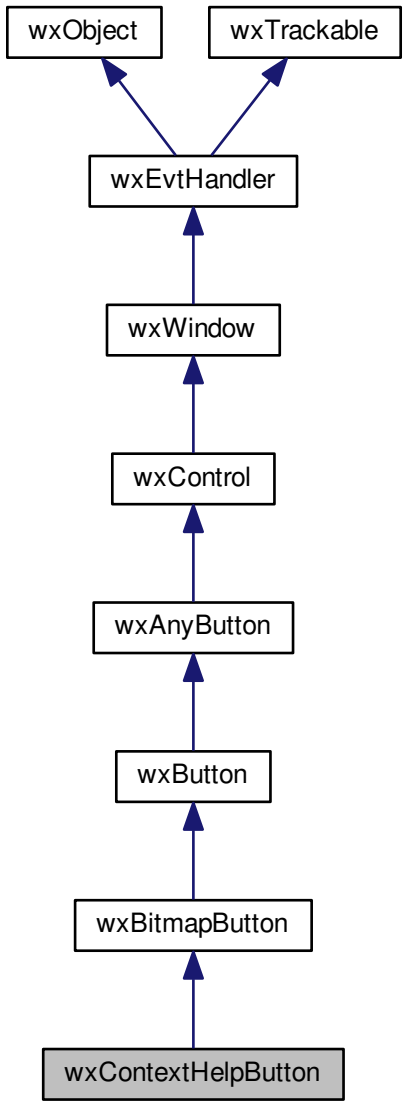
Ends context-sensitive help mode.

Not normally called by the application.

21.122 wxContextHelpButton Class Reference

`#include <wx/cshelp.h>`

Inheritance diagram for wxContextHelpButton:



21.122.1 Detailed Description

Instances of this class may be used to add a question mark button that when pressed, puts the application into context-help mode.

It does this by creating a wxContextHelp object which itself generates a `wxEVT_HELP` event when the user clicks on a window.

On Windows, you may add a question-mark icon to a dialog by use of the wxDIALOG_EX_CONTEXTHELP extra style, but on other platforms you will have to add a button explicitly, usually next to OK, Cancel or similar buttons.

Library: wxCore

Category: Help

See also

wxBitmapButton, wxContextHelp

Public Member Functions

- wxContextHelpButton (wxWindow *parent, wxWindowID id=wxID_CONTEXT_HELP, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxBU_AUTODRAW)

  Constructor, creating and showing a context help button.

Additional Inherited Members

21.122.2 Constructor & Destructor Documentation

wxContextHelpButton::wxContextHelpButton (wxWindow * *parent*, wxWindowID *id* = wxID_CONTEXT_HELP, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxBU_AUTODRAW)

Constructor, creating and showing a context help button.

Parameters

parent	Parent window. Must not be NULL.
id	Button identifier. Defaults to wxID_CONTEXT_HELP.
pos	Button position. If wxDefaultPosition is specified then a default position is chosen.
size	Button size. If wxDefaultSize is specified then the button is sized appropriately for the question mark bitmap.
style	Window style.

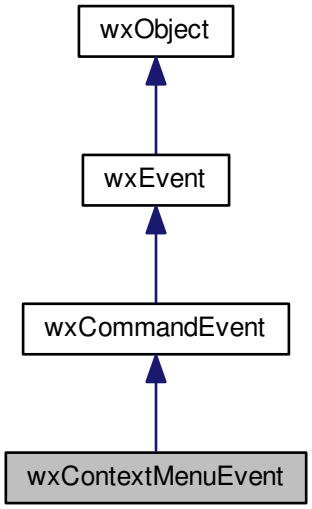
Remarks

Normally you only need pass the parent window to the constructor, and use the defaults for the remaining parameters.

21.123 wxContextMenuEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxContextMenuEvent:



21.123.1 Detailed Description

This class is used for context menu events, sent to give the application a chance to show a context (popup) menu for a wxWindow.

Note that if wxContextMenuEvent::GetPosition returns wxDefaultPosition, this means that the event originated from a keyboard context button event, and you should compute a suitable position yourself, for example by calling wx↩GetMousePosition().

Notice that the exact sequence of mouse events is different across the platforms. For example, under MSW the context menu event is generated after `EVT_RIGHT_UP` event and only if it was not handled but under GTK the context menu event is generated after `EVT_RIGHT_DOWN` event. This is correct in the sense that it ensures that the context menu is shown according to the current platform UI conventions and also means that you must not handle (or call wxEvent::Skip() in your handler if you do have one) neither right mouse down nor right mouse up event if you plan on handling `EVT_CONTEXT_MENU` event.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxContextMenuEvent& event)

Event macros:

- EVT_CONTEXT_MENU(func): A right click (or other context menu command depending on platform) has been detected.

Library: wxCore

Category: Events

See also

wxCommandEvent, Events and Event Handling

Public Member Functions

- wxContextMenuEvent (wxEventType type=wxEVT_NULL, int id=0, const wxPoint &pos=wxDefaultPosition)

  Constructor.
- const wxPoint & GetPosition () const

  Returns the position in screen coordinates at which the menu should be shown.
- void SetPosition (const wxPoint &point)

  Sets the position at which the menu should be shown.

Additional Inherited Members

21.123.2 Constructor & Destructor Documentation

wxContextMenuEvent::wxContextMenuEvent (wxEventType *type* = wxEVT_NULL, int *id* = 0, const wxPoint & *pos* = wxDefaultPosition)

Constructor.

21.123.3 Member Function Documentation

const wxPoint& wxContextMenuEvent::GetPosition ( ) const

Returns the position in screen coordinates at which the menu should be shown.

Use wxWindow::ScreenToClient to convert to client coordinates.

You can also omit a position from wxWindow::PopupMenu in order to use the current mouse pointer position.

If the event originated from a keyboard event, the value returned from this function will be wxDefaultPosition.

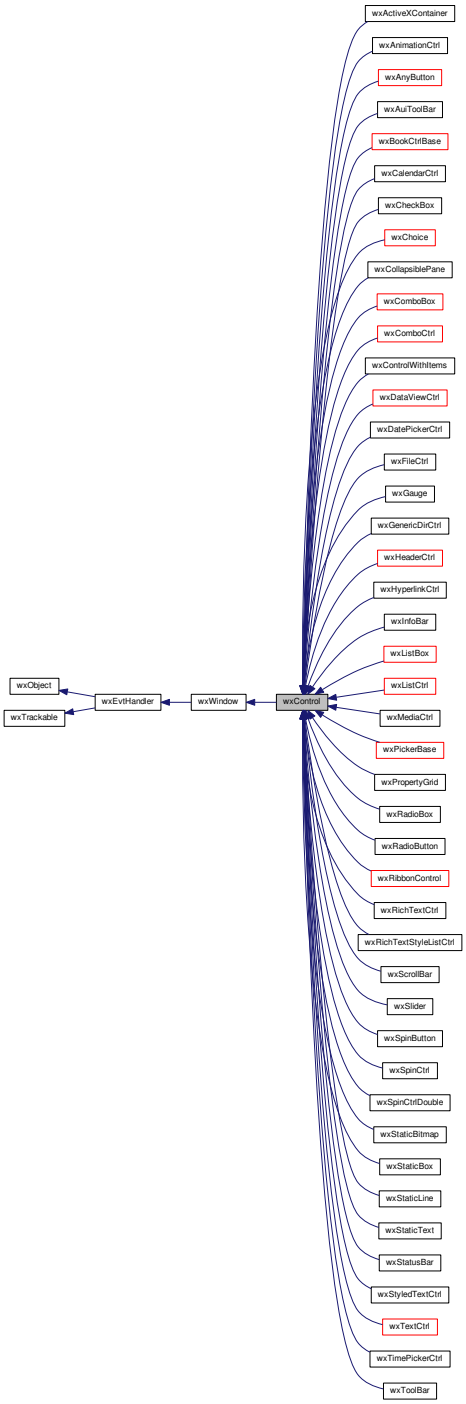
void wxContextMenuEvent::SetPosition (const wxPoint & *point*)

Sets the position at which the menu should be shown.

21.124 wxControl Class Reference

```
#include <wx/control.h>
```

Inheritance diagram for wxControl:



21.124.1 Detailed Description

This is the base class for a control or "widget".

A control is generally a small window which processes user input and/or displays one or more item of data.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxClipboardTextEvent& event)

Event macros for events emitted by this class:

- EVT_TEXT_COPY(id, func): Some or all of the controls content was copied to the clipboard.
- EVT_TEXT_CUT(id, func): Some or all of the controls content was cut (i.e. copied and deleted).
- EVT_TEXT_PASTE(id, func): Clipboard content was pasted into the control.

Library: wxCore

Category: Controls

See also

wxValidator

Public Member Functions

- wxControl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxControlNameStr)

  Constructs a control.
- wxControl ()

  Default constructor to allow 2-phase creation.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxControlNameStr)
- virtual void Command (wxCommandEvent &event)

  Simulates the effect of the user issuing a command to the item.
- wxString GetLabel () const

  Returns the control's label, as it was passed to SetLabel().
- wxString GetLabelText () const

  Returns the control's label without mnemonics.
- wxSize GetSizeFromTextSize (int xlen, int ylen=-1) const

  Determine the size needed by the control to leave the given area for its text.
- wxSize GetSizeFromTextSize (const wxSize &tsize) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetLabel (const wxString &label)

  Sets the control's label.
- void SetLabelText (const wxString &text)

  Sets the control's label to exactly the given string.
- bool SetLabelMarkup (const wxString &markup)

  Sets the controls label to a string using markup.

Static Public Member Functions

- static wxString GetLabelText (const wxString &label)

  Returns the given label string without mnemonics ("&" characters).

- static wxString RemoveMnemonics (const wxString &str)

  Returns the given str string without mnemonics ("&" characters).

- static wxString EscapeMnemonics (const wxString &text)

  Escapes the special mnemonics characters ("&") in the given string.

- static wxString Ellipsize (const wxString &label, const wxDC &dc, wxEllipsizeMode mode, int maxWidth, int flags=wxELLIPSIZE_FLAGS_DEFAULT)

  Replaces parts of the label string with ellipsis, if needed, so that it fits into maxWidth pixels if possible.

Additional Inherited Members

21.124.2 Constructor & Destructor Documentation

wxControl::wxControl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxControlNameStr)

Constructs a control.

Parameters

parent	Pointer to a parent window.
id	Control identifier. If wxID_ANY, will automatically create an identifier.
pos	Control position. wxDefaultPosition indicates that wxWidgets should generate a default position for the control.
size	Control size. wxDefaultSize indicates that wxWidgets should generate a default size for the window. If no suitable size can be found, the window will be sized to 20x20 pixels so that the window is visible but obviously not correctly sized.
style	Control style. For generic window styles, please see wxWindow.
validator	Control validator.
name	Control name.

wxControl::wxControl ( )

Default constructor to allow 2-phase creation.

21.124.3 Member Function Documentation

virtual void wxControl::Command (wxCommandEvent & *event*) [virtual]

Simulates the effect of the user issuing a command to the item.

See also

wxCommandEvent

Reimplemented in wxRichTextCtrl.

bool wxControl::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxControlNameStr)

static wxString wxControl::Ellipsize (const wxString & *label*, const wxDC & *dc*, wxEllipsizeMode *mode*, int *maxWidth*, int *flags* = wxELLIPSIZE_FLAGS_DEFAULT) [static]

Replaces parts of the *label* string with ellipsis, if needed, so that it fits into *maxWidth* pixels if possible.

Note that this function does *not* guarantee that the returned string will always be shorter than *maxWidth*; if *maxWidth* is extremely small, ellipsized text may be larger.

Parameters

label	The string to ellipsize
dc	The DC used to retrieve the character widths through the wxDC::GetPartialTextExtents() function.
mode	The ellipsization mode. This is the setting which determines which part of the string should be replaced by the ellipsis. See wxEllipsizeMode enumeration values for more info.
maxWidth	The maximum width of the returned string in pixels. This argument determines how much characters of the string need to be removed (and replaced by ellipsis).
flags	One or more of the wxEllipsizeFlags enumeration values combined.

static wxString wxControl::EscapeMnemonics (const wxString & *text*) [static]

Escapes the special mnemonics characters ("&") in the given string.

This function can be helpful if you need to set the controls label to a user-provided string. If the string contains ampersands, they wouldn't appear on the display but be used instead to indicate that the character following the first of them can be used as a control mnemonic. While this can sometimes be desirable (e.g. to allow the user to configure mnemonics of the controls), more often you will want to use this function before passing a user-defined string to SetLabel(). Alternatively, if the label is entirely user-defined, you can just call SetLabelText() directly – but this function must be used if the label is a combination of a part defined by program containing the control mnemonics and a user-defined part.

Parameters

text	The string such as it should appear on the display.

Returns

The same string with the ampersands in it doubled.

wxString wxControl::GetLabel ( ) const [virtual]

Returns the control's label, as it was passed to SetLabel().

Note that the returned string may contains mnemonics ("&" characters) if they were passed to the SetLabel() function; use GetLabelText() if they are undesired.

Also note that the returned string is always the string which was passed to SetLabel() but may be different from the string passed to SetLabelText() (since this last one escapes mnemonic characters).

Reimplemented from wxWindow.

wxString wxControl::GetLabelText ( ) const

Returns the control's label without mnemonics.

Note that because of the stripping of the mnemonics the returned string may differ from the string which was passed to SetLabel() but should always be the same which was passed to SetLabelText().

static wxString wxControl::GetLabelText (const wxString & *label*) [static]

Returns the given *label* string without mnemonics ("&" characters).

wxSize wxControl::GetSizeFromTextSize (int *xlen*, int *ylen* = -1) const

Determine the size needed by the control to leave the given area for its text.

This function is mostly useful with control displaying short amounts of text that can be edited by the user, e.g. wxTextCtrl, wxComboBox, wxSearchCtrl etc. Typically it is used to size these controls for the maximal amount of input they are supposed to contain, for example:

```
// Create a control for post code entry.
wxTextCtrl* postcode = new wxTextCtrl(this, ...);

// And set its initial and minimal size to be big enough for
// entering 5 digits.
postcode->SetInitialSize(
    postcode->GetSizeFromTextSize(
        postcode->GetTextExtent("99999")));
```

Currently this method is only implemented for wxTextCtrl, wxComboBox and wxChoice in wxMSW and wxGTK.

Parameters

xlen	The horizontal extent of the area to leave for text, in pixels.
ylen	The vertical extent of the area to leave for text, in pixels. By default -1 meaning that the vertical component of the returned size should be the default height of this control.

Returns

The size that the control should have to leave the area of the specified size for its text. May return wxDefaultSize if this method is not implemented for this particular control under the current platform.

Since

2.9.5

wxSize wxControl::GetSizeFromTextSize (const wxSize & *tsize*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

static wxString wxControl::RemoveMnemonics (const wxString & *str*) [static]

Returns the given *str* string without mnemonics ("&" characters).

Note

This function is identical to GetLabelText() and is provided mostly for symmetry with EscapeMnemonics().

void wxControl::SetLabel (const wxString & *label*) `[virtual]`

Sets the control's label.

All "&" characters in the *label* are special and indicate that the following character is a *mnemonic* for this control and can be used to activate it from the keyboard (typically by using *Alt* key in combination with it). To insert a literal ampersand character, you need to double it, i.e. use "&&". If this behaviour is undesirable, use SetLabelText() instead.

Reimplemented from wxWindow.

bool wxControl::SetLabelMarkup (const wxString & *markup*)

Sets the controls label to a string using markup.

Simple markup supported by this function can be used to apply different fonts or colours to different parts of the control label when supported. If markup is not supported by the control or platform, it is simply stripped and SetLabel() is used with the resulting string.

For example,

```
wxStaticText *text;
...
text->SetLabelMarkup("<b>&amp;amp;Bed</b> &amp;amp;mp; "
                     "<span foreground='red'>breakfast</span> "
                     "available <big>HERE</big>");
```

would show the string using bold, red and big for the corresponding words under wxGTK but will simply show the string "Bed & breakfast available HERE" on the other platforms. In any case, the "B" of "Bed" will be underlined to indicate that it can be used as a mnemonic for this control.

The supported tags are:

Tag	Description
<b>	bold text
<big>	bigger text
<i>	italic text
<s>	strike-through text
<small>	smaller text
<tt>	monospaced text
<u>	underlined text
	generic formatter tag, see the table below for supported attributes.

Supported attributes:

Name	Description
foreground, fgcolor, color	Foreground text colour, can be a name or RGB value.
background, bgcolor	Background text colour, can be a name or RGB value.
font_family, face	Font face name.
font_weight, weight	Numeric value in 0..900 range or one of "ultralight", "light", "normal" (all meaning non-bold), "bold", "ultrabold" and "heavy" (all meaning bold).
font_style, style	Either "oblique" or "italic" (both with the same meaning) or "normal".
size	The font size can be specified either as "smaller" or "larger" relatively to the current font, as a CSS font size name ("xx-small", "x-small", "small", "medium", "large", "x-large" or "xx-large") or as a number giving font size in 1024th parts of a point, i.e. 10240 for a 10pt font.

This markup language is a strict subset of Pango markup (described at http://library.gnome.org/devel/pango/unstable/PangoMarkupFormat.html) and any tags and span attributes not

documented above can't be used under non-GTK platforms.

Also note that you need to escape the following special characters:

Special character	Escape as
&	& or as &&
'	&apos;
"	&quot;
<	&lt;
>	&gt;

The non-escaped ampersand & characters are interpreted as mnemonics as with wxControl::SetLabel.

Parameters

markup	String containing markup for the label. It may contain markup tags described above and newline characters but currently only wxGTK and wxOSX support multiline labels with markup, the generic implementation (also used in wxMSW) only handles single line markup labels. Notice that the string must be well-formed (e.g. all tags must be correctly closed) and won't be shown at all otherwise.

Returns

true if the new label was set (even if markup in it was ignored) or false if we failed to parse the markup. In this case the label remains unchanged.

Currently wxButton supports markup in all major ports (wxMSW, wxGTK and wxOSX/Cocoa) while wxStaticText supports it in wxGTK and wxOSX and its generic version (which can be used under MSW if markup support is required). Extending support to more controls is planned in the future.

Since

2.9.2

void wxControl::SetLabelText (const wxString & *text*)

Sets the control's label to exactly the given string.

Unlike SetLabel(), this function shows exactly the *text* passed to it in the control, without interpreting ampersands in it in any way. Notice that it means that the control can't have any mnemonic defined for it using this function.

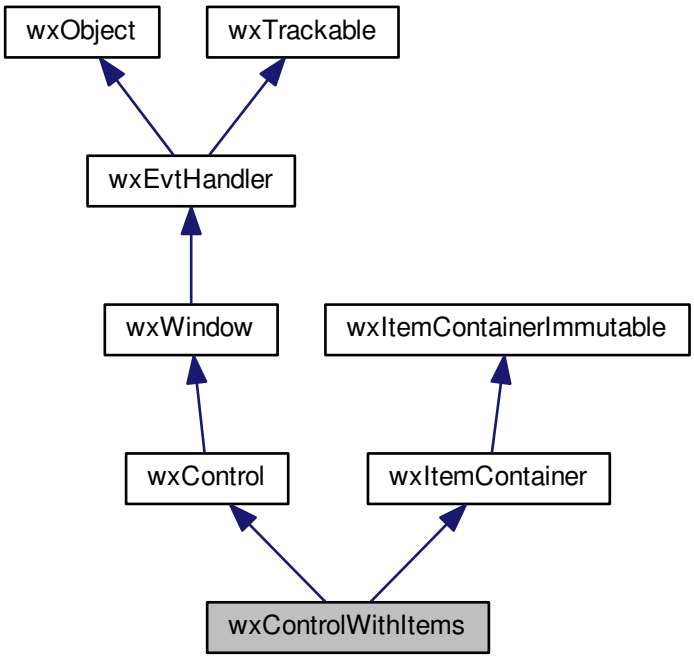
See also

EscapeMnemonics()

21.125 wxControlWithItems Class Reference

`#include <wx/ctrlsub.h>`

Inheritance diagram for wxControlWithItems:



21.125.1 Detailed Description

This is convenience class that derives from both wxControl and wxItemContainer.

It is used as basis for some wxWidgets controls (wxChoice and wxListBox).

Library: wxCore

Category: Controls

See also

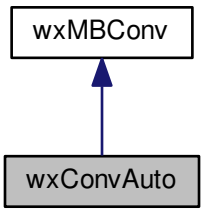
wxItemContainer, wxItemContainerImmutable

Additional Inherited Members

21.126 wxConvAuto Class Reference

`#include <wx/convauto.h>`

Inheritance diagram for wxConvAuto:



21.126.1 Detailed Description

This class implements a Unicode to/from multibyte converter capable of automatically recognizing the encoding of the multibyte text on input.

The logic used is very simple: the class uses the BOM (byte order mark) if it's present and tries to interpret the input as UTF-8 otherwise. If this fails, the input is interpreted as being in the default multibyte encoding which can be specified in the constructor of a wxConvAuto instance and, in turn, defaults to the value of GetFallbackEncoding() if not explicitly given.

For the conversion from Unicode to multibyte, the same encoding as was previously used for multibyte to Unicode conversion is reused. If there had been no previous multibyte to Unicode conversion, UTF-8 is used by default. Notice that once the multibyte encoding is automatically detected, it doesn't change any more, i.e. it is entirely determined by the first use of wxConvAuto object in the multibyte-to-Unicode direction. However creating a copy of wxConvAuto object, either via the usual copy constructor or assignment operator, or using wxMBConv::Clone(), resets the automatically detected encoding so that the new copy will try to detect the encoding of the input on first use.

This class is used by default in wxWidgets classes and functions reading text from files such as wxFile, wxFFile, wxTextFile, wxFileConfig and various stream classes so the encoding set with its SetFallbackEncoding() method will affect how these classes treat input files. In particular, use this method to change the fall-back multibyte encoding used to interpret the contents of the files whose contents isn't valid UTF-8 or to disallow it completely.

Library: wxBase

Category: Data Structures

See also

wxMBConv Overview

Public Member Functions

- wxConvAuto (wxFontEncoding enc=wxFONTENCODING_DEFAULT)

  Constructs a new wxConvAuto instance.

- wxBOM GetBOM () const

  Return the detected BOM type.
- const char * GetBOMChars (wxBOM bom, size_t *count)

  Return a pointer to the characters that makes up this BOM.

Static Public Member Functions

- static void DisableFallbackEncoding ()

  Disable the use of the fall back encoding: if the input doesn't have a BOM and is not valid UTF-8, the conversion will fail.
- static wxFontEncoding GetFallbackEncoding ()

  Returns the encoding used by default by wxConvAuto if no other encoding is explicitly specified in constructor.
- static void SetFallbackEncoding (wxFontEncoding enc)

  Changes the encoding used by default by wxConvAuto if no other encoding is explicitly specified in constructor.
- static wxBOM DetectBOM (const char *src, size_t srcLen)

  Return the BOM type of this buffer.

21.126.2 Constructor & Destructor Documentation

wxConvAuto::wxConvAuto (wxFontEncoding *enc* = wxFONTENCODING_DEFAULT)

Constructs a new wxConvAuto instance.

The object will try to detect the input of the multibyte text given to its wxMBConv::ToWChar() method automatically but if the automatic detection of Unicode encodings fails, the fall-back encoding *enc* will be used to interpret it as multibyte text.

The default value of *enc*, `wxFONTENCODING_DEFAULT`, means that the global default value (which can be set using SetFallbackEncoding()) should be used. As with that method, passing `wxFONTENCODING_MAX` inhibits using this encoding completely so the input multibyte text will always be interpreted as UTF-8 in the absence of BOM and the conversion will fail if the input doesn't form valid UTF-8 sequence.

Another special value is `wxFONTENCODING_SYSTEM` which means to use the encoding currently used on the user system, i.e. the encoding returned by wxLocale::GetSystemEncoding(). Any other encoding will be used as is, e.g. passing `wxFONTENCODING_ISO8859_1` ensures that non-UTF-8 input will be treated as latin1.

21.126.3 Member Function Documentation

static wxBOM wxConvAuto::DetectBOM (const char * *src*, size_t *srcLen*) `[static]`

Return the BOM type of this buffer.

This is a helper function which is normally only used internally by wxConvAuto but provided for convenience of the code that wants to detect the encoding of a stream by checking it for BOM presence on its own.

Since

2.9.3

static void wxConvAuto::DisableFallbackEncoding ( ) `[static]`

Disable the use of the fall back encoding: if the input doesn't have a BOM and is not valid UTF-8, the conversion will fail.

wxBOM wxConvAuto::GetBOM ( ) const

Return the detected BOM type.

The BOM type is detected after sufficiently many initial bytes have passed through this conversion object so it will always return wxBOM_Unknown immediately after the object creation but may return a different value later.

Since

2.9.3

const char* wxConvAuto::GetBOMChars (wxBOM *bom*, size_t * *count*)

Return a pointer to the characters that makes up this BOM.

The returned character count is 2, 3 or 4, or undefined if the return value is NULL.

Parameters

bom	A valid BOM type, i.e. not wxBOM_Unknown or wxBOM_None.
count	A non-NULL pointer receiving the number of characters in this BOM.

Returns

Pointer to characters composing the BOM or NULL if BOM is unknown or invalid. Notice that the returned string is not NUL-terminated and may contain embedded NULs so *count* must be used to handle it correctly.

Since

2.9.3

static wxFontEncoding wxConvAuto::GetFallbackEncoding ( ) [static]

Returns the encoding used by default by wxConvAuto if no other encoding is explicitly specified in constructor.

By default, returns `wxFONTENCODING_ISO8859_1` but can be changed using SetFallbackEncoding().

static void wxConvAuto::SetFallbackEncoding (wxFontEncoding *enc*) [static]

Changes the encoding used by default by wxConvAuto if no other encoding is explicitly specified in constructor.

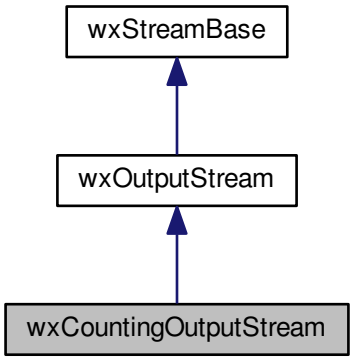
The default value, which can be retrieved using GetFallbackEncoding(), is `wxFONTENCODING_ISO8859_1`.

Special values of `wxFONTENCODING_SYSTEM` or `wxFONTENCODING_MAX` can be used for the *enc* parameter to use the encoding of the current user locale as fall back or not use any encoding for fall back at all, respectively (just as with the similar constructor parameter). However, `wxFONTENCODING_DEFAULT` can't be used here.

21.127 wxCountingOutputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxCountingOutputStream:



21.127.1 Detailed Description

wxCountingOutputStream is a specialized output stream which does not write any data anywhere, instead it counts how many bytes would get written if this were a normal stream.

This can sometimes be useful or required if some data gets serialized to a stream or compressed by using stream compression and thus the final size of the stream cannot be known other than pretending to write the stream. One case where the resulting size would have to be known is if the data has to be written to a piece of memory and the memory has to be allocated before writing to it (which is probably always the case when writing to a memory stream).

Library: wxBase

Category: Streams

Public Member Functions

- wxCountingOutputStream ()

  Creates a wxCountingOutputStream object.
- virtual ~wxCountingOutputStream ()

  Destructor.
- virtual wxFileOffset GetLength () const

  Returns the current length of the stream.

Additional Inherited Members

21.127.2 Constructor & Destructor Documentation

wxCountingOutputStream::wxCountingOutputStream ( )

Creates a wxCountingOutputStream object.

virtual wxCountingOutputStream::~wxCountingOutputStream ( ) [virtual]

Destructor.

21.127.3 Member Function Documentation

virtual wxFileOffset wxCountingOutputStream::GetLength ( ) const [virtual]

Returns the current length of the stream.

This is the amount of data written to the stream so far, in bytes.

Reimplemented from wxStreamBase.

21.128 wxCriticalSection Class Reference

`#include <wx/thread.h>`

21.128.1 Detailed Description

A critical section object is used for exactly the same purpose as a wxMutex.

The only difference is that under Windows platform critical sections are only visible inside one process, while mutexes may be shared among processes, so using critical sections is slightly more efficient.

The terminology is also slightly different: mutex may be locked (or acquired) and unlocked (or released) while critical section is entered and left by the program.

Finally, you should try to use wxCriticalSectionLocker class whenever possible instead of directly using wxCriticalSection for the same reasons wxMutexLocker is preferable to wxMutex - please see wxMutex for an example.

Library: wxBase

Category: Threading

Note

Critical sections can be used before the wxWidgets library is fully initialized. In particular, it's safe to create global wxCriticalSection instances.

See also

wxThread, wxCondition, wxCriticalSectionLocker

Public Member Functions

- wxCriticalSection (wxCriticalSectionType critSecType=wxCRITSEC_DEFAULT)

  Default constructor initializes critical section object.
- ~wxCriticalSection ()

  Destructor frees the resources.
- void Enter ()

  Enter the critical section (same as locking a mutex): if another thread has already entered it, this call will block until the other thread calls Leave().

- bool TryEnter ()

  Try to enter the critical section (same as trying to lock a mutex).

- void Leave ()

  Leave the critical section allowing other threads use the global data protected by it.

21.128.2 Constructor & Destructor Documentation

wxCriticalSection::wxCriticalSection (wxCriticalSectionType *critSecType* = wxCRITSEC_DEFAULT)

Default constructor initializes critical section object.

By default critical sections are recursive under Unix and Windows.

wxCriticalSection::~wxCriticalSection ( )

Destructor frees the resources.

21.128.3 Member Function Documentation

void wxCriticalSection::Enter ( )

Enter the critical section (same as locking a mutex): if another thread has already entered it, this call will block until the other thread calls Leave().

There is no error return for this function.

After entering the critical section protecting a data variable, the thread running inside the critical section may safely use/modify it.

Note that entering the same critical section twice or more from the same thread doesn't result in a deadlock; in this case in fact this function will immediately return.

void wxCriticalSection::Leave ( )

Leave the critical section allowing other threads use the global data protected by it.

There is no error return for this function.

bool wxCriticalSection::TryEnter ( )

Try to enter the critical section (same as trying to lock a mutex).

If it can't, immediately returns false.

Since

2.9.3

21.129 wxCriticalSectionLocker Class Reference

`#include <wx/thread.h>`

21.129.1 Detailed Description

This is a small helper class to be used with wxCriticalSection objects.

A wxCriticalSectionLocker enters the critical section in the constructor and leaves it in the destructor making it much more difficult to forget to leave a critical section (which, in general, will lead to serious and difficult to debug problems).

Example of using it:

```
void Set Foo()
{
    // gs_critSect is some (global) critical section guarding access to the
    // object "foo"
    wxCriticalSectionLocker locker(gs_critSect);

    if ( ... )
    {
        // do something
        ...

        return;
    }

    // do something else
    ...

    return;
}
```

Without wxCriticalSectionLocker, you would need to remember to manually leave the critical section before each `return`.

Library: wxBase

Category: Threading

See also

wxCriticalSection, wxMutexLocker

Public Member Functions

- wxCriticalSectionLocker (wxCriticalSection &criticalsection)

  Constructs a wxCriticalSectionLocker object associated with given criticalsection and enters it.
- ~wxCriticalSectionLocker ()

  Destructor leaves the critical section.

21.129.2 Constructor & Destructor Documentation

wxCriticalSectionLocker::wxCriticalSectionLocker (wxCriticalSection & *criticalsection*)

Constructs a wxCriticalSectionLocker object associated with given *criticalsection* and enters it.

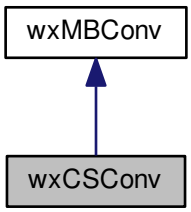
wxCriticalSectionLocker::~wxCriticalSectionLocker ( )

Destructor leaves the critical section.

21.130 wxCSConv Class Reference

`#include <wx/strconv.h>`

Inheritance diagram for wxCSConv:



21.130.1 Detailed Description

This class converts between any character set supported by the system and Unicode.

Please notice that this class uses system-provided conversion functions, e.g. `MultiByteToWideChar()` and `WideCharToMultiByte()` under MSW and `iconv(3)` under Unix systems and as such may support different encodings and different encoding names on different platforms (although all relatively common encodings are supported should be supported everywhere).

It has one predefined instance, **wxConvLocal**, for the default user character set.

Library: wxBase

Category: Text Conversion

See also

wxMBConv, wxEncodingConverter, wxMBConv Overview

Public Member Functions

- wxCSConv (const wxString &charset)

  Constructor.
- wxCSConv (wxFontEncoding encoding)

  Constructor.
- bool IsOk () const

  Returns true if the charset (or the encoding) given at constructor is really available to use.

Additional Inherited Members

21.130.2 Constructor & Destructor Documentation

wxCSConv::wxCSConv (const wxString & *charset*)

Constructor.

You can specify the name of the character set you want to convert from/to. If the character set name is not recognized, ISO 8859-1 is used as fall back, use IsOk() to test for this.

Parameters

charset	The name of the encoding, shouldn't be empty.

wxCSConv::wxCSConv (wxFontEncoding *encoding*)

Constructor.

You can specify an encoding constant for the character set you want to convert from/to. Use IsOk() after construction to check whether the encoding is supported by the current system.

Parameters

encoding	Any valid (i.e. not wxFONTENCODING_MAX) font encoding.

21.130.3 Member Function Documentation

bool wxCSConv::IsOk ( ) const

Returns true if the charset (or the encoding) given at constructor is really available to use.

Returns false if ISO 8859-1 will be used instead.

Note this does not mean that a given string will be correctly converted. A malformed string may still make conversion functions return `wxCONV_FAILED`.

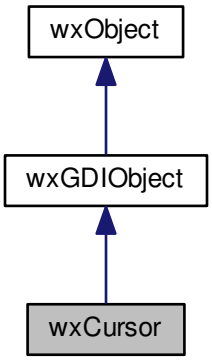
Since

2.8.2

21.131 wxCursor Class Reference

```
#include <wx/cursor.h>
```

Inheritance diagram for wxCursor:



21.131.1 Detailed Description

A cursor is a small bitmap usually used for denoting where the mouse pointer is, with a picture that might indicate the interpretation of a mouse click.

As with icons, cursors in X and MS Windows are created in a different manner. Therefore, separate cursors will be created for the different environments. Platform-specific methods for creating a wxCursor object are catered for, and this is an occasion where conditional compilation will probably be required (see wxIcon for an example).

A single cursor object may be used in many windows (any subwindow type). The wxWidgets convention is to set the cursor for a window, as in X, rather than to set it globally as in MS Windows, although a global wxSetCursor() function is also available for MS Windows use.

21.131.2 Creating a Custom Cursor

The following is an example of creating a cursor from 32x32 bitmap data (down_bits) and a mask (down_mask) where 1 is black and 0 is white for the bits, and 1 is opaque and 0 is transparent for the mask. It works on Windows and GTK+.

```
static char down_bits[] = { 255, 255, 255, 255, 31,
    255, 255, 255, 31, 255, 255, 255, 31, 255, 255, 255,
    31, 255, 255, 255, 31, 255, 255, 255, 31, 255, 255,
    255, 31, 255, 255, 255, 31, 255, 255, 255, 25, 243,
    255, 255, 19, 249, 255, 255, 7, 252, 255, 255, 15, 254,
    255, 255, 31, 255, 255, 255, 191, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
    255 };

static char down_mask[] = { 240, 1, 0, 0, 240, 1,
    0, 0, 240, 1, 0, 0, 240, 1, 0, 0, 240, 1, 0, 0, 240, 1,
    0, 0, 240, 1, 0, 0, 240, 1, 0, 0, 255, 31, 0, 0, 255,
    31, 0, 0, 254, 15, 0, 0, 252, 7, 0, 0, 248, 3, 0, 0,
    240, 1, 0, 0, 224, 0, 0, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0 };

#ifdef __WXMSW__
```

```
    wxBitmap down_bitmap(down_bits, 32, 32);
    wxBitmap down_mask_bitmap(down_mask, 32, 32);

    down_bitmap.SetMask(new wxMask(down_mask_bitmap));
    wxImage down_image = down_bitmap.ConvertToImage();
    down_image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_X, 6);
    down_image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_Y, 14);
    wxCursor down_cursor = wxCursor(down_image);
#elif defined(__WXGTK__) or defined(__WXMOTIF__)
    wxCursor down_cursor = wxCursor(down_bits, 32, 32, 6, 14,
                                    down_mask, wxWHITE, wxBLACK);
#endif
```

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers:

- wxNullCursor
- wxSTANDARD_CURSOR
- wxHOURGLASS_CURSOR
- wxCROSS_CURSOR

See also

wxBitmap, wxIcon, wxWindow::SetCursor(), wxSetCursor(), wxStockCursor

Public Member Functions

- wxCursor ()

  Default constructor.
- wxCursor (const char bits[], int width, int height, int hotSpotX=-1, int hotSpotY=-1, const char maskBits[]=NULL)

  Constructs a cursor by passing an array of bits (XBM data).
- wxCursor (const wxString &cursorName, wxBitmapType type=wxCURSOR_DEFAULT_TYPE, int hotSpotX=0, int hotSpotY=0)

  Constructs a cursor by passing a string resource name or filename.
- wxCursor (wxStockCursor cursorId)

  Constructs a cursor using a cursor identifier.
- wxCursor (const wxImage &image)

  Constructs a cursor from a wxImage.
- wxCursor (const wxCursor &cursor)

  Copy constructor, uses reference counting.
- virtual ~wxCursor ()

  Destroys the cursor.
- virtual bool IsOk () const

  Returns true if cursor data is present.
- wxPoint GetHotSpot () const

  Returns the coordinates of the cursor hot spot.
- wxCursor & operator= (const wxCursor &cursor)

  Assignment operator, using reference counting.

Additional Inherited Members

21.131.3 Constructor & Destructor Documentation

wxCursor::wxCursor ( )

Default constructor.

wxCursor::wxCursor (const char *bits[],* int *width,* int *height,* int *hotSpotX* = `-1`, int *hotSpotY* = `-1`, const char *maskBits[]* = `NULL`)

Constructs a cursor by passing an array of bits (XBM data).

The parameters *fg* and *bg* have an effect only on GTK+, and force the cursor to use particular background and foreground colours.

If either *hotSpotX* or *hotSpotY* is -1, the hotspot will be the centre of the cursor image (Motif only).

Parameters

bits	An array of XBM data bits.
width	Cursor width.
height	Cursor height.
hotSpotX	Hotspot x coordinate (relative to the top left of the image).
hotSpotY	Hotspot y coordinate (relative to the top left of the image).
maskBits	Bits for a mask bitmap.

Availability: only available for the wxGTK, wxMotif ports.

wxPerl Note: In wxPerl use Wx::Cursor->newData(bits, width, height, hotSpotX = -1, hotSpotY = -1, maskBits = 0).

wxCursor::wxCursor (const wxString & *cursorName,* wxBitmapType *type* = `wxCURSOR_DEFAULT_TYPE`, int *hotSpotX* = `0`, int *hotSpotY* = `0`)

Constructs a cursor by passing a string resource name or filename.

The arguments *hotSpotX* and *hotSpotY* are only used when there's no hotspot info in the resource/image-file to load (e.g. when using `wxBITMAP_TYPE_ICO` under wxMSW or `wxBITMAP_TYPE_XPM` under wxGTK).

Parameters

cursorName	The name of the resource or the image file to load.
type	Icon type to load. It defaults to `wxCURSOR_DEFAULT_TYPE`, which is a #define associated to different values on different platforms: • under Windows, it defaults to `wxBITMAP_TYPE_CUR_RESOURCE`. Other permitted types under Windows are `wxBITMAP_TYPE_CUR` (to load a cursor from a .cur cursor file), `wxBITMAP_TYPE_ICO` (to load a cursor from a .ico icon file) and `wxBITMAP_TYPE_ANI` (to load a cursor from a .ani icon file). • under MacOS, it defaults to `wxBITMAP_TYPE_MACCURSOR_RESOURCE`; when specifying a string resource name, first the color cursors 'crsr' and then the black/white cursors 'CURS' in the resource chain are scanned through. Note that resource forks are deprecated on OS X so this is only available for legacy reasons and should not be used in new code. • under GTK, it defaults to `wxBITMAP_TYPE_XPM`. See the wxCursor(const wxImage& image) ctor for more info. • under X11, it defaults to `wxBITMAP_TYPE_XPM`. • under Motif, it defaults to `wxBITMAP_TYPE_XBM`.
hotSpotX	Hotspot x coordinate (relative to the top left of the image).
hotSpotY	Hotspot y coordinate (relative to the top left of the image).

wxCursor::wxCursor (**wxStockCursor** *cursorId*)

Constructs a cursor using a cursor identifier.

Parameters

cursorId	A stock cursor identifier. See wxStockCursor.

wxCursor::wxCursor (**const wxImage &** *image*)

Constructs a cursor from a wxImage.

If cursor are monochrome on the current platform, colors with the RGB elements all greater than 127 will be foreground, colors less than this background. The mask (if any) will be used to specify the transparent area.

In wxMSW the foreground will be white and the background black. If the cursor is larger than 32x32 it is resized.

In wxGTK, colour cursors and alpha channel are supported (starting from GTK+ 2.2). Otherwise the two most frequent colors will be used for foreground and background. In any case, the cursor will be displayed at the size of the image.

Under wxMac (Cocoa), large cursors are supported.

Notice that the *image* can define the cursor hot spot. To set it you need to use wxImage::SetOption() with `wxIMAGE_OPTION_CUR_HOTSPOT_X` or `wxIMAGE_OPTION_CUR_HOTSPOT_Y`, e.g.

```
image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_X, hotSpotX);
image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_X, hotSpotY);
```

wxCursor::wxCursor (**const wxCursor &** *cursor*)

Copy constructor, uses reference counting.

Parameters

cursor	Pointer or reference to a cursor to copy.

virtual wxCursor::~wxCursor ( ) [virtual]

Destroys the cursor.

See reference-counted object destruction for more info.

A cursor can be reused for more than one window, and does not get destroyed when the window is destroyed. wxWidgets destroys all cursors on application exit, although it is best to clean them up explicitly.

21.131.4 Member Function Documentation

wxPoint wxCursor::GetHotSpot ( ) const

Returns the coordinates of the cursor hot spot.

The hot spot is the point at which the mouse is actually considered to be when this cursor is used.

This method is currently only implemented in wxMSW and wxGTK2+ and simply returns wxDefaultPosition in the other ports.

Since

3.1.0

virtual bool wxCursor::IsOk ( ) const [virtual]

Returns true if cursor data is present.

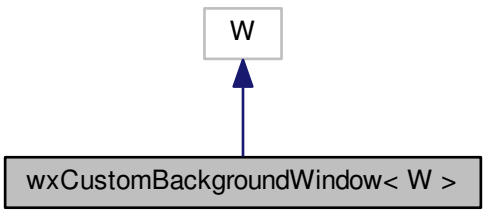
wxCursor& wxCursor::operator= (const wxCursor & *cursor*)

Assignment operator, using reference counting.

21.132 wxCustomBackgroundWindow< W > Class Template Reference

`#include <wx/custombgwin.h>`

Inheritance diagram for wxCustomBackgroundWindow< W >:



21.132.1 Detailed Description

template<class W>class wxCustomBackgroundWindow< W >

A helper class making it possible to use custom background for any window.

wxWindow itself only provides SetBackgroundColour() method taking a (solid) wxColour. This class extends it by allowing to use custom bitmap backgrounds with any window, provided that you inherit from it. Notice that the usual rule of not interfering with event handling or painting of native controls still applies, so you shouldn't try to use custom backgrounds with classes such as wxButton (even if this might work on some platforms, it's not guaranteed to work in general). But you can use this class in conjunction with wxWindow, wxPanel, wxFrame and other similar classes, e.g. the erase sample shows how to use it with wxScrolledWindow:

```
#include "wx/custombgwin.h"

class MyCanvas : public wxCustomBackgroundWindow<wxScrolledWindow>
{
public:
    MyCanvas(wxWindow* parent)
    {
        // Notice that we must explicitly call base class Create()
        // instead of using its ctor as wxCustomBackgroundWindow
        // doesn't define any non-default ctors.
        Create(parent, wxID_ANY);

        ...

        SetBackgroundBitmap(bitmap);
    }
};
```

Category: Miscellaneous Windows

Since

2.9.3

Public Member Functions

- wxCustomBackgroundWindow ()

  Trivial default constructor.
- void SetBackgroundBitmap (const wxBitmap &bmp)

  Set the background bitmap for this window.

21.132.2 Constructor & Destructor Documentation

template<class W > wxCustomBackgroundWindow< W >::wxCustomBackgroundWindow ( )

Trivial default constructor.

21.132.3 Member Function Documentation

template<class W > void wxCustomBackgroundWindow< W >::SetBackgroundBitmap (const wxBitmap & *bmp*)

Set the background bitmap for this window.

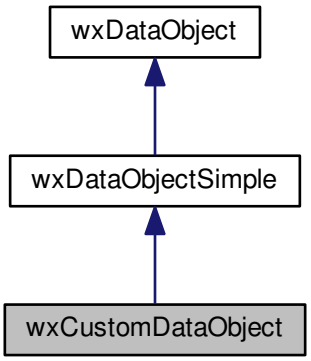
If *bmp* is a valid bitmap, this bitmap will be tiled over the panel background and show through any of its transparent children. Passing an invalid bitmap reverts to the default background appearance.

Notice that you must not prevent the base class EVT_ERASE_BACKGROUND handler from running (i.e. not to handle this event yourself) for this to work.

21.133 wxCustomDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxCustomDataObject:



21.133.1 Detailed Description

wxCustomDataObject is a specialization of wxDataObjectSimple for some application-specific data in arbitrary (either custom or one of the standard ones).

The only restriction is that it is supposed that this data can be copied bitwise (i.e. with `memcpy()`), so it would be a bad idea to make it contain a C++ object (though C struct is fine).

By default, wxCustomDataObject stores the data inside in a buffer. To put the data into the buffer you may use either SetData() or TakeData() depending on whether you want the object to make a copy of data or not.

This class may be used as is, but if you don't want store the data inside the object but provide it on demand instead, you should override GetSize(), GetData() and SetData() (or may be only the first two or only the last one if you only allow reading/writing the data).

Library: wxCore

Category: Clipboard and Drag & Drop

See also

wxDataObject

Public Member Functions

- wxCustomDataObject (const wxDataFormat &format=wxFormatInvalid)

  The constructor accepts a format argument which specifies the (single) format supported by this object.
- virtual ~wxCustomDataObject ()

  The destructor will free the data held by the object.
- virtual void * Alloc (size_t size)

This function is called to allocate size bytes of memory from SetData().

- virtual void Free ()

  This function is called when the data is freed, you may override it to anything you want (or may be nothing at all).

- virtual void * GetData () const

  Returns a pointer to the data.

- virtual size_t GetSize () const

  Returns the data size in bytes.

- virtual bool SetData (size_t size, const void *data)

  Set the data.

- void TakeData (size_t size, void *data)

  Like SetData(), but doesn't copy the data - instead the object takes ownership of the pointer.

Additional Inherited Members

21.133.2 Constructor & Destructor Documentation

wxCustomDataObject::wxCustomDataObject (const wxDataFormat & *format* = wxFormatInvalid)

The constructor accepts a *format* argument which specifies the (single) format supported by this object.

If it isn't set here, wxDataObjectSimple::SetFormat() should be used.

virtual wxCustomDataObject::~wxCustomDataObject ( ) [virtual]

The destructor will free the data held by the object.

Notice that although it calls the virtual Free() function, the base class version will always be called (C++ doesn't allow calling virtual functions from constructors or destructors), so if you override Free(), you should override the destructor in your class as well (which would probably just call the derived class' version of Free()).

21.133.3 Member Function Documentation

virtual void* wxCustomDataObject::Alloc (size_t *size*) [virtual]

This function is called to allocate *size* bytes of memory from SetData().

The default version just uses the operator new.

virtual void wxCustomDataObject::Free ( ) [virtual]

This function is called when the data is freed, you may override it to anything you want (or may be nothing at all).

The default version calls operator delete[] on the data.

virtual void* wxCustomDataObject::GetData ( ) const [virtual]

Returns a pointer to the data.

virtual size_t wxCustomDataObject::GetSize ( ) const [virtual]

Returns the data size in bytes.

virtual bool wxCustomDataObject::SetData (size_t *size*, const void * *data*) `[virtual]`

Set the data.

The data object will make an internal copy.

Reimplemented from wxDataObjectSimple.

void wxCustomDataObject::TakeData (size_t *size*, void * *data*)

Like SetData(), but doesn't copy the data - instead the object takes ownership of the pointer.

21.134 wxDataFormat Class Reference

`#include <wx/dataobj.h>`

21.134.1 Detailed Description

A wxDataFormat is an encapsulation of a platform-specific format handle which is used by the system for the clipboard and drag and drop operations.

The applications are usually only interested in, for example, pasting data from the clipboard only if the data is in a format the program understands and a data format is something which uniquely identifies this format.

On the system level, a data format is usually just a number (`CLIPFORMAT` under Windows or `Atom` under X11, for example) and the standard formats are, indeed, just numbers which can be implicitly converted to wxDataFormat. The standard formats are:

wxDF_INVALID	An invalid format - used as default argument for functions taking a wxDataFormat argument sometimes.
wxDF_TEXT	Text format (wxString).
wxDF_BITMAP	A bitmap (wxBitmap).
wxDF_METAFILE	A metafile (wxMetafile, Windows only).
wxDF_FILENAME	A list of filenames.
wxDF_HTML	An HTML string. This is currently only valid on Mac and MSW.

As mentioned above, these standard formats may be passed to any function taking wxDataFormat argument because wxDataFormat has an implicit conversion from them (or, to be precise from the type `wxDataFormat::NativeFormat` which is the type used by the underlying platform for data formats).

Aside the standard formats, the application may also use custom formats which are identified by their names (strings) and not numeric identifiers. Although internally custom format must be created (or *registered*) first, you shouldn't care about it because it is done automatically the first time the wxDataFormat object corresponding to a given format name is created. The only implication of this is that you should avoid having global wxDataFormat objects with non-default constructor because their constructors are executed before the program has time to perform all necessary initialisations and so an attempt to do clipboard format registration at this time will usually lead to a crash!

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, Drag & Drop Sample, wxDataObject

Public Member Functions

- wxDataFormat (wxDataFormatId format=wxDF_INVALID)

  Constructs a data format object for one of the standard data formats or an empty data object (use SetType() or SetId() later in this case).
- wxDataFormat (const wxString &format)

  Constructs a data format object for a custom format identified by its name format.
- wxString GetId () const

  Returns the name of a custom format (this function will fail for a standard format).
- wxDataFormatId GetType () const

  Returns the platform-specific number identifying the format.
- void SetId (const wxString &format)

  Sets the format to be the custom format identified by the given name.
- void SetType (wxDataFormatId type)

  Sets the format to the given value, which should be one of wxDF_XXX constants.
- bool operator!= (const wxDataFormat &format) const

  Returns true if the formats are different.
- bool operator!= (wxDataFormatId format) const

  Returns true if the formats are different.
- bool operator== (const wxDataFormat &format) const

  Returns true if the formats are equal.
- bool operator== (wxDataFormatId format) const

  Returns true if the formats are equal.

21.134.2 Constructor & Destructor Documentation

wxDataFormat::wxDataFormat (wxDataFormatId *format* = wxDF_INVALID)

Constructs a data format object for one of the standard data formats or an empty data object (use SetType() or SetId() later in this case).

wxPerl Note: In wxPerl use Wx::Bitmap->newNative(format).

wxDataFormat::wxDataFormat (const wxString & *format*)

Constructs a data format object for a custom format identified by its name *format*.

wxPerl Note: In wxPerl use Wx::Bitmap->newUser(format).

21.134.3 Member Function Documentation

wxString wxDataFormat::GetId ( ) const

Returns the name of a custom format (this function will fail for a standard format).

wxDataFormatId wxDataFormat::GetType ( ) const

Returns the platform-specific number identifying the format.

bool wxDataFormat::operator!= (const wxDataFormat & *format*) const

Returns true if the formats are different.

bool wxDataFormat::operator!= (wxDataFormatId *format*) const

Returns true if the formats are different.

bool wxDataFormat::operator== (const wxDataFormat & *format*) const

Returns true if the formats are equal.

bool wxDataFormat::operator== (wxDataFormatId *format*) const

Returns true if the formats are equal.

void wxDataFormat::SetId (const wxString & *format*)

Sets the format to be the custom format identified by the given name.

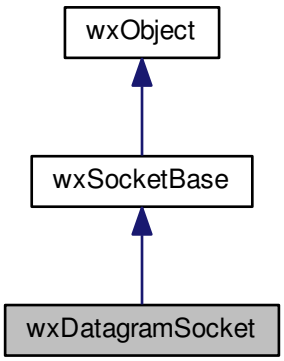
void wxDataFormat::SetType (wxDataFormatId *type*)

Sets the format to the given value, which should be one of wxDF_XXX constants.

21.135 wxDatagramSocket Class Reference

`#include <wx/socket.h>`

Inheritance diagram for wxDatagramSocket:



21.135.1 Detailed Description

Todo docme

Library: wxNet

Category: Networking

Public Member Functions

- wxDatagramSocket (const wxSockAddress &addr, wxSocketFlags flags=wxSOCKET_NONE)

  Constructor.
- virtual ~wxDatagramSocket ()

  Destructor.
- wxDatagramSocket & SendTo (const wxSockAddress &address, const void *buffer, wxUint32 nbytes)

  Write a buffer of nbytes bytes to the socket.

Additional Inherited Members

21.135.2 Constructor & Destructor Documentation

wxDatagramSocket::wxDatagramSocket (const wxSockAddress & *addr,* wxSocketFlags *flags* = wxSOCKET_NONE)

Constructor.

Parameters

addr	The socket address.
flags	Socket flags (See wxSocketBase::SetFlags()).

virtual wxDatagramSocket::~wxDatagramSocket ( ) [virtual]

Destructor.

Please see wxSocketBase::Destroy().

21.135.3 Member Function Documentation

wxDatagramSocket& wxDatagramSocket::SendTo (const wxSockAddress & *address,* const void * *buffer,* wxUint32 *nbytes*)

Write a buffer of *nbytes* bytes to the socket.

Use wxSocketBase::LastWriteCount() to verify the number of bytes actually wrote. Use wxSocketBase::Error() to determine if the operation succeeded.

Parameters

address	The address of the destination peer for this data.
buffer	Buffer where read data is.
nbytes	Number of bytes.

Returns

Returns a reference to the current object.

See also

wxSocketBase::LastError(), wxSocketBase::SetFlags()

21.136 wxDataInputStream Class Reference

`#include <wx/datstrm.h>`

21.136.1 Detailed Description

This class provides functions that read binary data types in a portable way.

Please see wxDataOutputStream for the discussion of the format expected by this stream on input, notably for the floating point values.

If you want to read data from text files (or streams) use wxTextInputStream instead.

The ">>" operator is overloaded and you can use this class like a standard C++ iostream. Note, however, that the arguments are the fixed size types wxUint32, wxInt32 etc and on a typical 32-bit computer, none of these match to the "long" type (wxInt32 is defined as signed int on 32-bit architectures) so that you cannot use long. To avoid problems (here and elsewhere), make use of the wxInt32, wxUint32, etc types.

For example:

```
wxFileInputStream input( "mytext.dat" );
wxDataInputStream store( input );
wxUint8 i1;
float f2;
wxString line;

store >> i1;       // read a 8 bit integer.
store >> i1 >> f2; // read a 8 bit integer followed by float.
store >> line;     // read a text line
```

Library: wxBase

Category: Streams

See also

wxDataOutputStream

Public Member Functions

- wxDataInputStream (wxInputStream &stream, const wxMBConv &conv=wxConvUTF8)

  Constructs a datastream object from an input stream.
- ~wxDataInputStream ()

  Destroys the wxDataInputStream object.
- void BigEndianOrdered (bool be_order)

  If be_order is true, all data will be read in big-endian order, such as written by programs on a big endian architecture (e.g.
- wxMBConv * GetConv () const

  Returns the current text conversion class used for reading strings.
- wxUint8 Read8 ()

  Reads a single byte from the stream.
- void Read8 (wxUint8 *buffer, size_t size)

  Reads bytes from the stream in a specified buffer.
- wxUint16 Read16 ()

  Reads a 16 bit unsigned integer from the stream.
- void Read16 (wxUint16 *buffer, size_t size)

  Reads 16 bit unsigned integers from the stream in a specified buffer.

- wxUint32 Read32 ()

  Reads a 32 bit unsigned integer from the stream.
- void Read32 (wxUint32 *buffer, size_t size)

  Reads 32 bit unsigned integers from the stream in a specified buffer.
- wxUint64 Read64 ()

  Reads a 64 bit unsigned integer from the stream.
- void Read64 (wxUint64 *buffer, size_t size)

  Reads 64 bit unsigned integers from the stream in a specified buffer.
- float ReadFloat ()

  Reads a float from the stream.
- void ReadFloat (float *buffer, size_t size)

  Reads float data from the stream in a specified buffer.
- double ReadDouble ()

  Reads a double from the stream.
- void ReadDouble (double *buffer, size_t size)

  Reads double data from the stream in a specified buffer.
- wxString ReadString ()

  Reads a string from a stream.
- void SetConv (const wxMBConv &conv)

  Sets the text conversion class used for reading strings.
- void UseBasicPrecisions ()

  Disables the use of extended precision format for floating point numbers.
- void UseExtendedPrecision ()

  Explicitly request the use of extended precision for floating point numbers.

21.136.2 Constructor & Destructor Documentation

wxDataInputStream::wxDataInputStream (wxInputStream & *stream*, const wxMBConv & *conv* = `wxConvUTF8`)

Constructs a datastream object from an input stream.

Only read methods will be available.

Note that the *conv* parameter is only available in Unicode builds of wxWidgets.

Parameters

stream	The input stream.
conv	Charset conversion object used to decode strings in Unicode mode (see ReadString() for a detailed description). Note that you must not destroy *conv* before you destroy this wxDataInputStream instance!

wxDataInputStream::~wxDataInputStream ( )

Destroys the wxDataInputStream object.

21.136.3 Member Function Documentation

void wxDataInputStream::BigEndianOrdered (bool *be_order*)

If *be_order* is true, all data will be read in big-endian order, such as written by programs on a big endian architecture (e.g.

Sparc) or written by Java-Streams (which always use big-endian order).

wxMBConv* wxDataInputStream::GetConv ( ) const

Returns the current text conversion class used for reading strings.

wxUint16 wxDataInputStream::Read16 ( )

Reads a 16 bit unsigned integer from the stream.

void wxDataInputStream::Read16 (wxUint16 * *buffer*, size_t *size*)

Reads 16 bit unsigned integers from the stream in a specified buffer.

The number of 16 bit unsigned integers to read is specified by the *size* variable.

wxUint32 wxDataInputStream::Read32 ( )

Reads a 32 bit unsigned integer from the stream.

void wxDataInputStream::Read32 (wxUint32 * *buffer*, size_t *size*)

Reads 32 bit unsigned integers from the stream in a specified buffer.

The number of 32 bit unsigned integers to read is specified by the *size* variable.

wxUint64 wxDataInputStream::Read64 ( )

Reads a 64 bit unsigned integer from the stream.

void wxDataInputStream::Read64 (wxUint64 * *buffer*, size_t *size*)

Reads 64 bit unsigned integers from the stream in a specified buffer.

The number of 64 bit unsigned integers to read is specified by the *size* variable.

wxUint8 wxDataInputStream::Read8 ( )

Reads a single byte from the stream.

void wxDataInputStream::Read8 (wxUint8 * *buffer*, size_t *size*)

Reads bytes from the stream in a specified buffer.

The number of bytes to read is specified by the *size* variable.

double wxDataInputStream::ReadDouble ( )

Reads a double from the stream.

The expected format is either 80 bit extended precision or, if UseBasicPrecisions() had been called, standard IEEE 754 64 bit double precision.

void wxDataInputStream::ReadDouble (double * *buffer,* size_t *size*)

Reads double data from the stream in a specified buffer.

The number of doubles to read is specified by the *size* variable.

float wxDataInputStream::ReadFloat ( )

Reads a float from the stream.

Notice that if UseBasicPrecisions() hadn't been called, this function simply reads a double and truncates it to float as by default the same (80 bit extended precision) representation is used for both float and double values.

Since

2.9.5

void wxDataInputStream::ReadFloat (float * *buffer,* size_t *size*)

Reads float data from the stream in a specified buffer.

The number of floats to read is specified by the *size* variable.

Since

2.9.5

wxString wxDataInputStream::ReadString ( )

Reads a string from a stream.

Actually, this function first reads a long integer specifying the length of the string (without the last null character) and then reads the string.

In Unicode build of wxWidgets, the function first reads multibyte (char*) string from the stream and then converts it to Unicode using the *conv* object passed to constructor and returns the result as wxString. You are responsible for using the same converter as when writing the stream.

See also

wxDataOutputStream::WriteString()

void wxDataInputStream::SetConv (const wxMBConv & *conv*)

Sets the text conversion class used for reading strings.

void wxDataInputStream::UseBasicPrecisions ( )

Disables the use of extended precision format for floating point numbers.

This method disables the use of 80 bit extended precision format for the `float` and `double` values read from the stream, which is used by default (unless `wxUSE_APPLE_IEEE` was set to `0` when building the library, in which case the extended format support is not available at all and this function does nothing).

After calling it, `float` values will be expected to appear in one of IEEE 754 "basic formats", i.e. 32 bit single precision format for floats and 64 bit double precision format for doubles in the input.

Since

2.9.5

void wxDataInputStream::UseExtendedPrecision ( )

Explicitly request the use of extended precision for floating point numbers.

This function allows the application code to explicitly request the use of 80 bit extended precision format for the floating point numbers. This is the case by default but using this function explicitly ensures that the compilation of code relying on reading the input containing numbers in extended precision format would fail when using a version of wxWidgets compiled with `wxUSE_APPLE_IEEE==0` and so not supporting this format at all.

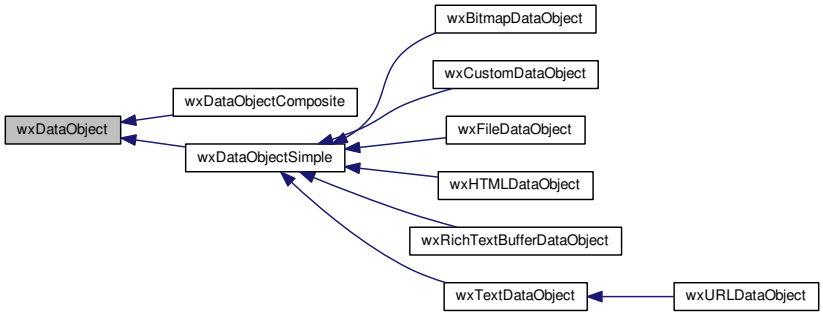
Since

2.9.5

21.137 wxDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxDataObject:



21.137.1 Detailed Description

A wxDataObject represents data that can be copied to or from the clipboard, or dragged and dropped.

The important thing about wxDataObject is that this is a 'smart' piece of data unlike 'dumb' data containers such as memory buffers or files. Being 'smart' here means that the data object itself should know what data formats it supports and how to render itself in each of its supported formats.

A supported format, incidentally, is exactly the format in which the data can be requested from a data object or from which the data object may be set. In the general case, an object may support different formats on 'input' and 'output', i.e. it may be able to render itself in a given format but not be created from data on this format or vice versa. wxDataObject defines the wxDataObject::Direction enumeration type which distinguishes between them.

See wxDataFormat documentation for more about formats.

Not surprisingly, being 'smart' comes at a price of added complexity. This is reasonable for the situations when you really need to support multiple formats, but may be annoying if you only want to do something simple like cut and paste text.

To provide a solution for both cases, wxWidgets has two predefined classes which derive from wxDataObject↩ : wxDataObjectSimple and wxDataObjectComposite. wxDataObjectSimple is the simplest wxDataObject possible

and only holds data in a single format (such as HTML or text) and wxDataObjectComposite is the simplest way to implement a wxDataObject that does support multiple formats because it achieves this by simply holding several wxDataObjectSimple objects.

So, you have several solutions when you need a wxDataObject class (and you need one as soon as you want to transfer data via the clipboard or drag and drop):

1. Use one of the built-in classes.
    - You may use wxTextDataObject, wxBitmapDataObject wxFileDataObject, wxURLDataObject in the simplest cases when you only need to support one format and your data is either text, bitmap or list of files.
2. Use wxDataObjectSimple
    - Deriving from wxDataObjectSimple is the simplest solution for custom data - you will only support one format and so probably won't be able to communicate with other programs, but data transfer will work in your program (or between different instances of it).
3. Use wxDataObjectComposite
    - This is a simple but powerful solution which allows you to support any number of formats (either standard or custom if you combine it with the previous solution).
4. Use wxDataObject directly
    - This is the solution for maximum flexibility and efficiency, but it is also the most difficult to implement.

Please note that the easiest way to use drag and drop and the clipboard with multiple formats is by using wxDataObjectComposite, but it is not the most efficient one as each wxDataObjectSimple would contain the whole data in its respective formats. Now imagine that you want to paste 200 pages of text in your proprietary format, as well as Word, RTF, HTML, Unicode and plain text to the clipboard and even today's computers are in trouble. For this case, you will have to derive from wxDataObject directly and make it enumerate its formats and provide the data in the requested format on demand.

Note that neither the GTK+ data transfer mechanisms for clipboard and drag and drop, nor OLE data transfer, *copies* any data until another application actually requests the data. This is in contrast to the 'feel' offered to the user of a program who would normally think that the data resides in the clipboard after having pressed 'Copy' - in reality it is only declared to be *available*.

You may also derive your own data object classes from wxCustomDataObject for user-defined types. The format of user-defined data is given as a mime-type string literal, such as "application/word" or "image/png". These strings are used as they are under Unix (so far only GTK+) to identify a format and are translated into their Windows equivalent under Win32 (using the OLE IDataObject for data exchange to and from the clipboard and for drag and drop). Note that the format string translation under Windows is not yet finished.

Each class derived directly from wxDataObject must override and implement all of its functions which are pure virtual in the base class. The data objects which only render their data or only set it (i.e. work in only one direction), should return 0 from GetFormatCount().

wxPerl Note: This class is not currently usable from wxPerl; you may use Wx::PlDataObjectSimple instead.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, Drag & Drop Sample, wxFileDataObject, wxTextDataObject, wxBitmapDataObject, wxCustomDataObject, wxDropTarget, wxDropSource, wxTextDropTarget, wxFileDropTarget

Public Types

- enum Direction {
  Get = 0x01,
  Set = 0x02,
  Both = 0x03 }

Public Member Functions

- wxDataObject ()

  Constructor.
- virtual ~wxDataObject ()

  Destructor.
- virtual void GetAllFormats (wxDataFormat *formats, Direction dir=Get) const =0

  Copies all formats supported in the given direction dir to the array pointed to by formats.
- virtual bool GetDataHere (const wxDataFormat &format, void *buf) const =0

  The method will write the data of the format format to the buffer buf.
- virtual size_t GetDataSize (const wxDataFormat &format) const =0

  Returns the data size of the given format format.
- virtual size_t GetFormatCount (Direction dir=Get) const =0

  Returns the number of available formats for rendering or setting the data.
- virtual wxDataFormat GetPreferredFormat (Direction dir=Get) const =0

  Returns the preferred format for either rendering the data (if dir is `Get`, its default value) or for setting it.
- virtual bool SetData (const wxDataFormat &format, size_t len, const void *buf)

  Set the data in the format format of the length len provided in the buffer buf.
- bool IsSupported (const wxDataFormat &format, Direction dir=Get) const

  Returns true if this format is supported.

21.137.2 Member Enumeration Documentation

enum wxDataObject::Direction

Enumerator

Get Format is supported by GetDataHere()

Set Format is supported by SetData()

Both Format is supported by both GetDataHere() and SetData() (unused currently)

21.137.3 Constructor & Destructor Documentation

wxDataObject::wxDataObject ( )

Constructor.

virtual wxDataObject::~wxDataObject ( ) [virtual]

Destructor.

21.137.4 Member Function Documentation

virtual void wxDataObject::GetAllFormats (wxDataFormat * *formats,* Direction *dir* = Get) const [pure virtual]

Copies all formats supported in the given direction *dir* to the array pointed to by *formats*.

There must be enough space for GetFormatCount(dir) formats in it.

wxPerl Note: In wxPerl this method only takes the *dir* parameter. In scalar context it returns the first format in the list, in list context it returns a list containing all the supported formats.

Implemented in wxTextDataObject.

virtual bool wxDataObject::GetDataHere (const wxDataFormat & *format,* void * *buf*) const [pure virtual]

The method will write the data of the format *format* to the buffer *buf*.

In other words, copy the data from this object in the given format to the supplied buffer. Returns true on success, false on failure.

Implemented in wxRichTextBufferDataObject.

virtual size_t wxDataObject::GetDataSize (const wxDataFormat & *format*) const [pure virtual]

Returns the data size of the given format *format*.

Implemented in wxRichTextBufferDataObject.

virtual size_t wxDataObject::GetFormatCount (Direction *dir* = Get) const [pure virtual]

Returns the number of available formats for rendering or setting the data.

Implemented in wxTextDataObject.

virtual wxDataFormat wxDataObject::GetPreferredFormat (Direction *dir* = Get) const [pure virtual]

Returns the preferred format for either rendering the data (if *dir* is `Get`, its default value) or for setting it.

Usually this will be the native format of the wxDataObject.

Implemented in wxRichTextBufferDataObject.

bool wxDataObject::IsSupported (const wxDataFormat & *format,* Direction *dir* = Get) const

Returns true if this format is supported.

virtual bool wxDataObject::SetData (const wxDataFormat & *format,* size_t *len,* const void * *buf*) [virtual]

Set the data in the format *format* of the length *len* provided in the buffer *buf*.

In other words, copy length bytes of data from the buffer to this data object.

Parameters

format	The format for which to set the data.

len	The size of data in bytes.
buf	Non-NULL pointer to the data.

Returns

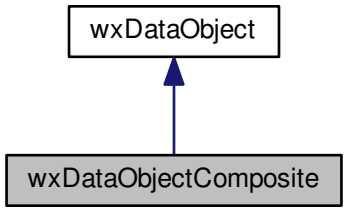
true on success, false on failure.

Reimplemented in wxRichTextBufferDataObject.

21.138 wxDataObjectComposite Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxDataObjectComposite:



21.138.1 Detailed Description

wxDataObjectComposite is the simplest wxDataObject derivation which may be used to support multiple formats.

It contains several wxDataObjectSimple objects and supports any format supported by at least one of them. Only one of these data objects is *preferred* (the first one if not explicitly changed by using the second parameter of Add()) and its format determines the preferred format of the composite data object as well.

See wxDataObject documentation for the reasons why you might prefer to use wxDataObject directly instead of wxDataObjectComposite for efficiency reasons.

This example shows how a composite data object capable of storing either bitmaps or file names (presumably of bitmap files) can be initialized and used:

```
MyDropTarget::MyDropTarget()
{
    wxDataObjectComposite* dataobj = new
      wxDataObjectComposite();
    dataobj->Add(new wxBitmapDataObject(), true);
    dataobj->Add(new wxFileDataObject());
    SetDataObject(dataobj);
}

wxDragResult MyDropTarget::OnData(wxCoord x, wxCoord y,
                                  wxDragResult defaultDragResult)
{
    wxDragResult dragResult = wxDropTarget::OnData(x, y, defaultDragResult)
      ;
    if ( dragResult == defaultDragResult )
    {
        wxDataObjectComposite *
            dataobjComp = static_cast<wxDataObjectComposite *>(GetDataObject());

        wxDataFormat format = dataObjects->GetReceivedFormat();
```

```
        wxDataObject *dataobj = dataobjComp->GetObject(format);
        switch ( format.GetType() )
        {
            case wxDF_BITMAP:
                {
                    wxBitmapDataObject *
                        dataobjBitmap = static_cast<wxBitmapDataObject *>(dataobj);

                    ... use dataobj->GetBitmap() ...
                }
                break;

            case wxDF_FILENAME:
                {
                    wxFileDataObject *
                        dataobjFile = static_cast<wxFileDataObject *>(dataobj);

                    ... use dataobj->GetFilenames() ...
                }
                break;

            default:
                wxFAIL_MSG( "unexpected data object format" );
        }
    }

    return dragResult;
}
```

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject, wxDataObjectSimple, wxFileDataObject, wxTextDataObject, wxBitmapDataObject

Public Member Functions

- wxDataObjectComposite ()

  The default constructor.
- void Add (wxDataObjectSimple *dataObject, bool preferred=false)

  Adds the dataObject to the list of supported objects and it becomes the preferred object if preferred is true.
- wxDataFormat GetReceivedFormat () const

  Report the format passed to the SetData() method.
- wxDataObjectSimple * GetObject (const wxDataFormat &format, wxDataObject::Direction dir=wxDataObject::Get) const

  Returns the pointer to the object which supports the passed format for the specified direction.

Additional Inherited Members

21.138.2 Constructor & Destructor Documentation

wxDataObjectComposite::wxDataObjectComposite ( )

The default constructor.

21.138.3 Member Function Documentation

void wxDataObjectComposite::Add (wxDataObjectSimple * *dataObject*, bool *preferred* = `false`)

Adds the *dataObject* to the list of supported objects and it becomes the preferred object if *preferred* is true.

wxDataObjectSimple* wxDataObjectComposite::GetObject (const wxDataFormat & *format*, wxDataObject::Direction *dir* = wxDataObject::Get) const

Returns the pointer to the object which supports the passed format for the specified direction.

NULL is returned if the specified *format* is not supported for this direction *dir*. The returned pointer is owned by wxDataObjectComposite itself and shouldn't be deleted by caller.

Since

2.9.1

wxDataFormat wxDataObjectComposite::GetReceivedFormat ( ) const

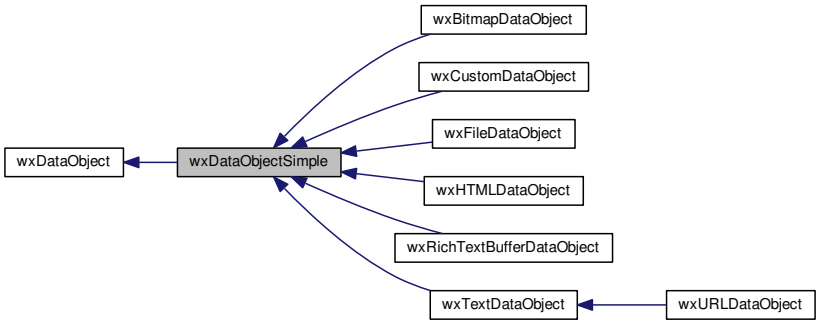
Report the format passed to the SetData() method.

This should be the format of the data object within the composite that received data from the clipboard or the DnD operation. You can use this method to find out what kind of data object was received.

21.139 wxDataObjectSimple Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxDataObjectSimple:



21.139.1 Detailed Description

This is the simplest possible implementation of the wxDataObject class.

The data object of (a class derived from) this class only supports **one format**, so the number of virtual functions to be implemented is reduced.

Notice that this is still an abstract base class and cannot be used directly, it must be derived. The objects supporting rendering the data must override GetDataSize() and GetDataHere() while the objects which may be set must override SetData(). Of course, the objects supporting both operations must override all three methods.

wxPerl Note: In wxPerl, you need to derive your data object class from Wx::PlDataObjectSimple.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, Drag & Drop Sample, wxFileDataObject, wxTextDataObject, wxBitmapDataObject

Public Member Functions

- wxDataObjectSimple (const wxDataFormat &format=wxFormatInvalid)

  Constructor accepts the supported format (none by default) which may also be set later with SetFormat().
- virtual bool GetDataHere (void ∗buf) const

  Copy the data to the buffer, return true on success.
- virtual size_t GetDataSize () const

  Gets the size of our data.
- const wxDataFormat & GetFormat () const

  Returns the (one and only one) format supported by this object.
- virtual bool SetData (size_t len, const void ∗buf)

  Copy the data from the buffer, return true on success.
- void SetFormat (const wxDataFormat &format)

  Sets the supported format.

Additional Inherited Members

21.139.2 Constructor & Destructor Documentation

wxDataObjectSimple::wxDataObjectSimple (const wxDataFormat & *format* = wxFormatInvalid)

Constructor accepts the supported format (none by default) which may also be set later with SetFormat().

21.139.3 Member Function Documentation

virtual bool wxDataObjectSimple::GetDataHere (void ∗ *buf*) const `[virtual]`

Copy the data to the buffer, return true on success.

Must be implemented in the derived class if the object supports rendering its data.

Reimplemented in wxRichTextBufferDataObject.

virtual size_t wxDataObjectSimple::GetDataSize ( ) const `[virtual]`

Gets the size of our data.

Must be implemented in the derived class if the object supports rendering its data.

Reimplemented in wxRichTextBufferDataObject.

const wxDataFormat& wxDataObjectSimple::GetFormat ( ) const

Returns the (one and only one) format supported by this object.

It is assumed that the format is supported in both directions.

virtual bool wxDataObjectSimple::SetData (size_t *len,* const void * *buf*) `[virtual]`

Copy the data from the buffer, return true on success.

Must be implemented in the derived class if the object supports setting its data.

Reimplemented in wxRichTextBufferDataObject, and wxCustomDataObject.

void wxDataObjectSimple::SetFormat (const wxDataFormat & *format*)

Sets the supported format.

21.140 wxDataOutputStream Class Reference

`#include <wx/datstrm.h>`

21.140.1 Detailed Description

This class provides functions that write binary data types in a portable way.

Data can be written in either big-endian or little-endian format, little-endian being the default on all architectures but BigEndianOrdered() can be used to change this. The default format for the floating point types is 80 bit "extended precision" unless `wxUSE_APPLE_IEEE` was turned off during the library compilation, in which case extended precision is not available at all. You can call UseBasicPrecisions() to change this and use the standard IEEE 754 32 bit single precision format for floats and standard 64 bit double precision format for doubles. This is recommended for the new code for better interoperability with other software that typically uses standard IEEE 754 formats for its data, the use of extended precision by default is solely due to backwards compatibility.

If you want to write data to text files (or streams) use wxTextOutputStream instead.

The "<<" operator is overloaded and you can use this class like a standard C++ iostream. See wxDataInputStream for its usage and caveats.

Library: wxBase

Category: Streams

See also

wxDataInputStream

Public Member Functions

- wxDataOutputStream (wxOutputStream &stream, const wxMBConv &conv=wxConvUTF8)

  Constructs a datastream object from an output stream.
- ~wxDataOutputStream ()

  Destroys the wxDataOutputStream object.
- void BigEndianOrdered (bool be_order)

  If be_order is true, all data will be written in big-endian order, e.g.
- wxMBConv * GetConv () const

  Returns the current text conversion class used for writing strings.
- void SetConv (const wxMBConv &conv)

  Sets the text conversion class used for writing strings.

- void UseBasicPrecisions ()

  Disables the use of extended precision format for floating point numbers.
- void UseExtendedPrecision ()

  Explicitly request the use of extended precision for floating point numbers.
- void Write8 (wxUint8 i8)

  Writes the single byte i8 to the stream.
- void Write8 (const wxUint8 *buffer, size_t size)

  Writes an array of bytes to the stream.
- void Write16 (wxUint16 i16)

  Writes the 16 bit unsigned integer i16 to the stream.
- void Write16 (const wxUint16 *buffer, size_t size)

  Writes an array of 16 bit unsigned integer to the stream.
- void Write32 (wxUint32 i32)

  Writes the 32 bit unsigned integer i32 to the stream.
- void Write32 (const wxUint32 *buffer, size_t size)

  Writes an array of 32 bit unsigned integer to the stream.
- void Write64 (wxUint64 i64)

  Writes the 64 bit unsigned integer i64 to the stream.
- void Write64 (const wxUint64 *buffer, size_t size)

  Writes an array of 64 bit unsigned integer to the stream.
- void WriteFloat (float f)

  Writes the float f to the stream.
- void WriteFloat (const float *buffer, size_t size)

  Writes an array of float to the stream.
- void WriteDouble (double d)

  Writes the double d to the stream.
- void WriteDouble (const double *buffer, size_t size)

  Writes an array of double to the stream.
- void WriteString (const wxString &string)

  Writes string to the stream.

21.140.2 Constructor & Destructor Documentation

wxDataOutputStream::wxDataOutputStream (wxOutputStream & *stream*, const wxMBConv & *conv* = wxConvUTF8)

Constructs a datastream object from an output stream.

Only write methods will be available.

Note that the *conv* parameter is only available in Unicode builds of wxWidgets.

Parameters

stream	The output stream.
conv	Charset conversion object used to encoding Unicode strings before writing them to the stream in Unicode mode (see WriteString() for a detailed description). Note that you must not destroy *conv* before you destroy this wxDataOutputStream instance! It is recommended to use the default value (UTF-8).

wxDataOutputStream::~wxDataOutputStream ( )

Destroys the wxDataOutputStream object.

21.140.3 Member Function Documentation

void wxDataOutputStream::BigEndianOrdered (bool *be_order*)

If *be_order* is true, all data will be written in big-endian order, e.g.

for reading on a Sparc or from Java-Streams (which always use big-endian order), otherwise data will be written in little-endian order.

wxMBConv* wxDataOutputStream::GetConv ( ) const

Returns the current text conversion class used for writing strings.

void wxDataOutputStream::SetConv (const wxMBConv & *conv*)

Sets the text conversion class used for writing strings.

void wxDataOutputStream::UseBasicPrecisions ( )

Disables the use of extended precision format for floating point numbers.

This method disables the use of 80 bit extended precision format for the `float` and `double` values written to the stream, which is used by default (unless `wxUSE_APPLE_IEEE` was set to `0` when building the library, in which case the extended format support is not available at all and this function does nothing).

After calling it, `float` values will be written out in one of IEEE 754 "basic formats", i.e. 32 bit single precision format for floats and 64 bit double precision format for doubles.

Since

2.9.5

void wxDataOutputStream::UseExtendedPrecision ( )

Explicitly request the use of extended precision for floating point numbers.

This function allows the application code to explicitly request the use of 80 bit extended precision format for the floating point numbers. This is the case by default but using this function explicitly ensures that the compilation of code relying on producing the output stream using extended precision would fail when using a version of wxWidgets compiled with `wxUSE_APPLE_IEEE==0` and so not supporting this format at all.

Since

2.9.5

void wxDataOutputStream::Write16 (wxUint16 *i16*)

Writes the 16 bit unsigned integer *i16* to the stream.

void wxDataOutputStream::Write16 (const wxUint16 * *buffer*, size_t *size*)

Writes an array of 16 bit unsigned integer to the stream.

The number of 16 bit unsigned integer to write is specified with the *size* variable.

void wxDataOutputStream::Write32 (wxUint32 *i32*)

Writes the 32 bit unsigned integer *i32* to the stream.

void wxDataOutputStream::Write32 (const wxUint32 * *buffer*, size_t *size*)

Writes an array of 32 bit unsigned integer to the stream.

The number of 32 bit unsigned integer to write is specified with the *size* variable.

void wxDataOutputStream::Write64 (wxUint64 *i64*)

Writes the 64 bit unsigned integer *i64* to the stream.

void wxDataOutputStream::Write64 (const wxUint64 * *buffer*, size_t *size*)

Writes an array of 64 bit unsigned integer to the stream.

The number of 64 bit unsigned integer to write is specified with the *size* variable.

void wxDataOutputStream::Write8 (wxUint8 *i8*)

Writes the single byte *i8* to the stream.

void wxDataOutputStream::Write8 (const wxUint8 * *buffer*, size_t *size*)

Writes an array of bytes to the stream.

The number of bytes to write is specified with the *size* variable.

void wxDataOutputStream::WriteDouble (double *d*)

Writes the double *d* to the stream.

The output format is either 80 bit extended precision or, if UseBasicPrecisions() had been called, standard IEEE 754 64 bit double precision.

void wxDataOutputStream::WriteDouble (const double * *buffer*, size_t *size*)

Writes an array of double to the stream.

The number of doubles to write is specified by the *size* variable.

void wxDataOutputStream::WriteFloat (float *f*)

Writes the float *f* to the stream.

If UseBasicPrecisions() had been called, the value is written out using the standard IEEE 754 32 bit single precision format. Otherwise, this method uses the same format as WriteDouble(), i.e. 80 bit extended precision representation.

Since

2.9.5

void wxDataOutputStream::WriteFloat (const float * *buffer*, size_t *size*)

Writes an array of float to the stream.

The number of floats to write is specified by the *size* variable.

Since

2.9.5

void wxDataOutputStream::WriteString (const wxString & *string*)

Writes *string* to the stream.

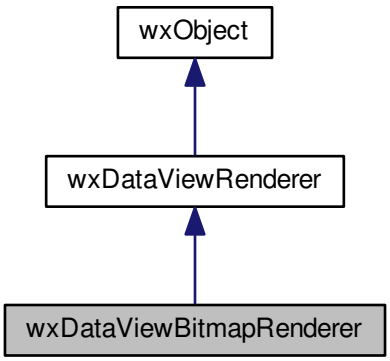
Actually, this method writes the size of the string before writing *string* itself.

In ANSI build of wxWidgets, the string is written to the stream in exactly same way it is represented in memory. In Unicode build, however, the string is first converted to multibyte representation with *conv* object passed to stream's constructor (consequently, ANSI applications can read data written by Unicode application, as long as they agree on encoding) and this representation is written to the stream. UTF-8 is used by default.

21.141 wxDataViewBitmapRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewBitmapRenderer:



21.141.1 Detailed Description

This class is used by wxDataViewCtrl to render bitmap controls.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewBitmapRenderer (const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Additional Inherited Members

21.141.2 Constructor & Destructor Documentation

wxDataViewBitmapRenderer::wxDataViewBitmapRenderer (const wxString & *varianttype* = GetDefaultType (), wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.141.3 Member Function Documentation

static wxString wxDataViewBitmapRenderer::GetDefaultType ( ) `[static]`

Returns the wxVariant type used with this renderer.

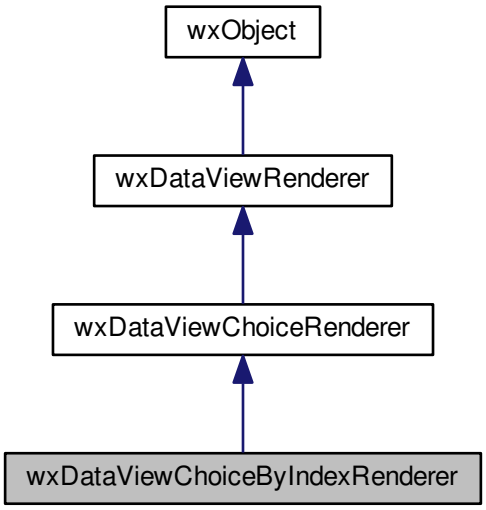
Since

3.1.0

21.142 wxDataViewChoiceByIndexRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewChoiceByIndexRenderer:



21.142.1 Detailed Description

A wxDataViewCtrl renderer using wxChoice control and indexes into it.

Unlike its base wxDataViewChoiceRenderer class, this one stores the choice index, i.e. an `int`, in the variant used by its SetValue() and GetValue().

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewChoiceByIndexRenderer (const wxArrayString &choices, wxDataViewCellMode mode=wxDATAVIEW_CELL_EDITABLE, int alignment=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Additional Inherited Members

21.142.2 Constructor & Destructor Documentation

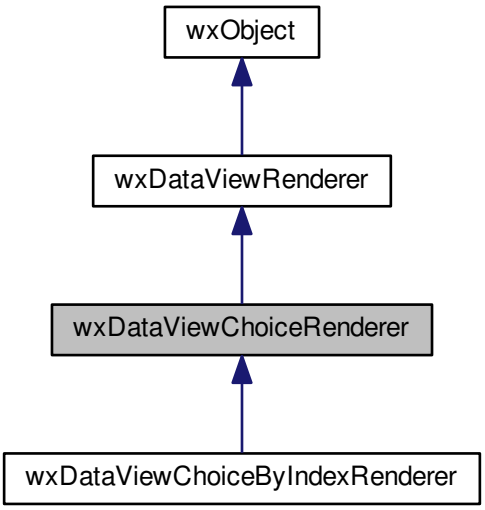
wxDataViewChoiceByIndexRenderer::wxDataViewChoiceByIndexRenderer (const wxArrayString & *choices*, wxDataViewCellMode *mode* = wxDATAVIEW_CELL_EDITABLE, int *alignment* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.143 wxDataViewChoiceRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewChoiceRenderer:



21.143.1 Detailed Description

A wxDataViewCtrl renderer using wxChoice control and values of strings in it.

This class is used by wxDataViewCtrl to render choice controls. It stores a string so that SetValue() and GetValue() operate on a variant holding a string.

See also

wxDataViewChoiceByIndexRenderer

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewChoiceRenderer (const wxArrayString &choices, wxDataViewCellMode mode=wxDATAVIEW_CELL_EDITABLE, int alignment=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.
- wxString GetChoice (size_t index) const

  Returns the choice referred to by index.
- const wxArrayString & GetChoices () const

  Returns all choices.

Additional Inherited Members

21.143.2 Constructor & Destructor Documentation

wxDataViewChoiceRenderer::wxDataViewChoiceRenderer (const wxArrayString & *choices*, wxDataViewCellMode *mode* = wxDATAVIEW_CELL_EDITABLE, int *alignment* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.143.3 Member Function Documentation

wxString wxDataViewChoiceRenderer::GetChoice (size_t *index*) const

Returns the choice referred to by index.

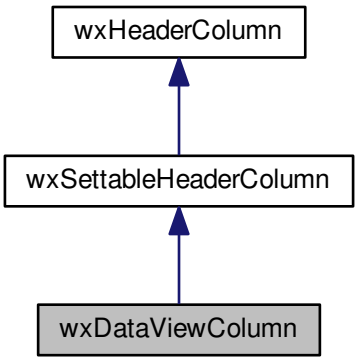
const wxArrayString& wxDataViewChoiceRenderer::GetChoices ( ) const

Returns all choices.

21.144 wxDataViewColumn Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewColumn:



21.144.1 Detailed Description

This class represents a column in a wxDataViewCtrl.

One wxDataViewColumn is bound to one column in the data model to which the wxDataViewCtrl has been associated.

An instance of wxDataViewRenderer is used by this class to render its data.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewColumn (const wxString &title, wxDataViewRenderer *renderer, unsigned int model_column, int width=wxDVC_DEFAULT_WIDTH, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Constructs a text column.
- wxDataViewColumn (const wxBitmap &bitmap, wxDataViewRenderer *renderer, unsigned int model_column, int width=wxDVC_DEFAULT_WIDTH, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Constructs a bitmap column.
- unsigned int GetModelColumn () const

  Returns the index of the column of the model, which this wxDataViewColumn is displaying.
- wxDataViewCtrl * GetOwner () const

  Returns the owning wxDataViewCtrl.
- wxDataViewRenderer * GetRenderer () const

  Returns the renderer of this wxDataViewColumn.

21.144.2 Constructor & Destructor Documentation

wxDataViewColumn::wxDataViewColumn (const wxString & *title*, wxDataViewRenderer * *renderer*, unsigned int *model_column*, int *width* = wxDVC_DEFAULT_WIDTH, wxAlignment *align* = wxALIGN_CENTER, int *flags* = wxDATAVIEW_COL_RESIZABLE)

Constructs a text column.

Parameters

title	The title of the column.
renderer	The class which will render the contents of this column.
model_column	The index of the model's column which is associated with this object.
width	The width of the column. The `wxDVC_DEFAULT_WIDTH` value is the fixed default value.
align	The alignment of the column title.
flags	One or more flags of the wxDataViewColumnFlags enumeration.

wxDataViewColumn::wxDataViewColumn (const wxBitmap & *bitmap*, wxDataViewRenderer * *renderer*, unsigned int *model_column*, int *width* = wxDVC_DEFAULT_WIDTH, wxAlignment *align* = wxALIGN_CENTER, int *flags* = wxDATAVIEW_COL_RESIZABLE)

Constructs a bitmap column.

Parameters

bitmap	The bitmap of the column.
renderer	The class which will render the contents of this column.
model_column	The index of the model's column which is associated with this object.
width	The width of the column. The `wxDVC_DEFAULT_WIDTH` value is the fixed default value.
align	The alignment of the column title.
flags	One or more flags of the wxDataViewColumnFlags enumeration.

21.144.3 Member Function Documentation

unsigned int wxDataViewColumn::GetModelColumn ( ) const

Returns the index of the column of the model, which this wxDataViewColumn is displaying.

wxDataViewCtrl* wxDataViewColumn::GetOwner ( ) const

Returns the owning wxDataViewCtrl.

wxDataViewRenderer* wxDataViewColumn::GetRenderer ( ) const

Returns the renderer of this wxDataViewColumn.

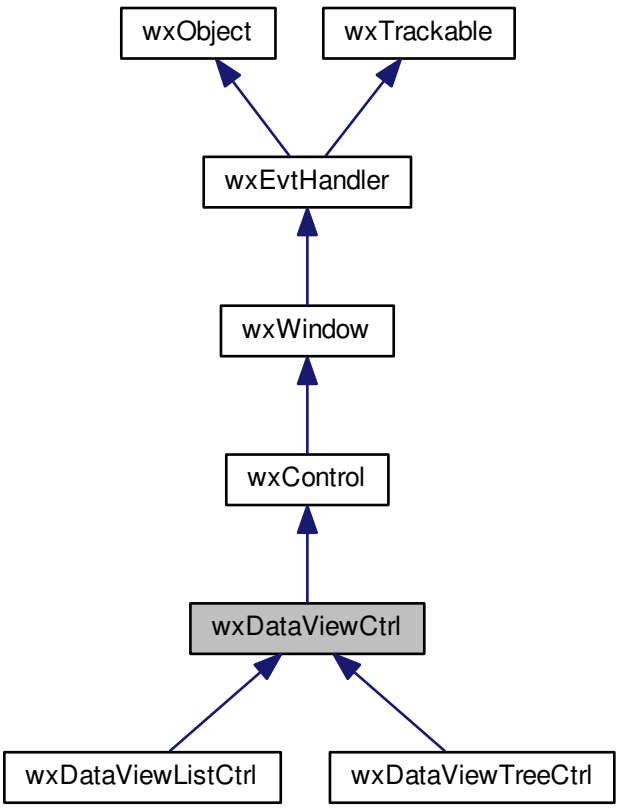
See also

wxDataViewRenderer.

21.145 wxDataViewCtrl Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewCtrl:



21.145.1 Detailed Description

wxDataViewCtrl is a control to display data either in a tree like fashion or in a tabular form or both.

If you only need to display a simple tree structure with an API more like the older wxTreeCtrl class, then the specialized wxDataViewTreeCtrl can be used. Likewise, if you only want to display simple table structure you can use the specialized wxDataViewListCtrl class. Both wxDataViewTreeCtrl and wxDataViewListCtrl can be used without defining your own wxDataViewModel.

A wxDataViewItem is used to represent a (visible) item in the control.

Unlike wxListCtrl, wxDataViewCtrl doesn't get its data from the user through virtual functions or by setting it directly. Instead you need to write your own wxDataViewModel and associate it with this control. Then you need to add a number of wxDataViewColumn to this control to define what each column shall display. Each wxDataViewColumn in turn owns 1 instance of a wxDataViewRenderer to render its cells.

A number of standard renderers for rendering text, dates, images, toggle, a progress bar etc. are provided. Additionally, the user can write custom renderers deriving from wxDataViewCustomRenderer for displaying anything.

All data transfer from the control to the model and the user code is done through wxVariant which can be extended to support more data formats as necessary. Accordingly, all type information uses the strings returned from wxVariant::GetType.

This control supports single column sorting and on some platforms (currently only those using the generic version, i.e. not wxGTK nor wxOSX) also sorting by multiple columns at once. The latter must be explicitly enabled using AllowMultiColumnSort(), which will also indicate whether this feature is supported, as it changes the default behaviour of right clicking the column header to add or remove it to the set of columns used for sorting. If this behaviour is not appropriate, you may handle `wxEVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK` event yourself to prevent it from happening. In this case you would presumably call ToggleSortByColumn() from some other event handler to still allow the user to configure sort order somehow.

Styles

This class supports the following styles:

- wxDV_SINGLE: Single selection mode. This is the default.
- wxDV_MULTIPLE: Multiple selection mode.
- wxDV_ROW_LINES: Use alternating colours for rows if supported by platform and theme. Currently only supported by the native GTK and OS X implementations but not by the generic one.
- wxDV_HORIZ_RULES: Display the separator lines between rows.
- wxDV_VERT_RULES: Display the separator lines between columns.
- wxDV_VARIABLE_LINE_HEIGHT: Allow variable line heights. This can be inefficient when displaying large number of items.
- wxDV_NO_HEADER: Do not show column headers (which are shown by default).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxDataViewEvent& event)

Event macros for events emitted by this class:

- EVT_DATAVIEW_SELECTION_CHANGED(id, func): Process a `wxEVT_DATAVIEW_SELECTION_CHANGED` event.

- EVT_DATAVIEW_ITEM_ACTIVATED(id, func): Process a `wxEVT_DATAVIEW_ITEM_ACTIVATED` event. This event is triggered by double clicking an item or pressing some special key (usually "Enter") when it is focused.
- EVT_DATAVIEW_ITEM_START_EDITING(id, func): Process a `wxEVT_DATAVIEW_ITEM_START_EDITING` event. This event can be vetoed in order to prevent editing on an item by item basis.
- EVT_DATAVIEW_ITEM_EDITING_STARTED(id, func): Process a `wxEVT_DATAVIEW_ITEM_EDITING_STARTED` event.
- EVT_DATAVIEW_ITEM_EDITING_DONE(id, func): Process a `wxEVT_DATAVIEW_ITEM_EDITING_DONE` event.
- EVT_DATAVIEW_ITEM_COLLAPSING(id, func): Process a `wxEVT_DATAVIEW_ITEM_COLLAPSING` event.
- EVT_DATAVIEW_ITEM_COLLAPSED(id, func): Process a `wxEVT_DATAVIEW_ITEM_COLLAPSED` event.
- EVT_DATAVIEW_ITEM_EXPANDING(id, func): Process a `wxEVT_DATAVIEW_ITEM_EXPANDING` event.
- EVT_DATAVIEW_ITEM_EXPANDED(id, func): Process a `wxEVT_DATAVIEW_ITEM_EXPANDED` event.
- EVT_DATAVIEW_ITEM_VALUE_CHANGED(id, func): Process a `wxEVT_DATAVIEW_ITEM_VALUE_CHANGED` event.
- EVT_DATAVIEW_ITEM_CONTEXT_MENU(id, func): Process a `wxEVT_DATAVIEW_ITEM_CONTEXT_MENU` event generated when the user right clicks inside the control. Notice that this menu is generated even if the click didn't occur on any valid item, in this case wxDataViewEvent::GetItem() simply returns an invalid item.
- EVT_DATAVIEW_COLUMN_HEADER_CLICK(id, func): Process a `wxEVT_DATAVIEW_COLUMN_HEADER_CLICK` event.
- EVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK(id, func): Process a `wxEVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK` event. Notice that currently this event is not generated in the native OS X versions of the control.
- EVT_DATAVIEW_COLUMN_SORTED(id, func): Process a `wxEVT_DATAVIEW_COLUMN_SORTED` event.
- EVT_DATAVIEW_COLUMN_REORDERED(id, func): Process a `wxEVT_DATAVIEW_COLUMN_REORDERED` event.
- EVT_DATAVIEW_ITEM_BEGIN_DRAG(id, func): Process a `wxEVT_DATAVIEW_ITEM_BEGIN_DRAG` event.
- EVT_DATAVIEW_ITEM_DROP_POSSIBLE(id, func): Process a `wxEVT_DATAVIEW_ITEM_DROP_POSSIBLE` event.
- EVT_DATAVIEW_ITEM_DROP(id, func): Process a `wxEVT_DATAVIEW_ITEM_DROP` event.

Notice that this control doesn't allow to process generic mouse events such as `wxEVT_LEFT_DOWN` in all ports (notably it doesn't work in wxGTK). If you need to handle any mouse events not covered by the ones above, consider using a custom renderer for the cells that must handle them.

Library: wxAdvanced

Category: Controls, wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewCtrl ()

  Default Constructor.
- wxDataViewCtrl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxDataViewCtrlNameStr)

  Constructor.
- virtual ~wxDataViewCtrl ()

  Destructor.
- bool AllowMultiColumnSort (bool allow)

  Call to allow using multiple columns for sorting.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxDataViewCtrlNameStr)

  Create the control.
- virtual bool AppendColumn (wxDataViewColumn *col)

  Appends a wxDataViewColumn to the control.
- virtual bool PrependColumn (wxDataViewColumn *col)

  Prepends a wxDataViewColumn to the control.
- virtual bool InsertColumn (unsigned int pos, wxDataViewColumn *col)

  Inserts a wxDataViewColumn to the control.
- virtual bool AssociateModel (wxDataViewModel *model)

  Associates a wxDataViewModel with the control.
- virtual bool ClearColumns ()

  Removes all columns.
- virtual void Collapse (const wxDataViewItem &item)

  Collapses the item.
- virtual bool DeleteColumn (wxDataViewColumn *column)

  Deletes given column.
- virtual void EditItem (const wxDataViewItem &item, const wxDataViewColumn *column)

  Programmatically starts editing given cell of item.
- virtual bool EnableDragSource (const wxDataFormat &format)

  Enable drag operations using the given format.
- virtual bool EnableDropTarget (const wxDataFormat &format)

  Enable drop operations using the given format.
- virtual void EnsureVisible (const wxDataViewItem &item, const wxDataViewColumn *column=NULL)

  Call this to ensure that the given item is visible.
- virtual void Expand (const wxDataViewItem &item)

  Expands the item.
- virtual void ExpandAncestors (const wxDataViewItem &item)

  Expands all ancestors of the item.
- virtual wxDataViewColumn * GetColumn (unsigned int pos) const

  Returns pointer to the column.
- virtual unsigned int GetColumnCount () const

  Returns the number of columns.
- virtual int GetColumnPosition (const wxDataViewColumn *column) const

  Returns the position of the column or -1 if not found in the control.
- wxDataViewColumn * GetExpanderColumn () const

  Returns column containing the expanders.
- wxDataViewItem GetCurrentItem () const

Returns the currently focused item.

- wxDataViewColumn * GetCurrentColumn () const

  Returns the column that currently has focus.

- int GetIndent () const

  Returns indentation.

- virtual wxRect GetItemRect (const wxDataViewItem &item, const wxDataViewColumn *col=NULL) const

  Returns item rectangle.

- wxDataViewModel * GetModel ()

  Returns pointer to the data model associated with the control (if any).

- virtual int GetSelectedItemsCount () const

  Returns the number of currently selected items.

- virtual wxDataViewItem GetSelection () const

  Returns first selected item or an invalid item if none is selected.

- virtual int GetSelections (wxDataViewItemArray &sel) const

  Fills sel with currently selected items and returns their number.

- virtual wxDataViewColumn * GetSortingColumn () const

  Returns the wxDataViewColumn currently responsible for sorting or NULL if none has been selected.

- virtual wxVector
  < wxDataViewColumn * > GetSortingColumns () const

  Returns the columns which should be used for sorting the data in this control.

- bool HasSelection () const

  Returns true if any items are currently selected.

- virtual void HitTest (const wxPoint &point, wxDataViewItem &item, wxDataViewColumn *&col) const

  Hittest.

- virtual bool IsExpanded (const wxDataViewItem &item) const

  Return true if the item is expanded.

- bool IsMultiColumnSortAllowed () const

  Return true if using more than one column for sorting is allowed.

- virtual bool IsSelected (const wxDataViewItem &item) const

  Return true if the item is selected.

- virtual void Select (const wxDataViewItem &item)

  Select the given item.

- virtual void SelectAll ()

  Select all items.

- void SetExpanderColumn (wxDataViewColumn *col)

  Set which column shall contain the tree-like expanders.

- void SetCurrentItem (const wxDataViewItem &item)

  Changes the currently focused item.

- void SetIndent (int indent)

  Sets the indentation.

- virtual void SetSelections (const wxDataViewItemArray &sel)

  Sets the selection to the array of wxDataViewItems.

- virtual void Unselect (const wxDataViewItem &item)

  Unselect the given item.

- virtual void UnselectAll ()

  Unselect all item.

- virtual bool SetRowHeight (int rowHeight)

  Sets the row height.

- virtual void ToggleSortByColumn (int column)

  Toggle sorting by the given column.

- wxDataViewColumn * AppendBitmapColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a bitmap.
- wxDataViewColumn * AppendBitmapColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a bitmap.

- wxDataViewColumn * PrependBitmapColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a bitmap.
- wxDataViewColumn * PrependBitmapColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a bitmap.

- wxDataViewColumn * AppendDateColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_ACTIVATABLE, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a date.
- wxDataViewColumn * AppendDateColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_ACTIVATABLE, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a date.

- wxDataViewColumn * PrependDateColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_ACTIVATABLE, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a date.
- wxDataViewColumn * PrependDateColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_ACTIVATABLE, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a date.

- wxDataViewColumn * AppendIconTextColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering text with an icon.
- wxDataViewColumn * AppendIconTextColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering text with an icon.

- wxDataViewColumn * PrependIconTextColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering text with an icon.
- wxDataViewColumn * PrependIconTextColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering text with an icon.

- wxDataViewColumn * AppendProgressColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=80, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a progress indicator.
- wxDataViewColumn * AppendProgressColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=80, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a progress indicator.

- wxDataViewColumn * PrependProgressColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=80, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a progress indicator.
- wxDataViewColumn * PrependProgressColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=80, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a progress indicator.

- wxDataViewColumn * AppendTextColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering text.
- wxDataViewColumn * AppendTextColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering text.

- wxDataViewColumn * PrependTextColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering text.
- wxDataViewColumn * PrependTextColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_NOT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering text.

- wxDataViewColumn * AppendToggleColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=30, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a toggle.
- wxDataViewColumn * AppendToggleColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=30, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a column for rendering a toggle.

- wxDataViewColumn * PrependToggleColumn (const wxString &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=30, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a toggle.
- wxDataViewColumn * PrependToggleColumn (const wxBitmap &label, unsigned int model_column, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=30, wxAlignment align=wxALIGN_CENTER, int flags=wxDATAVIEW_COL_RESIZABLE)

  Prepends a column for rendering a toggle.

Additional Inherited Members

21.145.2 Constructor & Destructor Documentation

wxDataViewCtrl::wxDataViewCtrl ( )

Default Constructor.

wxDataViewCtrl::wxDataViewCtrl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxDataViewCtrlNameStr)

Constructor.

Calls Create().

virtual wxDataViewCtrl::~wxDataViewCtrl ( ) [virtual]

Destructor.

21.145.3 Member Function Documentation

bool wxDataViewCtrl::AllowMultiColumnSort (bool *allow*)

Call to allow using multiple columns for sorting.

When using multiple column for sorting, GetSortingColumns() method should be used to retrieve all the columns which should be used to effectively sort the data when processing the sorted event.

Currently multiple column sort is only implemented in the generic version, i.e. this functionality is not available when using the native wxDataViewCtrl implementation in wxGTK nor wxOSX.

Returns

true if sorting by multiple columns could be enabled, false otherwise, typically because this feature is not supported.

Since

3.1.0

wxDataViewColumn* wxDataViewCtrl::AppendBitmapColumn (const wxString & *label*, unsigned int *model_column*, wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *width* = -1, wxAlignment *align* = wxALIGN_CENTER, int *flags* = wxDATAVIEW_COL_RESIZABLE)

Appends a column for rendering a bitmap.

Returns the wxDataViewColumn created in the function or NULL on failure.

wxDataViewColumn* wxDataViewCtrl::AppendBitmapColumn (const wxBitmap & *label*, unsigned int *model_column*, wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *width* = -1, wxAlignment *align* = wxALIGN_CENTER, int *flags* = wxDATAVIEW_COL_RESIZABLE)

Appends a column for rendering a bitmap.

Returns the wxDataViewColumn created in the function or NULL on failure.

virtual bool wxDataViewCtrl::AppendColumn (**wxDataViewColumn** * *col*) [virtual]

Appends a wxDataViewColumn to the control.

Returns true on success.

Note that there is a number of short cut methods which implicitly create a wxDataViewColumn and a wxDataViewRenderer for it (see below).

Reimplemented in wxDataViewListCtrl.

wxDataViewColumn* wxDataViewCtrl::AppendDateColumn (const **wxString** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_ACTIVATABLE**, int *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering a date.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendDateColumn (const **wxBitmap** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_ACTIVATABLE**, int *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering a date.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendIconTextColumn (const **wxString** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering text with an icon.

Returns the wxDataViewColumn created in the function or NULL on failure. This method uses the wxDataViewIconTextRenderer class.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendIconTextColumn (const **wxBitmap** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering text with an icon.

Returns the wxDataViewColumn created in the function or NULL on failure. This method uses the wxDataViewIconTextRenderer class.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendProgressColumn (**const wxString &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = 80, **wxAlignment** *align* = **wxALIGN_CENTER**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering a progress indicator.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendProgressColumn (**const wxBitmap &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = 80, **wxAlignment** *align* = **wxALIGN_CENTER**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering a progress indicator.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendTextColumn (**const wxString &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering text.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendTextColumn (**const wxBitmap &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering text.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::AppendToggleColumn (**const wxString &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = 30, **wxAlignment** *align* = **wxALIGN_CENTER**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering a toggle.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* **wxDataViewCtrl::AppendToggleColumn** (**const wxBitmap &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = `30`, **wxAlignment** *align* = **wxALIGN_CENTER**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a column for rendering a toggle.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

virtual bool wxDataViewCtrl::AssociateModel (**wxDataViewModel** * *model*) `[virtual]`

Associates a wxDataViewModel with the control.

This increases the reference count of the model by 1.

virtual bool wxDataViewCtrl::ClearColumns ( ) `[virtual]`

Removes all columns.

virtual void wxDataViewCtrl::Collapse (**const wxDataViewItem &** *item*) `[virtual]`

Collapses the item.

bool wxDataViewCtrl::Create (**wxWindow** * *parent*, **wxWindowID** *id*, **const wxPoint &** *pos* = **wxDefaultPosition**, **const wxSize &** *size* = **wxDefaultSize**, **long** *style* = `0`, **const wxValidator &** *validator* = **wxDefaultValidator**, **const wxString &** *name* = `wxDataViewCtrlNameStr`)

Create the control.

Useful for two step creation.

virtual bool wxDataViewCtrl::DeleteColumn (**wxDataViewColumn** * *column*) `[virtual]`

Deletes given column.

virtual void wxDataViewCtrl::EditItem (**const wxDataViewItem &** *item*, **const wxDataViewColumn** * *column*) `[virtual]`

Programmatically starts editing given cell of *item*.

Doesn't do anything if the item or this column is not editable.

Note

Currently not implemented in wxOSX/Carbon.

Since

2.9.4

virtual bool wxDataViewCtrl::EnableDragSource (const wxDataFormat & *format*) [virtual]

Enable drag operations using the given *format*.

virtual bool wxDataViewCtrl::EnableDropTarget (const wxDataFormat & *format*) [virtual]

Enable drop operations using the given *format*.

virtual void wxDataViewCtrl::EnsureVisible (const wxDataViewItem & *item*, const wxDataViewColumn * *column* = NULL) [virtual]

Call this to ensure that the given item is visible.

virtual void wxDataViewCtrl::Expand (const wxDataViewItem & *item*) [virtual]

Expands the item.

virtual void wxDataViewCtrl::ExpandAncestors (const wxDataViewItem & *item*) [virtual]

Expands all ancestors of the *item*.

This method also ensures that the item itself as well as all ancestor items have been read from the model by the control.

virtual wxDataViewColumn* wxDataViewCtrl::GetColumn (unsigned int *pos*) const [virtual]

Returns pointer to the column.

pos refers to the position in the control which may change after reordering columns by the user.

virtual unsigned int wxDataViewCtrl::GetColumnCount ( ) const [virtual]

Returns the number of columns.

virtual int wxDataViewCtrl::GetColumnPosition (const wxDataViewColumn * *column*) const [virtual]

Returns the position of the column or -1 if not found in the control.

wxDataViewColumn* wxDataViewCtrl::GetCurrentColumn ( ) const

Returns the column that currently has focus.

If the focus is set to individual cell within the currently focused item (as opposed to being on the item as a whole), then this is the column that the focus is on.

Returns NULL if no column currently has focus.

See also

GetCurrentItem()

Since

2.9.4

wxDataViewItem wxDataViewCtrl::GetCurrentItem ( ) const

Returns the currently focused item.

This is the item that the keyboard commands apply to. It may be invalid if there is no focus currently.

This method is mostly useful for the controls with `wxDV_MULTIPLE` style as in the case of single selection it returns the same thing as GetSelection().

Notice that under all platforms except Mac OS X the currently focused item may be selected or not but under OS X the current item is always selected.

See also

SetCurrentItem(), GetCurrentColumn()

Since

2.9.2

wxDataViewColumn* wxDataViewCtrl::GetExpanderColumn ( ) const

Returns column containing the expanders.

int wxDataViewCtrl::GetIndent ( ) const

Returns indentation.

virtual wxRect wxDataViewCtrl::GetItemRect (const wxDataViewItem & *item*, const wxDataViewColumn * *col* = `NULL`) const `[virtual]`

Returns item rectangle.

This method is currently not implemented at all in wxGTK and only implemented for non-NULL *col* argument in wxOSX. It is fully implemented in the generic version of the control.

Parameters

item	A valid item.
col	If non-NULL, the rectangle returned corresponds to the intersection of the item with the specified column. If NULL, the rectangle spans all the columns.

wxDataViewModel* wxDataViewCtrl::GetModel ( )

Returns pointer to the data model associated with the control (if any).

virtual int wxDataViewCtrl::GetSelectedItemsCount ( ) const `[virtual]`

Returns the number of currently selected items.

This method may be called for both the controls with single and multiple selections and returns the number of selected item, possibly 0, in any case.

Since

2.9.3

virtual wxDataViewItem wxDataViewCtrl::GetSelection ( ) const `[virtual]`

Returns first selected item or an invalid item if none is selected.

This method may be called for both the controls with single and multiple selections but returns an invalid item if more than one item is selected in the latter case, use HasSelection() to determine if there are any selected items when using multiple selection.

virtual int wxDataViewCtrl::GetSelections (wxDataViewItemArray & *sel*) const `[virtual]`

Fills *sel* with currently selected items and returns their number.

This method may be called for both the controls with single and multiple selections. In the single selection case it returns the array with at most one element in it.

See also

GetSelectedItemsCount()

virtual wxDataViewColumn* wxDataViewCtrl::GetSortingColumn ( ) const `[virtual]`

Returns the wxDataViewColumn currently responsible for sorting or NULL if none has been selected.

virtual wxVector<wxDataViewColumn *> wxDataViewCtrl::GetSortingColumns ( ) const `[virtual]`

Returns the columns which should be used for sorting the data in this control.

This method is only useful when sorting by multiple columns had been enabled using AllowMultiColumnSort() previously, otherwise GetSortingColumn() is more convenient.

Returns

A possibly empty vector containing all the columns used selected by the user for sorting. The sort order can be retrieved from each column object separately.

Since

3.1.0

bool wxDataViewCtrl::HasSelection ( ) const

Returns true if any items are currently selected.

This method may be called for both the controls with single and multiple selections.

Calling this method is equivalent to calling GetSelectedItemsCount() and comparing its result with 0 but is more clear and might also be implemented more efficiently in the future.

Since

2.9.3

virtual void wxDataViewCtrl::HitTest (const wxPoint & *point,* wxDataViewItem & *item,* wxDataViewColumn *& *col*) const `[virtual]`

Hittest.

virtual bool wxDataViewCtrl::InsertColumn (unsigned int *pos*, **wxDataViewColumn** * *col*) [virtual]

Inserts a wxDataViewColumn to the control.

Returns true on success.

Reimplemented in wxDataViewListCtrl.

virtual bool wxDataViewCtrl::IsExpanded (const **wxDataViewItem** & *item*) const [virtual]

Return true if the item is expanded.

bool wxDataViewCtrl::IsMultiColumnSortAllowed ( ) const

Return true if using more than one column for sorting is allowed.

See AllowMultiColumnSort() and GetSortingColumns().

Since

3.1.0

virtual bool wxDataViewCtrl::IsSelected (const **wxDataViewItem** & *item*) const [virtual]

Return true if the item is selected.

wxDataViewColumn* wxDataViewCtrl::PrependBitmapColumn (const **wxString** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = −1, **wxAlignment** *align* = **wxALIGN_CENTER**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a bitmap.

Returns the wxDataViewColumn created in the function or NULL on failure.

wxDataViewColumn* wxDataViewCtrl::PrependBitmapColumn (const **wxBitmap** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = −1, **wxAlignment** *align* = **wxALIGN_CENTER**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a bitmap.

Returns the wxDataViewColumn created in the function or NULL on failure.

virtual bool wxDataViewCtrl::PrependColumn (**wxDataViewColumn** * *col*) [virtual]

Prepends a wxDataViewColumn to the control.

Returns true on success.

Note that there is a number of short cut methods which implicitly create a wxDataViewColumn and a wxDataView↩ Renderer for it.

Reimplemented in wxDataViewListCtrl.

wxDataViewColumn* wxDataViewCtrl::PrependDateColumn (const **wxString** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_ACTIVATABLE**, int *width* = −1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a date.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* **wxDataViewCtrl::PrependDateColumn** (**const wxBitmap &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_ACTIVATABLE**, **int** *width* = `-1`, **wxAlignment** *align* = **wxALIGN_NOT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a date.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* **wxDataViewCtrl::PrependIconTextColumn** (**const wxString &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = `-1`, **wxAlignment** *align* = **wxALIGN_NOT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering text with an icon.

Returns the wxDataViewColumn created in the function or NULL on failure. This method uses the wxDataViewIconTextRenderer class.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* **wxDataViewCtrl::PrependIconTextColumn** (**const wxBitmap &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = `-1`, **wxAlignment** *align* = **wxALIGN_NOT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering text with an icon.

Returns the wxDataViewColumn created in the function or NULL on failure. This method uses the wxDataViewIconTextRenderer class.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* **wxDataViewCtrl::PrependProgressColumn** (**const wxString &** *label*, **unsigned int** *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = `80`, **wxAlignment** *align* = **wxALIGN_CENTER**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a progress indicator.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::PrependProgressColumn (const **wxBitmap** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = 80, **wxAlignment** *align* = **wxALIGN_CENTER**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a progress indicator.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::PrependTextColumn (const **wxString** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering text.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::PrependTextColumn (const **wxBitmap** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = -1, **wxAlignment** *align* = **wxALIGN_NOT**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering text.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::PrependToggleColumn (const **wxString** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = 30, **wxAlignment** *align* = **wxALIGN_CENTER**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a toggle.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

wxDataViewColumn* wxDataViewCtrl::PrependToggleColumn (const **wxBitmap** & *label*, unsigned int *model_column*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, int *width* = 30, **wxAlignment** *align* = **wxALIGN_CENTER**, int *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Prepends a column for rendering a toggle.

Returns the wxDataViewColumn created in the function or NULL on failure.

Note

The *align* parameter is applied to both the column header and the column renderer.

virtual void wxDataViewCtrl::Select (const wxDataViewItem & *item*) [virtual]

Select the given item.

In single selection mode this changes the (unique) currently selected item. In multi selection mode, the *item* is selected and the previously selected items remain selected.

virtual void wxDataViewCtrl::SelectAll ( ) [virtual]

Select all items.

void wxDataViewCtrl::SetCurrentItem (const wxDataViewItem & *item*)

Changes the currently focused item.

The *item* parameter must be valid, there is no way to remove the current item from the control.

In single selection mode, calling this method is the same as calling Select() and is thus not very useful. In multiple selection mode this method only moves the current item however without changing the selection except under OS X where the current item is always selected, so calling SetCurrentItem() selects *item* if it hadn't been selected before.

See also

GetCurrentItem()

Since

2.9.2

void wxDataViewCtrl::SetExpanderColumn (wxDataViewColumn * *col*)

Set which column shall contain the tree-like expanders.

void wxDataViewCtrl::SetIndent (int *indent*)

Sets the indentation.

virtual bool wxDataViewCtrl::SetRowHeight (int *rowHeight*) [virtual]

Sets the row height.

This function can only be used when all rows have the same height, i.e. when wxDV_VARIABLE_LINE_HEIGHT flag is not used.

Currently this is implemented in the generic and native GTK versions only and nothing is done (and false returned) when using OS X port.

Also notice that this method can only be used to increase the row height compared with the default one (as determined by the return value of wxDataViewRenderer::GetSize()), if it is set to a too small value then the minimum required by the renderers will be used.

Returns

true if the line height was changed or false otherwise.

Since

2.9.2

virtual void wxDataViewCtrl::SetSelections (const wxDataViewItemArray & *sel*) `[virtual]`

Sets the selection to the array of wxDataViewItems.

virtual void wxDataViewCtrl::ToggleSortByColumn (int *column*) `[virtual]`

Toggle sorting by the given column.

This method should only be used when sorting by multiple columns is allowed, see AllowMultiColumnSort(), and does nothing otherwise.

Since

3.1.0

virtual void wxDataViewCtrl::Unselect (const wxDataViewItem & *item*) `[virtual]`

Unselect the given item.

virtual void wxDataViewCtrl::UnselectAll ( ) `[virtual]`

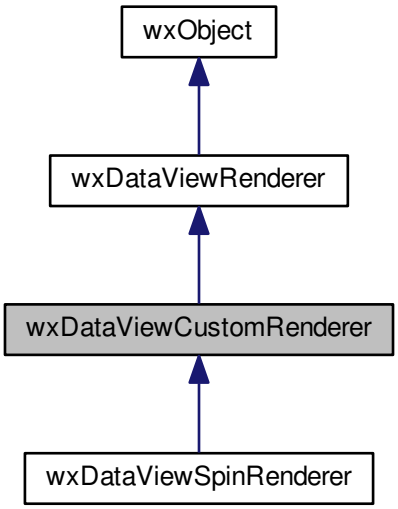
Unselect all item.

This method only has effect if multiple selections are allowed.

21.146 wxDataViewCustomRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewCustomRenderer:



21.146.1 Detailed Description

You need to derive a new class from wxDataViewCustomRenderer in order to write a new renderer.

You need to override at least wxDataViewRenderer::SetValue, wxDataViewRenderer::GetValue, wxDataViewCustomRenderer::GetSize and wxDataViewCustomRenderer::Render.

If you want your renderer to support in-place editing then you also need to override wxDataViewCustomRenderer::HasEditorCtrl, wxDataViewCustomRenderer::CreateEditorCtrl and wxDataViewCustomRenderer::GetValueFromEditorCtrl.

Note that a special event handler will be pushed onto that editor control which handles <*ENTER*> and focus out events in order to end the editing.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewCustomRenderer (const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  Constructor.
- virtual ~wxDataViewCustomRenderer ()

  Destructor.
- virtual bool ActivateCell (const wxRect &cell, wxDataViewModel *model, const wxDataViewItem &item, unsigned int col, const wxMouseEvent *mouseEvent)

  Override this to react to cell activation.
- virtual wxWindow * CreateEditorCtrl (wxWindow *parent, wxRect labelRect, const wxVariant &value)

  Override this to create the actual editor control once editing is about to start.
- const wxDataViewItemAttr & GetAttr () const

  Return the attribute to be used for rendering.
- virtual wxSize GetSize () const =0

  Return size required to show content.
- virtual bool GetValueFromEditorCtrl (wxWindow *editor, wxVariant &value)

  Override this so that the renderer can get the value from the editor control (pointed to by editor):
- virtual bool HasEditorCtrl () const

  Override this and make it return true in order to indicate that this renderer supports in-place editing.
- virtual bool LeftClick (wxPoint cursor, wxRect cell, wxDataViewModel *model, const wxDataViewItem &item, unsigned int col)

  Override this to react to a left click.
- virtual bool Activate (wxRect cell, wxDataViewModel *model, const wxDataViewItem &item, unsigned int col)

  Override this to react to the activation of a cell.
- virtual bool Render (wxRect cell, wxDC *dc, int state)=0

  Override this to render the cell.
- void RenderText (const wxString &text, int xoffset, wxRect cell, wxDC *dc, int state)

  This method should be called from within Render() whenever you need to render simple text.
- virtual bool StartDrag (const wxPoint &cursor, const wxRect &cell, wxDataViewModel *model, const wxDataViewItem &item, unsigned int col)

  Override this to start a drag operation.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Protected Member Functions

- wxSize GetTextExtent (const wxString &str) const

  Helper for GetSize() implementations, respects attributes.

Additional Inherited Members

21.146.2 Constructor & Destructor Documentation

wxDataViewCustomRenderer::wxDataViewCustomRenderer (const wxString & *varianttype* = GetDefaultType () , wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

Constructor.

virtual wxDataViewCustomRenderer::~wxDataViewCustomRenderer ( ) [virtual]

Destructor.

21.146.3 Member Function Documentation

virtual bool wxDataViewCustomRenderer::Activate (wxRect *cell*, wxDataViewModel * *model*, const wxDataViewItem & *item*, unsigned int *col*) [virtual]

Override this to react to the activation of a cell.

Deprecated Use ActivateCell instead.

virtual bool wxDataViewCustomRenderer::ActivateCell (const wxRect & *cell*, wxDataViewModel * *model*, const wxDataViewItem & *item*, unsigned int *col*, const wxMouseEvent * *mouseEvent*) [virtual]

Override this to react to cell *activation*.

Activating a cell is an alternative to showing inline editor when the value can be edited in a simple way that doesn't warrant full editor control. The most typical use of cell activation is toggling the checkbox in wxDataViewToggleRenderer; others would be e.g. an embedded volume slider or a five-star rating column.

The exact means of activating a cell are platform-dependent, but they are usually similar to those used for inline editing of values. Typically, a cell would be activated by Space or Enter keys or by left mouse click.

This method will only be called if the cell has the wxDATAVIEW_CELL_ACTIVATABLE mode.

Parameters

cell	Coordinates of the activated cell's area.
model	The model to manipulate in response.
item	Activated item.
col	Activated column of *item*.
mouseEvent	If the activation was triggered by mouse click, contains the corresponding event. Is NULL otherwise (for keyboard activation). Mouse coordinates are adjusted to be relative to the cell.

Since

2.9.3

Note

Do not confuse this method with item activation in wxDataViewCtrl and the wxEVT_DATAVIEW_ITEM_ACTIVATED event. That one is used for activating the item (or, to put it differently, the entire row) similarly to analogous messages in wxTreeCtrl and wxListCtrl, and the effect differs (play a song, open a file etc.). Cell activation, on the other hand, is all about interacting with the individual cell.

See also

CreateEditorCtrl()

virtual wxWindow* wxDataViewCustomRenderer::CreateEditorCtrl (wxWindow * *parent*, wxRect *labelRect*, const wxVariant & *value*) [virtual]

Override this to create the actual editor control once editing is about to start.

This method will only be called if the cell has the wxDATAVIEW_CELL_EDITABLE mode. Editing is typically triggered by slowly double-clicking the cell or by a platform-dependent keyboard shortcut (F2 is typical on Windows, Space and/or Enter is common elsewhere and supported on Windows too).

Parameters

parent	The parent of the editor control.
labelRect	Indicates the position and size of the editor control. The control should be created in place of the cell and *labelRect* should be respected as much as possible.
value	Initial value of the editor.

An example:

```
{
    long l = value;
    return new wxSpinCtrl( parent, wxID_ANY, wxEmptyString,
                labelRect.GetTopLeft(), labelRect.GetSize(), 0, 0, 100, l );
}
```

See also

ActivateCell()

Reimplemented from wxDataViewRenderer.

const wxDataViewItemAttr& wxDataViewCustomRenderer::GetAttr ( ) const

Return the attribute to be used for rendering.

This function may be called from Render() implementation to use the attributes defined for the item if the renderer supports them.

Notice that when Render() is called, the wxDC object passed to it is already set up to use the correct attributes (e.g. its font is set to bold or italic version if wxDataViewItemAttr::GetBold() or GetItalic() returns true) so it may not be necessary to call it explicitly if you only want to render text using the items attributes.

Since

2.9.1

static wxString wxDataViewCustomRenderer::GetDefaultType ( ) [static]

Returns the wxVariant type used with this renderer.

Since

3.1.0

virtual wxSize wxDataViewCustomRenderer::GetSize ( ) const [pure virtual]

Return size required to show content.

wxSize wxDataViewCustomRenderer::GetTextExtent (const wxString & *str*) const [protected]

Helper for GetSize() implementations, respects attributes.

virtual bool wxDataViewCustomRenderer::GetValueFromEditorCtrl (wxWindow * *editor*, wxVariant & *value*) [virtual]

Override this so that the renderer can get the value from the editor control (pointed to by *editor*):

```
{
    wxSpinCtrl *sc = (wxSpinCtrl*) editor;
    long l = sc->GetValue();
    value = l;
    return true;
}
```

Reimplemented from wxDataViewRenderer.

virtual bool wxDataViewCustomRenderer::HasEditorCtrl ( ) const [virtual]

Override this and make it return true in order to indicate that this renderer supports in-place editing.

Reimplemented from wxDataViewRenderer.

virtual bool wxDataViewCustomRenderer::LeftClick (wxPoint *cursor*, wxRect *cell*, wxDataViewModel * *model*, const wxDataViewItem & *item*, unsigned int *col*) [virtual]

Override this to react to a left click.

This method will only be called in `wxDATAVIEW_CELL_ACTIVATABLE` mode.

Deprecated Use ActivateCell instead.

virtual bool wxDataViewCustomRenderer::Render (wxRect *cell*, wxDC * *dc*, int *state*) [pure virtual]

Override this to render the cell.

Before this is called, wxDataViewRenderer::SetValue was called so that this instance knows what to render.

void wxDataViewCustomRenderer::RenderText (const wxString & *text*, int *xoffset*, wxRect *cell*, wxDC * *dc*, int *state*)

This method should be called from within Render() whenever you need to render simple text.

This will ensure that the correct colour, font and vertical alignment will be chosen so the text will look the same as text drawn by native renderers.

virtual bool wxDataViewCustomRenderer::StartDrag (const wxPoint & *cursor,* const wxRect & *cell,* wxDataViewModel * *model,* const wxDataViewItem & *item,* unsigned int *col*) [virtual]

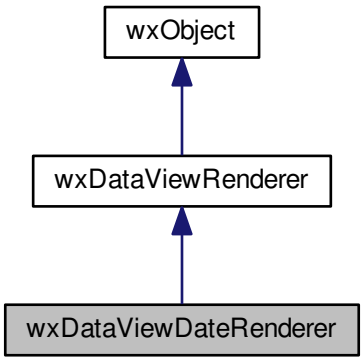
Override this to start a drag operation.

Not yet supported.

21.147 wxDataViewDateRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewDateRenderer:



21.147.1 Detailed Description

This class is used by wxDataViewCtrl to render calendar controls.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewDateRenderer (const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wx↩DATAVIEW_CELL_ACTIVATABLE, int align=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Additional Inherited Members

21.147.2 Constructor & Destructor Documentation

wxDataViewDateRenderer::wxDataViewDateRenderer (const wxString & *varianttype* = GetDefaultType (), wxDataViewCellMode *mode* = wxDATAVIEW_CELL_ACTIVATABLE, int *align* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.147.3 Member Function Documentation

static wxString wxDataViewDateRenderer::GetDefaultType ( ) `[static]`

Returns the wxVariant type used with this renderer.

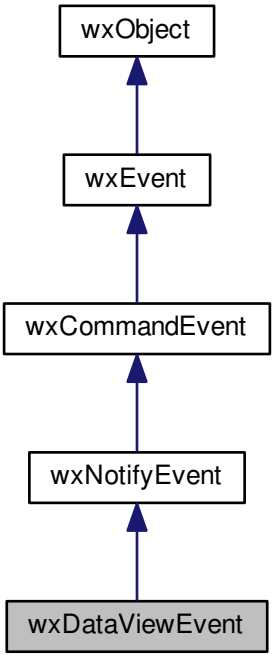
Since

3.1.0

21.148 wxDataViewEvent Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewEvent:



21.148.1 Detailed Description

This is the event class for the wxDataViewCtrl notifications.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxDataViewEvent& event)

Event macros:

- EVT_DATAVIEW_SELECTION_CHANGED(id, func): Process a `wxEVT_DATAVIEW_SELECTION_CHANGED` event.
- EVT_DATAVIEW_ITEM_ACTIVATED(id, func): Process a `wxEVT_DATAVIEW_ITEM_ACTIVATED` event.
- EVT_DATAVIEW_ITEM_EDITING_STARTED(id, func): Process a `wxEVT_DATAVIEW_ITEM_EDITING_STARTED` event.
- EVT_DATAVIEW_ITEM_EDITING_DONE(id, func): Process a `wxEVT_DATAVIEW_ITEM_EDITING_DONE` event.
- EVT_DATAVIEW_ITEM_COLLAPSING(id, func): Process a `wxEVT_DATAVIEW_ITEM_COLLAPSING` event.
- EVT_DATAVIEW_ITEM_COLLAPSED(id, func): Process a `wxEVT_DATAVIEW_ITEM_COLLAPSED` event.
- EVT_DATAVIEW_ITEM_EXPANDING(id, func): Process a `wxEVT_DATAVIEW_ITEM_EXPANDING` event.
- EVT_DATAVIEW_ITEM_EXPANDED(id, func): Process a `wxEVT_DATAVIEW_ITEM_EXPANDED` event.
- EVT_DATAVIEW_ITEM_VALUE_CHANGED(id, func): Process a `wxEVT_DATAVIEW_ITEM_VALUE_CHANGED` event.
- EVT_DATAVIEW_ITEM_CONTEXT_MENU(id, func): Process a `wxEVT_DATAVIEW_ITEM_CONTEXT_MENU` event.
- EVT_DATAVIEW_COLUMN_HEADER_CLICK(id, func): Process a `wxEVT_DATAVIEW_COLUMN_HEADER_CLICK` event.
- EVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK(id, func): Process a `wxEVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK` event.
- EVT_DATAVIEW_COLUMN_SORTED(id, func): Process a `wxEVT_DATAVIEW_COLUMN_SORTED` event.
- EVT_DATAVIEW_COLUMN_REORDERED(id, func): Process a `wxEVT_DATAVIEW_COLUMN_REORDERED` event. Currently this even is only generated when using the native OSX version.
- EVT_DATAVIEW_ITEM_BEGIN_DRAG(id, func): Process a `wxEVT_DATAVIEW_ITEM_BEGIN_DRAG` event.
- EVT_DATAVIEW_ITEM_DROP_POSSIBLE(id, func): Process a `wxEVT_DATAVIEW_ITEM_DROP_POSSIBLE` event.
- EVT_DATAVIEW_ITEM_DROP(id, func): Process a `wxEVT_DATAVIEW_ITEM_DROP` event.
- EVT_DATAVIEW_CACHE_HINT(id, func): Process a `wxEVT_DATAVIEW_CACHE_HINT` event.

Library: wxAdvanced

Category: Events, wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewEvent (wxEventType commandType=wxEVT_NULL, int winid=0)

  Constructor.
- int GetColumn () const

  Returns the position of the column in the control or -1 if no column field was set by the event emitter.
- wxDataViewColumn * GetDataViewColumn () const

  Returns a pointer to the wxDataViewColumn from which the event was emitted or NULL.
- wxDataViewModel * GetModel () const

  Returns the wxDataViewModel associated with the event.
- wxPoint GetPosition () const

  Returns the position of a context menu event in screen coordinates.
- const wxVariant & GetValue () const

  Returns a reference to a value.
- bool IsEditCancelled () const

  Can be used to determine whether the new value is going to be accepted in wxEVT_DATAVIEW_ITEM_EDITING↩ _DONE handler.
- void SetColumn (int col)

  Sets the column index associated with this event.
- void SetDataViewColumn (wxDataViewColumn *col)

  For `wxEVT_DATAVIEW_COLUMN_HEADER_CLICK` only.
- void SetModel (wxDataViewModel *model)

  Sets the dataview model associated with this event.
- void SetValue (const wxVariant &value)

  Sets the value associated with this event.
- void SetDataObject (wxDataObject *obj)

  Set wxDataObject for data transfer within a drag operation.
- wxDataFormat GetDataFormat () const

  Gets the wxDataFormat during a drop operation.
- size_t GetDataSize () const

  Gets the data size for a drop data transfer.
- void * GetDataBuffer () const

  Gets the data buffer for a drop data transfer.
- void SetDragFlags (int flags)

  Specify the kind of the drag operation to perform.
- wxDragResult GetDropEffect () const

  Returns the effect the user requested to happen to the dropped data.
- int GetCacheFrom () const

  Return the first row that will be displayed.
- int GetCacheTo () const

  Return the last row that will be displayed.
- wxDataViewItem GetItem () const

  Returns the item affected by the event.
- void SetItem (const wxDataViewItem &item)
- void SetEditCanceled (bool editCancelled)
- void SetPosition (int x, int y)

- void SetCache (int from, int to)
- wxDataObject * GetDataObject () const
- void SetDataFormat (const wxDataFormat &format)
- void SetDataSize (size_t size)
- void SetDataBuffer (void *buf)
- int GetDragFlags () const
- void SetDropEffect (wxDragResult effect)

Additional Inherited Members

21.148.2 Constructor & Destructor Documentation

wxDataViewEvent::wxDataViewEvent (wxEventType *commandType* = wxEVT_NULL, int *winid* = 0)

Constructor.

Typically used by wxWidgets internals only.

21.148.3 Member Function Documentation

int wxDataViewEvent::GetCacheFrom ( ) const

Return the first row that will be displayed.

int wxDataViewEvent::GetCacheTo ( ) const

Return the last row that will be displayed.

int wxDataViewEvent::GetColumn ( ) const

Returns the position of the column in the control or -1 if no column field was set by the event emitter.

void* wxDataViewEvent::GetDataBuffer ( ) const

Gets the data buffer for a drop data transfer.

wxDataFormat wxDataViewEvent::GetDataFormat ( ) const

Gets the wxDataFormat during a drop operation.

wxDataObject* wxDataViewEvent::GetDataObject ( ) const

size_t wxDataViewEvent::GetDataSize ( ) const

Gets the data size for a drop data transfer.

wxDataViewColumn* wxDataViewEvent::GetDataViewColumn ( ) const

Returns a pointer to the wxDataViewColumn from which the event was emitted or NULL.

int wxDataViewEvent::GetDragFlags ( ) const

wxDragResult wxDataViewEvent::GetDropEffect ( ) const

Returns the effect the user requested to happen to the dropped data.

This function can be used inside wxEVT_DATAVIEW_ITEM_DROP_POSSIBLE and wxEVT_DATAVIEW_ITEM_DROP handlers and returns whether the user is trying to copy (the return value is wxDragCopy) or move (if the return value is wxDragMove) the data.

Currently this is only available when using the generic version of wxDataViewCtrl (used e.g. under MSW) and always returns wxDragNone in the GTK and OS X native versions.

Since

2.9.4

wxDataViewItem wxDataViewEvent::GetItem ( ) const

Returns the item affected by the event.

Notice that for `wxEVT_DATAVIEW_ITEM_DROP_POSSIBLE` and `wxEVT_DATAVIEW_ITEM_DROP` event handlers, the item may be invalid, indicating that the drop is about to happen outside of the item area.

wxDataViewModel* wxDataViewEvent::GetModel ( ) const

Returns the wxDataViewModel associated with the event.

wxPoint wxDataViewEvent::GetPosition ( ) const

Returns the position of a context menu event in screen coordinates.

const wxVariant& wxDataViewEvent::GetValue ( ) const

Returns a reference to a value.

bool wxDataViewEvent::IsEditCancelled ( ) const

Can be used to determine whether the new value is going to be accepted in wxEVT_DATAVIEW_ITEM_EDITING_DONE handler.

Returns true if editing the item was cancelled or if the user tried to enter an invalid value (refused by wxDataViewRenderer::Validate()). If this method returns false, it means that the value in the model is about to be changed to the new one.

Notice that wxEVT_DATAVIEW_ITEM_EDITING_DONE event handler can call wxNotifyEvent::Veto() to prevent this from happening.

Currently support for setting this field and for vetoing the change is only available in the generic version of wxDataViewCtrl, i.e. under MSW but not GTK nor OS X.

Since

2.9.3

void wxDataViewEvent::SetCache (int *from,* int *to*)

void wxDataViewEvent::SetColumn (int *col*)

Sets the column index associated with this event.

void wxDataViewEvent::SetDataBuffer (void * *buf*)

void wxDataViewEvent::SetDataFormat (const wxDataFormat & *format*)

void wxDataViewEvent::SetDataObject (wxDataObject * *obj*)

Set wxDataObject for data transfer within a drag operation.

void wxDataViewEvent::SetDataSize (size_t *size*)

void wxDataViewEvent::SetDataViewColumn (wxDataViewColumn * *col*)

For `wxEVT_DATAVIEW_COLUMN_HEADER_CLICK` only.

void wxDataViewEvent::SetDragFlags (int *flags*)

Specify the kind of the drag operation to perform.

This method can be used inside a wxEVT_DATAVIEW_ITEM_BEGIN_DRAG handler in order to configure the drag operation. Valid values are wxDrag_CopyOnly (default), wxDrag_AllowMove (allow the data to be moved) and wxDrag_DefaultMove.

Currently it is only honoured by the generic version of wxDataViewCtrl (used e.g. under MSW) and not supported by the native GTK and OS X versions.

See also

GetDropEffect()

Since

2.9.4

void wxDataViewEvent::SetDropEffect (wxDragResult *effect*)

void wxDataViewEvent::SetEditCanceled (bool *editCancelled*)

void wxDataViewEvent::SetItem (const wxDataViewItem & *item*)

void wxDataViewEvent::SetModel (wxDataViewModel * *model*)

Sets the dataview model associated with this event.

void wxDataViewEvent::SetPosition (int *x,* int *y*)

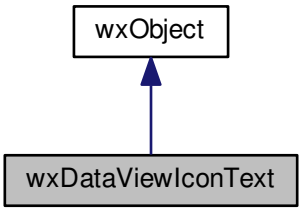
void wxDataViewEvent::SetValue (const wxVariant & *value*)

Sets the value associated with this event.

21.149 wxDataViewIconText Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewIconText:



21.149.1 Detailed Description

wxDataViewIconText is used by wxDataViewIconTextRenderer for data transfer.

This class can be converted to and from a wxVariant.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- const wxIcon & GetIcon () const

  Gets the icon.
- wxString GetText () const

  Gets the text.
- void SetIcon (const wxIcon &icon)

  Set the icon.
- void SetText (const wxString &text)

  Set the text.

- wxDataViewIconText (const wxString &text=wxEmptyString, const wxIcon &icon=wxNullIcon)

  Constructor.
- wxDataViewIconText (const wxDataViewIconText &other)

  Constructor.

Additional Inherited Members

21.149.2 Constructor & Destructor Documentation

wxDataViewIconText::wxDataViewIconText (const wxString & *text* = wxEmptyString, const wxIcon & *icon* = wxNullIcon)

Constructor.

wxDataViewIconText::wxDataViewIconText (const wxDataViewIconText & *other*)

Constructor.

21.149.3 Member Function Documentation

const wxIcon& wxDataViewIconText::GetIcon ( ) const

Gets the icon.

wxString wxDataViewIconText::GetText ( ) const

Gets the text.

void wxDataViewIconText::SetIcon (const wxIcon & *icon*)

Set the icon.

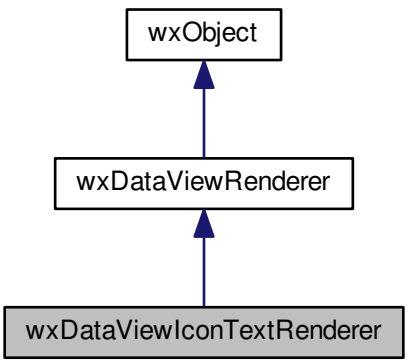
void wxDataViewIconText::SetText (const wxString & *text*)

Set the text.

21.150 wxDataViewIconTextRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewIconTextRenderer:



21.150.1 Detailed Description

The wxDataViewIconTextRenderer class is used to display text with a small icon next to it as it is typically done in a file manager.

This classes uses the wxDataViewIconText helper class to store its data. wxDataViewIconText can be converted to and from a wxVariant using the left shift operator.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewIconTextRenderer (const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Additional Inherited Members

21.150.2 Constructor & Destructor Documentation

wxDataViewIconTextRenderer::wxDataViewIconTextRenderer (const wxString & *varianttype* = GetDefaultType (), wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.150.3 Member Function Documentation

static wxString wxDataViewIconTextRenderer::GetDefaultType ( ) `[static]`

Returns the wxVariant type used with this renderer.

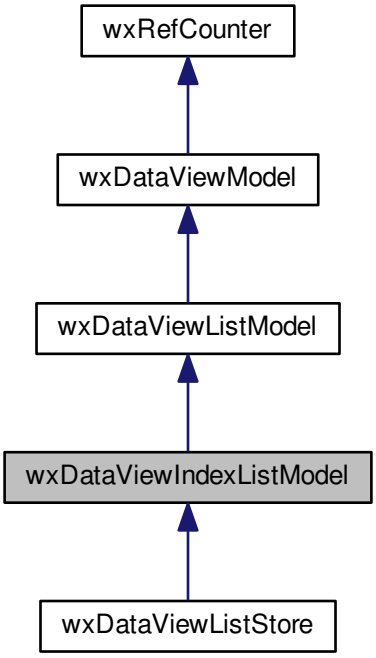
Since

3.1.0

21.151 wxDataViewIndexListModel Class Reference

```
#include <wx/dataview.h>
```

Inheritance diagram for wxDataViewIndexListModel:



21.151.1 Detailed Description

wxDataViewIndexListModel is a specialized data model which lets you address an item by its position (row) rather than its wxDataViewItem (which you can obtain from this class).

This model also provides its own wxDataViewIndexListModel::Compare method which sorts the model's data by the index.

This model is not a virtual model since the control stores each wxDataViewItem. Use wxDataViewVirtualListModel if you need to display millions of items or have other reason to use a virtual control.

See also

wxDataViewListModel for the API.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewIndexListModel (unsigned int initial_size=0)

  Constructor.
- wxDataViewItem GetItem (unsigned int row) const

Returns the wxDataViewItem at the given row.

- void Reset (unsigned int new_size)

  Call this after if the data has to be read again from the model.
- void RowAppended ()

  Call this after a row has been appended to the model.
- void RowChanged (unsigned int row)

  Call this after a row has been changed.
- void RowDeleted (unsigned int row)

  Call this after a row has been deleted.
- void RowInserted (unsigned int before)

  Call this after a row has been inserted at the given position.
- void RowPrepended ()

  Call this after a row has been prepended to the model.
- void RowValueChanged (unsigned int row, unsigned int col)

  Call this after a value has been changed.
- void RowsDeleted (const wxArrayInt &rows)

  Call this after rows have been deleted.

Additional Inherited Members

21.151.2 Constructor & Destructor Documentation

wxDataViewIndexListModel::wxDataViewIndexListModel (unsigned int *initial_size* = 0)

Constructor.

21.151.3 Member Function Documentation

wxDataViewItem wxDataViewIndexListModel::GetItem (unsigned int *row*) const

Returns the wxDataViewItem at the given *row*.

void wxDataViewIndexListModel::Reset (unsigned int *new_size*)

Call this after if the data has to be read again from the model.

This is useful after major changes when calling the methods below (possibly thousands of times) doesn't make sense.

void wxDataViewIndexListModel::RowAppended ( )

Call this after a row has been appended to the model.

void wxDataViewIndexListModel::RowChanged (unsigned int *row*)

Call this after a row has been changed.

void wxDataViewIndexListModel::RowDeleted (unsigned int *row*)

Call this after a row has been deleted.

void wxDataViewIndexListModel::RowInserted (unsigned int *before*)

Call this after a row has been inserted at the given position.

void wxDataViewIndexListModel::RowPrepended ( )

Call this after a row has been prepended to the model.

void wxDataViewIndexListModel::RowsDeleted (const wxArrayInt & *rows*)

Call this after rows have been deleted.

The array will internally get copied and sorted in descending order so that the rows with the highest position will be deleted first.

void wxDataViewIndexListModel::RowValueChanged (unsigned int *row,* unsigned int *col*)

Call this after a value has been changed.

21.152 wxDataViewItem Class Reference

`#include <wx/dataview.h>`

21.152.1 Detailed Description

wxDataViewItem is a small opaque class that represents an item in a wxDataViewCtrl in a persistent way, i.e.

independent of the position of the item in the control or changes to its contents.

It must hold a unique ID of type *void*∗ in its only field and can be converted to and from it.

If the ID is NULL the wxDataViewItem is invalid and wxDataViewItem::IsOk will return false which used in many places in the API of wxDataViewCtrl to indicate that e.g. no item was found. An ID of NULL is also used to indicate the invisible root. Examples for this are wxDataViewModel::GetParent and wxDataViewModel::GetChildren.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- void ∗ GetID () const
  Returns the ID.
- bool IsOk () const
  Returns true if the ID is not NULL.

- wxDataViewItem ()
  Constructor.
- wxDataViewItem (const wxDataViewItem &item)
  Constructor.
- wxDataViewItem (void ∗id)
  Constructor.

21.152.2 Constructor & Destructor Documentation

wxDataViewItem::wxDataViewItem ( )

Constructor.

wxDataViewItem::wxDataViewItem (const wxDataViewItem & *item*)

Constructor.

wxDataViewItem::wxDataViewItem (void * *id*) `[explicit]`

Constructor.

21.152.3 Member Function Documentation

void* wxDataViewItem::GetID ( ) const

Returns the ID.

bool wxDataViewItem::IsOk ( ) const

Returns true if the ID is not NULL.

21.153 wxDataViewItemAttr Class Reference

`#include <wx/dataview.h>`

21.153.1 Detailed Description

This class is used to indicate to a wxDataViewCtrl that a certain item (see wxDataViewItem) has extra font attributes for its renderer.

For this, it is required to override wxDataViewModel::GetAttr.

Attributes are currently only supported by wxDataViewTextRendererText.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewItemAttr ()

  Constructor.
- void SetBold (bool set)

  Call this to indicate that the item shall be displayed in bold text.
- void SetColour (const wxColour &colour)

  Call this to indicate that the item shall be displayed with that colour.

- void SetBackgroundColour (const wxColour &colour)

  Call this to set the background colour to use.
- void SetItalic (bool set)

  Call this to indicate that the item shall be displayed in italic text.
- bool HasColour () const

  Returns true if the colour property has been set.
- const wxColour & GetColour () const

  Returns this attribute's colour.
- bool HasFont () const

  Returns true if any property affecting the font has been set.
- bool GetBold () const

  Returns value of the bold property.
- bool GetItalic () const

  Returns value of the italics property.
- bool HasBackgroundColour () const

  Returns true if the background colour property has been set.
- const wxColour & GetBackgroundColour () const

  Returns the colour to be used for the background.
- bool IsDefault () const

  Returns true if none of the properties have been set.
- wxFont GetEffectiveFont (const wxFont &font) const

  Return the font based on the given one with this attribute applied to it.

21.153.2 Constructor & Destructor Documentation

wxDataViewItemAttr::wxDataViewItemAttr ( )

Constructor.

21.153.3 Member Function Documentation

const wxColour& wxDataViewItemAttr::GetBackgroundColour ( ) const

Returns the colour to be used for the background.

bool wxDataViewItemAttr::GetBold ( ) const

Returns value of the bold property.

const wxColour& wxDataViewItemAttr::GetColour ( ) const

Returns this attribute's colour.

wxFont wxDataViewItemAttr::GetEffectiveFont (const wxFont & *font*) const

Return the font based on the given one with this attribute applied to it.

bool wxDataViewItemAttr::GetItalic ( ) const

Returns value of the italics property.

bool wxDataViewItemAttr::HasBackgroundColour ( ) const

Returns true if the background colour property has been set.

bool wxDataViewItemAttr::HasColour ( ) const

Returns true if the colour property has been set.

bool wxDataViewItemAttr::HasFont ( ) const

Returns true if any property affecting the font has been set.

bool wxDataViewItemAttr::IsDefault ( ) const

Returns true if none of the properties have been set.

void wxDataViewItemAttr::SetBackgroundColour (const wxColour & *colour*)

Call this to set the background colour to use.

Currently this attribute is only supported in the generic version of wxDataViewCtrl and ignored by the native GTK+ and OS X implementations.

Since

2.9.4

void wxDataViewItemAttr::SetBold (bool *set*)

Call this to indicate that the item shall be displayed in bold text.

void wxDataViewItemAttr::SetColour (const wxColour & *colour*)

Call this to indicate that the item shall be displayed with that colour.

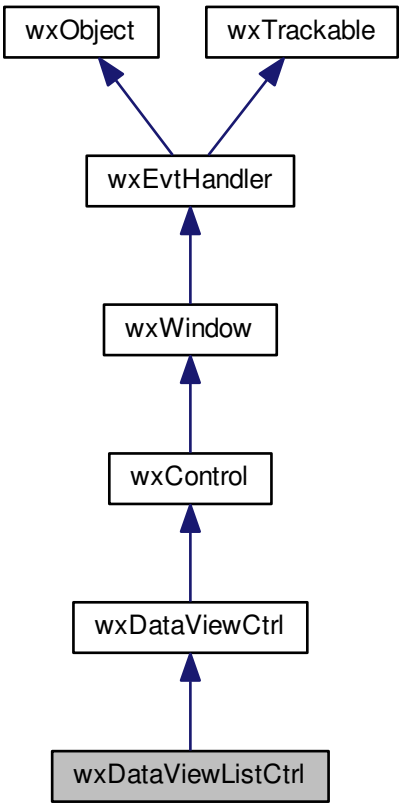
void wxDataViewItemAttr::SetItalic (bool *set*)

Call this to indicate that the item shall be displayed in italic text.

21.154 wxDataViewListCtrl Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewListCtrl:



21.154.1 Detailed Description

This class is a wxDataViewCtrl which internally uses a wxDataViewListStore and forwards most of its API to that class.

The purpose of this class is to offer a simple way to display and edit a small table of data without having to write your own wxDataViewModel.

```
wxDataViewListCtrl *listctrl = new wxDataViewListCtrl( parent,
      wxID_ANY );

listctrl->AppendToggleColumn( "Toggle" );
listctrl->AppendTextColumn( "Text" );

wxVector<wxVariant> data;
data.push_back( wxVariant(true) );
data.push_back( wxVariant("row 1") );
listctrl->AppendItem( data );

data.clear();
data.push_back( wxVariant(false) );
data.push_back( wxVariant("row 3") );
listctrl->AppendItem( data );
```

Styles

This class supports the following styles:

See wxDataViewCtrl for the list of supported styles.

Events emitted by this class

Event macros for events emitted by this class:

See wxDataViewCtrl for the list of events emitted by this class.

Library: wxAdvanced

Category: Controls, wxDataViewCtrl Related Classes

Since

2.9.0

Public Member Functions

- wxDataViewListCtrl ()

  Default ctor.
- wxDataViewListCtrl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDV_ROW_LINES, const wxValidator &validator=wxDefaultValidator)

  Constructor.
- ~wxDataViewListCtrl ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDV_ROW_LINES, const wxValidator &validator=wxDefaultValidator)

  Creates the control and a wxDataViewListStore as its internal model.
- int ItemToRow (const wxDataViewItem &item) const

  Returns the position of given item or wxNOT_FOUND if it's not a valid item.
- wxDataViewItem RowToItem (int row) const

  Returns the wxDataViewItem at the given row.

- wxDataViewListStore * GetStore ()

  Returns the store.
- const wxDataViewListStore * GetStore () const

  Returns the store.

Selection handling functions

- int GetSelectedRow () const

  Returns index of the selected row or wxNOT_FOUND.
- void SelectRow (unsigned row)

  Selects given row.
- void UnselectRow (unsigned row)

  Unselects given row.
- bool IsRowSelected (unsigned row) const

  Returns true if row is selected.

Column management functions

- virtual bool AppendColumn (wxDataViewColumn *column)

  Appends a column to the control and additionally appends a column to the store with the type string.

- void AppendColumn (wxDataViewColumn *column, const wxString &varianttype)

  Appends a column to the control and additionally appends a column to the list store with the type varianttype.
- wxDataViewColumn * AppendTextColumn (const wxString &label, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_LEFT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a text column to the control and the store.
- wxDataViewColumn * AppendToggleColumn (const wxString &label, wxDataViewCellMode mode=wxDATAVIEW_CELL_ACTIVATABLE, int width=-1, wxAlignment align=wxALIGN_LEFT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a toggle column to the control and the store.
- wxDataViewColumn * AppendProgressColumn (const wxString &label, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_LEFT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends a progress column to the control and the store.
- wxDataViewColumn * AppendIconTextColumn (const wxString &label, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int width=-1, wxAlignment align=wxALIGN_LEFT, int flags=wxDATAVIEW_COL_RESIZABLE)

  Appends an icon-and-text column to the control and the store.
- virtual bool InsertColumn (unsigned int pos, wxDataViewColumn *column)

  Inserts a column to the control and additionally inserts a column to the store with the type string.
- void InsertColumn (unsigned int pos, wxDataViewColumn *column, const wxString &varianttype)

  Inserts a column to the control and additionally inserts a column to the list store with the type varianttype.
- virtual bool PrependColumn (wxDataViewColumn *column)

  Prepends a column to the control and additionally prepends a column to the store with the type string.
- void PrependColumn (wxDataViewColumn *column, const wxString &varianttype)

  Prepends a column to the control and additionally prepends a column to the list store with the type varianttype.

Item management functions

- void AppendItem (const wxVector< wxVariant > &values, wxUIntPtr data=NULL)

  Appends an item (=row) to the control and store.
- void PrependItem (const wxVector< wxVariant > &values, wxUIntPtr data=NULL)

  Prepends an item (=row) to the control and store.
- void InsertItem (unsigned int row, const wxVector< wxVariant > &values, wxUIntPtr data=NULL)

  Inserts an item (=row) to the control and store.
- void DeleteItem (unsigned row)

  Delete the row at position row.
- void DeleteAllItems ()

  Delete all items (= all rows).
- unsigned int GetItemCount () const

  Returns the number of items (=rows) in the control.
- wxUIntPtr GetItemData (const wxDataViewItem &item) const

  Returns the client data associated with the item.
- void SetValue (const wxVariant &value, unsigned int row, unsigned int col)

  Sets the value in the store and update the control.
- void GetValue (wxVariant &value, unsigned int row, unsigned int col)

  Returns the value from the store.
- void SetTextValue (const wxString &value, unsigned int row, unsigned int col)

  Sets the value in the store and update the control.
- wxString GetTextValue (unsigned int row, unsigned int col) const

  Returns the value from the store.
- void SetToggleValue (bool value, unsigned int row, unsigned int col)

  Sets the value in the store and update the control.
- bool GetToggleValue (unsigned int row, unsigned int col) const

  Returns the value from the store.
- void SetItemData (const wxDataViewItem &item, wxUIntPtr data)

  Associates a client data pointer with the given item.

Additional Inherited Members

21.154.2 Constructor & Destructor Documentation

wxDataViewListCtrl::wxDataViewListCtrl ( )

Default ctor.

wxDataViewListCtrl::wxDataViewListCtrl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDV_ROW_LINES, const wxValidator & *validator* = wxDefaultValidator)

Constructor.

Calls Create().

wxDataViewListCtrl::~wxDataViewListCtrl ( )

Destructor.

Deletes the image list if any.

21.154.3 Member Function Documentation

virtual bool wxDataViewListCtrl::AppendColumn (wxDataViewColumn * *column*) [virtual]

Appends a column to the control and additionally appends a column to the store with the type string.

Reimplemented from wxDataViewCtrl.

void wxDataViewListCtrl::AppendColumn (wxDataViewColumn * *column*, const wxString & *varianttype*)

Appends a column to the control and additionally appends a column to the list store with the type *varianttype*.

wxDataViewColumn* wxDataViewListCtrl::AppendIconTextColumn (const wxString & *label*, wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *width* = -1, wxAlignment *align* = wxALIGN_LEFT, int *flags* = wxDATAVIEW_COL_RESIZABLE)

Appends an icon-and-text column to the control and the store.

See wxDataViewColumn::wxDataViewColumn for more info about the parameters.

void wxDataViewListCtrl::AppendItem (const wxVector< wxVariant > & *values*, wxUIntPtr *data* = NULL)

Appends an item (=row) to the control and store.

wxDataViewColumn* wxDataViewListCtrl::AppendProgressColumn (const wxString & *label*, wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *width* = -1, wxAlignment *align* = wxALIGN_LEFT, int *flags* = wxDATAVIEW_COL_RESIZABLE)

Appends a progress column to the control and the store.

See wxDataViewColumn::wxDataViewColumn for more info about the parameters.

wxDataViewColumn* **wxDataViewListCtrl::AppendTextColumn** (**const wxString &** *label*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_INERT**, **int** *width* = **-1**, **wxAlignment** *align* = **wxALIGN_LEFT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a text column to the control and the store.

See wxDataViewColumn::wxDataViewColumn for more info about the parameters.

wxDataViewColumn* **wxDataViewListCtrl::AppendToggleColumn** (**const wxString &** *label*, **wxDataViewCellMode** *mode* = **wxDATAVIEW_CELL_ACTIVATABLE**, **int** *width* = **-1**, **wxAlignment** *align* = **wxALIGN_LEFT**, **int** *flags* = **wxDATAVIEW_COL_RESIZABLE**)

Appends a toggle column to the control and the store.

See wxDataViewColumn::wxDataViewColumn for more info about the parameters.

bool wxDataViewListCtrl::Create (**wxWindow** * *parent*, **wxWindowID** *id*, **const wxPoint &** *pos* = **wxDefaultPosition**, **const wxSize &** *size* = **wxDefaultSize**, **long** *style* = **wxDV_ROW_LINES**, **const wxValidator &** *validator* = **wxDefaultValidator**)

Creates the control and a wxDataViewListStore as its internal model.

void wxDataViewListCtrl::DeleteAllItems ( )

Delete all items (= all rows).

void wxDataViewListCtrl::DeleteItem (**unsigned** *row*)

Delete the row at position *row*.

unsigned int wxDataViewListCtrl::GetItemCount ( ) **const**

Returns the number of items (=rows) in the control.

Since

2.9.4

wxUIntPtr wxDataViewListCtrl::GetItemData (**const wxDataViewItem &** *item*) **const**

Returns the client data associated with the item.

See also

SetItemData()

Since

2.9.4

int wxDataViewListCtrl::GetSelectedRow ( ) const

Returns index of the selected row or wxNOT_FOUND.

See also

wxDataViewCtrl::GetSelection()

Since

2.9.2

wxDataViewListStore* wxDataViewListCtrl::GetStore ( )

Returns the store.

const wxDataViewListStore* wxDataViewListCtrl::GetStore ( ) const

Returns the store.

wxString wxDataViewListCtrl::GetTextValue (unsigned int *row*, unsigned int *col*) const

Returns the value from the store.

This method assumes that the string is stored in respective column.

bool wxDataViewListCtrl::GetToggleValue (unsigned int *row*, unsigned int *col*) const

Returns the value from the store.

This method assumes that the boolean value is stored in respective column.

void wxDataViewListCtrl::GetValue (wxVariant & *value*, unsigned int *row*, unsigned int *col*)

Returns the value from the store.

virtual bool wxDataViewListCtrl::InsertColumn (unsigned int *pos*, wxDataViewColumn * *column*) [virtual]

Inserts a column to the control and additionally inserts a column to the store with the type string.

Reimplemented from wxDataViewCtrl.

void wxDataViewListCtrl::InsertColumn (unsigned int *pos*, wxDataViewColumn * *column*, const wxString & *varianttype*)

Inserts a column to the control and additionally inserts a column to the list store with the type *varianttype*.

void wxDataViewListCtrl::InsertItem (unsigned int *row*, const wxVector< wxVariant > & *values*, wxUIntPtr *data* = NULL)

Inserts an item (=row) to the control and store.

bool wxDataViewListCtrl::IsRowSelected (unsigned *row*) const

Returns true if *row* is selected.

See also

wxDataViewCtrl::IsSelected()

Since

2.9.2

int wxDataViewListCtrl::ItemToRow (const wxDataViewItem & *item*) const

Returns the position of given *item* or wxNOT_FOUND if it's not a valid item.

Since

2.9.2

virtual bool wxDataViewListCtrl::PrependColumn (wxDataViewColumn * *column*) [virtual]

Prepends a column to the control and additionally prepends a column to the store with the type string.

Reimplemented from wxDataViewCtrl.

void wxDataViewListCtrl::PrependColumn (wxDataViewColumn * *column,* const wxString & *varianttype*)

Prepends a column to the control and additionally prepends a column to the list store with the type *varianttype*.

void wxDataViewListCtrl::PrependItem (const wxVector< wxVariant > & *values,* wxUIntPtr *data* = NULL)

Prepends an item (=row) to the control and store.

wxDataViewItem wxDataViewListCtrl::RowToItem (int *row*) const

Returns the wxDataViewItem at the given *row*.

Since

2.9.2

void wxDataViewListCtrl::SelectRow (unsigned *row*)

Selects given row.

See also

wxDataViewCtrl::Select()

Since

2.9.2

void wxDataViewListCtrl::SetItemData (const wxDataViewItem & *item,* wxUIntPtr *data*)

Associates a client data pointer with the given item.

Notice that the control does *not* take ownership of the pointer for compatibility with wxListCtrl. I.e. it will *not* delete the pointer (if it is a pointer and not a number) itself, it is up to you to do it.

See also

GetItemData()

Since

2.9.4

void wxDataViewListCtrl::SetTextValue (const wxString & *value,* unsigned int *row,* unsigned int *col*)

Sets the value in the store and update the control.

This method assumes that the string is stored in respective column.

void wxDataViewListCtrl::SetToggleValue (bool *value,* unsigned int *row,* unsigned int *col*)

Sets the value in the store and update the control.

This method assumes that the boolean value is stored in respective column.

void wxDataViewListCtrl::SetValue (const wxVariant & *value,* unsigned int *row,* unsigned int *col*)

Sets the value in the store and update the control.

void wxDataViewListCtrl::UnselectRow (unsigned *row*)

Unselects given row.

See also

wxDataViewCtrl::Unselect()

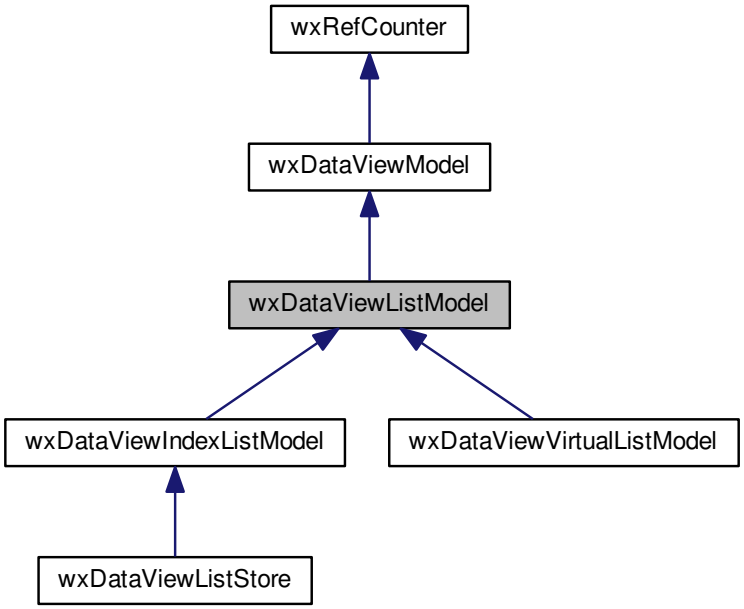
Since

2.9.2

21.155 wxDataViewListModel Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewListModel:



21.155.1 Detailed Description

Base class with abstract API for wxDataViewIndexListModel and wxDataViewVirtualListModel.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- virtual ~wxDataViewListModel ()

  Destructor.
- int Compare (const wxDataViewItem &item1, const wxDataViewItem &item2, unsigned int column, bool ascending) const

  Compare method that sorts the items by their index.
- virtual bool GetAttrByRow (unsigned int row, unsigned int col, wxDataViewItemAttr &attr) const

  Override this to indicate that the row has special font attributes.
- virtual bool IsEnabledByRow (unsigned int row, unsigned int col) const

  Override this if you want to disable specific items.
- unsigned int GetCount () const =0

  Returns the number of items (or rows) in the list.
- unsigned int GetRow (const wxDataViewItem &item) const =0

  Returns the position of given item.

- virtual void GetValueByRow (wxVariant &variant, unsigned int row, unsigned int col) const =0

  Override this to allow getting values from the model.
- virtual bool SetValueByRow (const wxVariant &variant, unsigned int row, unsigned int col)=0

  Called in order to set a value in the model.

Additional Inherited Members

21.155.2 Constructor & Destructor Documentation

virtual wxDataViewListModel::~wxDataViewListModel ( ) [virtual]

Destructor.

21.155.3 Member Function Documentation

int wxDataViewListModel::Compare (const wxDataViewItem & *item1,* const wxDataViewItem & *item2,* unsigned int *column,* bool *ascending*) const [virtual]

Compare method that sorts the items by their index.

Reimplemented from wxDataViewModel.

virtual bool wxDataViewListModel::GetAttrByRow (unsigned int *row,* unsigned int *col,* wxDataViewItemAttr & *attr*) const [virtual]

Override this to indicate that the row has special font attributes.

This only affects the wxDataViewTextRendererText() renderer.

The base class version always simply returns false.

See also

wxDataViewItemAttr.

Parameters

row	The row for which the attribute is requested.
col	The column for which the attribute is requested.
attr	The attribute to be filled in if the function returns true.

Returns

true if this item has an attribute or false otherwise.

unsigned int wxDataViewListModel::GetCount ( ) const [pure virtual]

Returns the number of items (or rows) in the list.

unsigned int wxDataViewListModel::GetRow (const wxDataViewItem & *item*) const [pure virtual]

Returns the position of given *item*.

virtual void wxDataViewListModel::GetValueByRow (wxVariant & *variant,* unsigned int *row,* unsigned int *col*) const `[pure virtual]`

Override this to allow getting values from the model.

Implemented in wxDataViewListStore.

virtual bool wxDataViewListModel::IsEnabledByRow (unsigned int *row,* unsigned int *col*) const `[virtual]`

Override this if you want to disable specific items.

The base class version always returns true, thus making all items enabled by default.

Parameters

row	The row of the item whose enabled status is requested.
col	The column of the item whose enabled status is requested.

Returns

true if the item at this row and column should be enabled, false otherwise.

Note

See wxDataViewModel::IsEnabled() for the current status of support for disabling the items under different platforms.

Since

2.9.2

virtual bool wxDataViewListModel::SetValueByRow (const wxVariant & *variant,* unsigned int *row,* unsigned int *col*) `[pure virtual]`

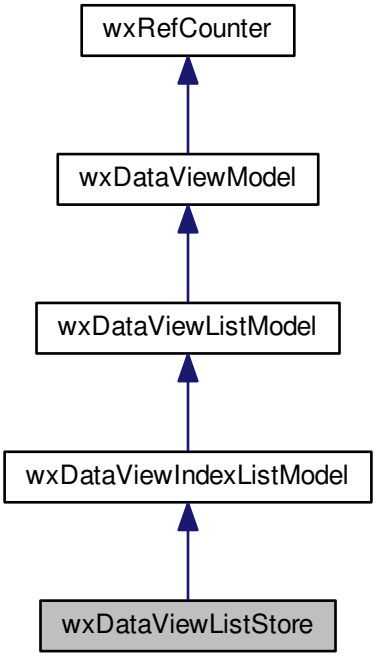
Called in order to set a value in the model.

Implemented in wxDataViewListStore.

21.156 wxDataViewListStore Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewListStore:



21.156.1 Detailed Description

wxDataViewListStore is a specialised wxDataViewModel for storing a simple table of data.

Since it derives from wxDataViewIndexListModel its data is be accessed by row (i.e. by index) instead of only by wxDataViewItem.

This class actually stores the values (therefore its name) and implements all virtual methods from the base classes so it can be used directly without having to derive any class from it, but it is mostly used from within wxDataView↩ ListCtrl.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewListStore ()

  Constructor.
- ∼wxDataViewListStore ()

  Destructor.
- void PrependColumn (const wxString &varianttype)

  Prepends a data column.

- void InsertColumn (unsigned int pos, const wxString &varianttype)

  Inserts a data column before pos.
- void AppendColumn (const wxString &varianttype)

  Appends a data column.
- void AppendItem (const wxVector< wxVariant > &values, wxUIntPtr data=NULL)

  Appends an item (=row) and fills it with values.
- void PrependItem (const wxVector< wxVariant > &values, wxUIntPtr data=NULL)

  Prepends an item (=row) and fills it with values.
- void InsertItem (unsigned int row, const wxVector< wxVariant > &values, wxUIntPtr data=NULL)

  Inserts an item (=row) and fills it with values.
- void DeleteItem (unsigned pos)

  Delete the item (=row) at position pos.
- void DeleteAllItems ()

  Delete all item (=all rows) in the store.
- unsigned int GetItemCount () const

  Returns the number of items (=rows) in the control.
- wxUIntPtr GetItemData (const wxDataViewItem &item) const

  Returns the client data associated with the item.
- virtual unsigned int GetColumnCount () const

  Overridden from wxDataViewModel.
- virtual wxString GetColumnType (unsigned int col) const

  Overridden from wxDataViewModel.
- void SetItemData (const wxDataViewItem &item, wxUIntPtr data)

  Sets the client data associated with the item.
- virtual void GetValueByRow (wxVariant &value, unsigned int row, unsigned int col) const

  Overridden from wxDataViewIndexListModel.
- virtual bool SetValueByRow (const wxVariant &value, unsigned int row, unsigned int col)

  Overridden from wxDataViewIndexListModel.

Additional Inherited Members

21.156.2 Constructor & Destructor Documentation

wxDataViewListStore::wxDataViewListStore ( )

Constructor.

wxDataViewListStore::~wxDataViewListStore ( )

Destructor.

21.156.3 Member Function Documentation

void wxDataViewListStore::AppendColumn (const wxString & *varianttype*)

Appends a data column.

variantype indicates the type of values store in the column.

This does not automatically fill in any (default) values in rows which exist in the store already.

void wxDataViewListStore::AppendItem (const wxVector< wxVariant > & *values*, wxUIntPtr *data* = NULL)

Appends an item (=row) and fills it with *values*.

The values must match the values specifies in the column in number and type. No (default) values are filled in automatically.

void wxDataViewListStore::DeleteAllItems ( )

Delete all item (=all rows) in the store.

void wxDataViewListStore::DeleteItem (unsigned *pos*)

Delete the item (=row) at position *pos*.

virtual unsigned int wxDataViewListStore::GetColumnCount ( ) const [virtual]

Overridden from wxDataViewModel.

Implements wxDataViewModel.

virtual wxString wxDataViewListStore::GetColumnType (unsigned int *col*) const [virtual]

Overridden from wxDataViewModel.

Implements wxDataViewModel.

unsigned int wxDataViewListStore::GetItemCount ( ) const

Returns the number of items (=rows) in the control.

Since

2.9.4

wxUIntPtr wxDataViewListStore::GetItemData (const wxDataViewItem & *item*) const

Returns the client data associated with the item.

See also

SetItemData()

Since

2.9.4

virtual void wxDataViewListStore::GetValueByRow (wxVariant & *value*, unsigned int *row*, unsigned int *col*) const [virtual]

Overridden from wxDataViewIndexListModel.

Implements wxDataViewListModel.

void wxDataViewListStore::InsertColumn (unsigned int *pos*, const wxString & *varianttype*)

Inserts a data column before *pos*.

variantype indicates the type of values store in the column.

This does not automatically fill in any (default) values in rows which exist in the store already.

void wxDataViewListStore::InsertItem (unsigned int *row*, const wxVector< wxVariant > & *values*, wxUIntPtr *data* = NULL)

Inserts an item (=row) and fills it with *values*.

The values must match the values specifies in the column in number and type. No (default) values are filled in automatically.

void wxDataViewListStore::PrependColumn (const wxString & *varianttype*)

Prepends a data column.

variantype indicates the type of values store in the column.

This does not automatically fill in any (default) values in rows which exist in the store already.

void wxDataViewListStore::PrependItem (const wxVector< wxVariant > & *values*, wxUIntPtr *data* = NULL)

Prepends an item (=row) and fills it with *values*.

The values must match the values specifies in the column in number and type. No (default) values are filled in automatically.

void wxDataViewListStore::SetItemData (const wxDataViewItem & *item*, wxUIntPtr *data*)

Sets the client data associated with the item.

Notice that this class does *not* take ownership of the passed in pointer and will not delete it.

See also

GetItemData()

Since

2.9.4

virtual bool wxDataViewListStore::SetValueByRow (const wxVariant & *value*, unsigned int *row*, unsigned int *col*) [virtual]

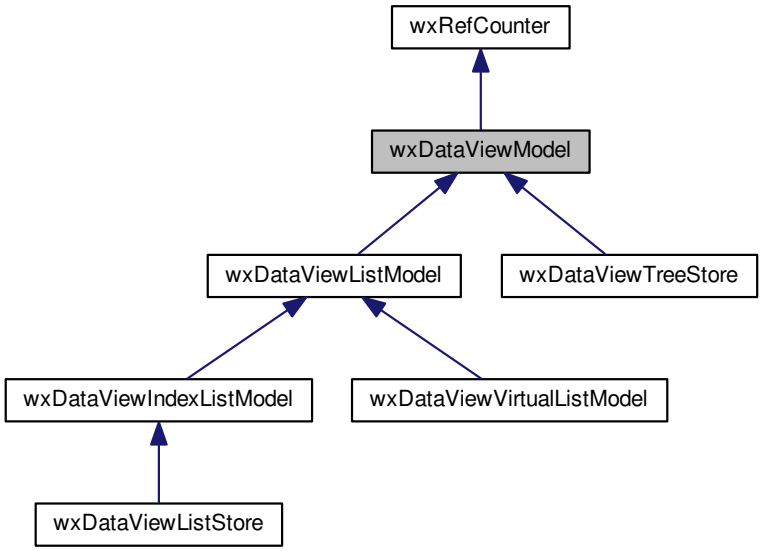
Overridden from wxDataViewIndexListModel.

Implements wxDataViewListModel.

21.157 wxDataViewModel Class Reference

```
#include <wx/dataview.h>
```

Inheritance diagram for wxDataViewModel:



21.157.1 Detailed Description

wxDataViewModel is the base class for all data model to be displayed by a wxDataViewCtrl.

All other models derive from it and must implement its pure virtual functions in order to define a complete data model. In detail, you need to override wxDataViewModel::IsContainer, wxDataViewModel::GetParent, wxDataViewModel::GetChildren, wxDataViewModel::GetColumnCount, wxDataViewModel::GetColumnType and wxDataViewModel::GetValue in order to define the data model which acts as an interface between your actual data and the wxDataViewCtrl.

Note that wxDataViewModel does not define the position or index of any item in the control because different controls might display the same data differently. wxDataViewModel does provide a wxDataViewModel::Compare method which the wxDataViewCtrl may use to sort the data either in conjunction with a column header or without (see wxDataViewModel::HasDefaultCompare).

wxDataViewModel (as indeed the entire wxDataViewCtrl code) is using wxVariant to store data and its type in a generic way. wxVariant can be extended to contain almost any data without changes to the original class. To a certain extent, you can use (the somewhat more elegant) wxAny instead of wxVariant as there is code to convert between the two, but it is unclear what impact this will have on performance.

Since you will usually allow the wxDataViewCtrl to change your data through its graphical interface, you will also have to override wxDataViewModel::SetValue which the wxDataViewCtrl will call when a change to some data has been committed.

If the data represented by the model is changed by something else than its associated wxDataViewCtrl, the control has to be notified about the change. Depending on what happened you need to call one of the following methods:

- wxDataViewModel::ValueChanged,
- wxDataViewModel::ItemAdded,
- wxDataViewModel::ItemDeleted,
- wxDataViewModel::ItemChanged,

- wxDataViewModel::Cleared.

There are plural forms for notification of addition, change or removal of several item at once. See:

- wxDataViewModel::ItemsAdded,
- wxDataViewModel::ItemsDeleted,
- wxDataViewModel::ItemsChanged.

This class maintains a list of wxDataViewModelNotifier which link this class to the specific implementations on the supported platforms so that e.g. calling wxDataViewModel::ValueChanged on this model will just call wxDataViewModelNotifier::ValueChanged for each notifier that has been added. You can also add your own notifier in order to get informed about any changes to the data in the list model.

Currently wxWidgets provides the following models apart from the base model: wxDataViewIndexListModel, wxDataViewVirtualListModel, wxDataViewTreeStore, wxDataViewListStore.

Note that wxDataViewModel is reference counted, derives from wxRefCounter and cannot be deleted directly as it can be shared by several wxDataViewCtrls. This implies that you need to decrease the reference count after associating the model with a control like this:

```
wxDataViewCtrl *musicCtrl = new wxDataViewCtrl( this,
      wxID_ANY );
wxDataViewModel *musicModel = new MyMusicModel;
m_musicCtrl->AssociateModel( musicModel );
musicModel->DecRef();  // avoid memory leak !!

// add columns now
```

A potentially better way to avoid memory leaks is to use wxObjectDataPtr

```
wxObjectDataPtr<MyMusicModel> musicModel;

wxDataViewCtrl *musicCtrl = new wxDataViewCtrl( this,
      wxID_ANY );
musicModel = new MyMusicModel;
m_musicCtrl->AssociateModel( musicModel.get() );

// add columns now
```

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewModel ()

  Constructor.
- void AddNotifier (wxDataViewModelNotifier *notifier)

  Adds a wxDataViewModelNotifier to the model.
- bool ChangeValue (const wxVariant &variant, const wxDataViewItem &item, unsigned int col)

  Change the value of the given item and update the control to reflect it.
- virtual bool Cleared ()

  Called to inform the model that all data has been cleared.
- virtual int Compare (const wxDataViewItem &item1, const wxDataViewItem &item2, unsigned int column, bool ascending) const

  The compare function to be used by control.
- virtual bool GetAttr (const wxDataViewItem &item, unsigned int col, wxDataViewItemAttr &attr) const

Override this to indicate that the item has special font attributes.

- virtual bool IsEnabled (const wxDataViewItem &item, unsigned int col) const

  Override this to indicate that the item should be disabled.
- virtual unsigned int GetChildren (const wxDataViewItem &item, wxDataViewItemArray &children) const =0

  Override this so the control can query the child items of an item.
- virtual unsigned int GetColumnCount () const =0

  Override this to indicate the number of columns in the model.
- virtual wxString GetColumnType (unsigned int col) const =0

  Override this to indicate what type of data is stored in the column specified by col.
- virtual wxDataViewItem GetParent (const wxDataViewItem &item) const =0

  Override this to indicate which wxDataViewItem representing the parent of item or an invalid wxDataViewItem if the root item is the parent item.
- virtual void GetValue (wxVariant &variant, const wxDataViewItem &item, unsigned int col) const =0

  Override this to indicate the value of item.
- virtual bool HasContainerColumns (const wxDataViewItem &item) const

  Override this method to indicate if a container item merely acts as a headline (or for categorisation) or if it also acts a normal item with entries for further columns.
- virtual bool HasDefaultCompare () const

  Override this to indicate that the model provides a default compare function that the control should use if no wxDataViewColumn has been chosen for sorting.
- bool HasValue (const wxDataViewItem &item, unsigned col) const

  Return true if there is a value in the given column of this item.
- virtual bool IsContainer (const wxDataViewItem &item) const =0

  Override this to indicate of item is a container, i.e. if it can have child items.
- bool ItemAdded (const wxDataViewItem &parent, const wxDataViewItem &item)

  Call this to inform the model that an item has been added to the data.
- bool ItemChanged (const wxDataViewItem &item)

  Call this to inform the model that an item has changed.
- bool ItemDeleted (const wxDataViewItem &parent, const wxDataViewItem &item)

  Call this to inform the model that an item has been deleted from the data.
- bool ItemsAdded (const wxDataViewItem &parent, const wxDataViewItemArray &items)

  Call this to inform the model that several items have been added to the data.
- bool ItemsChanged (const wxDataViewItemArray &items)

  Call this to inform the model that several items have changed.
- bool ItemsDeleted (const wxDataViewItem &parent, const wxDataViewItemArray &items)

  Call this to inform the model that several items have been deleted.
- void RemoveNotifier (wxDataViewModelNotifier *notifier)

  Remove the notifier from the list of notifiers.
- virtual void Resort ()

  Call this to initiate a resort after the sort function has been changed.
- virtual bool SetValue (const wxVariant &variant, const wxDataViewItem &item, unsigned int col)=0

  This gets called in order to set a value in the data model.
- virtual bool ValueChanged (const wxDataViewItem &item, unsigned int col)

  Call this to inform this model that a value in the model has been changed.
- virtual bool IsListModel () const
- virtual bool IsVirtualListModel () const

Protected Member Functions

- virtual ~wxDataViewModel ()

  Destructor.

21.157.2 Constructor & Destructor Documentation

wxDataViewModel::wxDataViewModel ( )

Constructor.

virtual wxDataViewModel::~wxDataViewModel ( ) [protected], [virtual]

Destructor.

This should not be called directly. Use DecRef() instead.

21.157.3 Member Function Documentation

void wxDataViewModel::AddNotifier (wxDataViewModelNotifier * *notifier*)

Adds a wxDataViewModelNotifier to the model.

bool wxDataViewModel::ChangeValue (const wxVariant & *variant*, const wxDataViewItem & *item*, unsigned int *col*)

Change the value of the given item and update the control to reflect it.

This function simply calls SetValue() and, if it succeeded, ValueChanged().

Since

2.9.1

Parameters

variant	The new value.
item	The item (row) to update.
col	The column to update.

Returns

true if both SetValue() and ValueChanged() returned true.

virtual bool wxDataViewModel::Cleared ( ) [virtual]

Called to inform the model that all data has been cleared.

The control will reread the data from the model again.

virtual int wxDataViewModel::Compare (const wxDataViewItem & *item1*, const wxDataViewItem & *item2*, unsigned int *column*, bool *ascending*) const [virtual]

The compare function to be used by control.

The default compare function sorts by container and other items separately and in ascending order. Override this for a different sorting behaviour.

See also

HasDefaultCompare().

Reimplemented in wxDataViewListModel.

virtual bool wxDataViewModel::GetAttr (const wxDataViewItem & *item,* unsigned int *col,* wxDataViewItemAttr & *attr*) const `[virtual]`

Override this to indicate that the item has special font attributes.

This only affects the wxDataViewTextRendererText renderer.

The base class version always simply returns false.

See also

wxDataViewItemAttr.

Parameters

item	The item for which the attribute is requested.
col	The column of the item for which the attribute is requested.
attr	The attribute to be filled in if the function returns true.

Returns

true if this item has an attribute or false otherwise.

virtual unsigned int wxDataViewModel::GetChildren (const wxDataViewItem & *item,* wxDataViewItemArray & *children*) const `[pure virtual]`

Override this so the control can query the child items of an item.

Returns the number of items.

virtual unsigned int wxDataViewModel::GetColumnCount ( ) const `[pure virtual]`

Override this to indicate the number of columns in the model.

Implemented in wxDataViewListStore.

virtual wxString wxDataViewModel::GetColumnType (unsigned int *col*) const `[pure virtual]`

Override this to indicate what type of data is stored in the column specified by *col*.

This should return a string indicating the type of data as reported by wxVariant.

Implemented in wxDataViewListStore.

virtual wxDataViewItem wxDataViewModel::GetParent (const wxDataViewItem & *item*) const `[pure virtual]`

Override this to indicate which wxDataViewItem representing the parent of *item* or an invalid wxDataViewItem if the root item is the parent item.

virtual void wxDataViewModel::GetValue (wxVariant & *variant,* const wxDataViewItem & *item,* unsigned int *col*) const `[pure virtual]`

Override this to indicate the value of *item*.

A wxVariant is used to store the data.

virtual bool wxDataViewModel::HasContainerColumns (const wxDataViewItem & *item*) const [virtual]

Override this method to indicate if a container item merely acts as a headline (or for categorisation) or if it also acts a normal item with entries for further columns.

By default returns false.

virtual bool wxDataViewModel::HasDefaultCompare ( ) const [virtual]

Override this to indicate that the model provides a default compare function that the control should use if no wxDataViewColumn has been chosen for sorting.

Usually, the user clicks on a column header for sorting, the data will be sorted alphanumerically.

If any other order (e.g. by index or order of appearance) is required, then this should be used. See wxDataViewIndexListModel for a model which makes use of this.

bool wxDataViewModel::HasValue (const wxDataViewItem & *item*, unsigned *col*) const

Return true if there is a value in the given column of this item.

All normal items have values in all columns but the container items only show their label in the first column (*col* == 0) by default (but see HasContainerColumns()). So this function always returns true for the first column while for the other ones it returns true only if the item is not a container or HasContainerColumns() was overridden to return true for it.

Since

2.9.1

virtual bool wxDataViewModel::IsContainer (const wxDataViewItem & *item*) const [pure virtual]

Override this to indicate of *item* is a container, i.e. if it can have child items.

virtual bool wxDataViewModel::IsEnabled (const wxDataViewItem & *item*, unsigned int *col*) const [virtual]

Override this to indicate that the item should be disabled.

Disabled items are displayed differently (e.g. grayed out) and cannot be interacted with.

The base class version always returns true, thus making all items enabled by default.

Parameters

item	The item whose enabled status is requested.
col	The column of the item whose enabled status is requested.

Returns

true if this item should be enabled, false otherwise.

Note

Currently disabling items is not supported by the wxOSX/Carbon implementation.

Since

2.9.2

virtual bool wxDataViewModel::IsListModel ( ) const `[virtual]`

virtual bool wxDataViewModel::IsVirtualListModel ( ) const `[virtual]`

bool wxDataViewModel::ItemAdded (const wxDataViewItem & *parent,* const wxDataViewItem & *item*)

Call this to inform the model that an item has been added to the data.

bool wxDataViewModel::ItemChanged (const wxDataViewItem & *item*)

Call this to inform the model that an item has changed.

This will eventually emit a `wxEVT_DATAVIEW_ITEM_VALUE_CHANGED` event (in which the column fields will not be set) to the user.

bool wxDataViewModel::ItemDeleted (const wxDataViewItem & *parent,* const wxDataViewItem & *item*)

Call this to inform the model that an item has been deleted from the data.

bool wxDataViewModel::ItemsAdded (const wxDataViewItem & *parent,* const wxDataViewItemArray & *items*)

Call this to inform the model that several items have been added to the data.

bool wxDataViewModel::ItemsChanged (const wxDataViewItemArray & *items*)

Call this to inform the model that several items have changed.

This will eventually emit `wxEVT_DATAVIEW_ITEM_VALUE_CHANGED` events (in which the column fields will not be set) to the user.

bool wxDataViewModel::ItemsDeleted (const wxDataViewItem & *parent,* const wxDataViewItemArray & *items*)

Call this to inform the model that several items have been deleted.

void wxDataViewModel::RemoveNotifier (wxDataViewModelNotifier * *notifier*)

Remove the *notifier* from the list of notifiers.

virtual void wxDataViewModel::Resort ( ) `[virtual]`

Call this to initiate a resort after the sort function has been changed.

virtual bool wxDataViewModel::SetValue (const wxVariant & *variant,* const wxDataViewItem & *item,* unsigned int *col*) `[pure virtual]`

This gets called in order to set a value in the data model.

The most common scenario is that the wxDataViewCtrl calls this method after the user changed some data in the view.

This is the function you need to override in your derived class but if you want to call it, ChangeValue() is usually more convenient as otherwise you need to manually call ValueChanged() to update the control itself.

virtual bool wxDataViewModel::ValueChanged (const wxDataViewItem & *item,* unsigned int *col*) `[virtual]`

Call this to inform this model that a value in the model has been changed.

This is also called from wxDataViewCtrl's internal editing code, e.g. when editing a text field in the control.

This will eventually emit a `wxEVT_DATAVIEW_ITEM_VALUE_CHANGED` event to the user.

21.158 wxDataViewModelNotifier Class Reference

`#include <wx/dataview.h>`

21.158.1 Detailed Description

A wxDataViewModelNotifier instance is owned by a wxDataViewModel and mirrors its notification interface.

See the documentation of that class for further information.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewModelNotifier ()
  Constructor.
- virtual ~wxDataViewModelNotifier ()
  Destructor.
- virtual bool Cleared ()=0
  Called by owning model.
- wxDataViewModel * GetOwner () const
  Get owning wxDataViewModel.
- virtual bool ItemAdded (const wxDataViewItem &parent, const wxDataViewItem &item)=0
  Called by owning model.
- virtual bool ItemChanged (const wxDataViewItem &item)=0
  Called by owning model.
- virtual bool ItemDeleted (const wxDataViewItem &parent, const wxDataViewItem &item)=0
  Called by owning model.
- virtual bool ItemsAdded (const wxDataViewItem &parent, const wxDataViewItemArray &items)
  Called by owning model.
- virtual bool ItemsChanged (const wxDataViewItemArray &items)
  Called by owning model.
- virtual bool ItemsDeleted (const wxDataViewItem &parent, const wxDataViewItemArray &items)
  Called by owning model.
- virtual void Resort ()=0
  Called by owning model.
- void SetOwner (wxDataViewModel *owner)
  Set owner of this notifier.
- virtual bool ValueChanged (const wxDataViewItem &item, unsigned int col)=0
  Called by owning model.

21.158.2 Constructor & Destructor Documentation

wxDataViewModelNotifier::wxDataViewModelNotifier ( )

Constructor.

virtual wxDataViewModelNotifier::~wxDataViewModelNotifier ( ) [virtual]

Destructor.

21.158.3 Member Function Documentation

virtual bool wxDataViewModelNotifier::Cleared ( ) [pure virtual]

Called by owning model.

wxDataViewModel* wxDataViewModelNotifier::GetOwner ( ) const

Get owning wxDataViewModel.

virtual bool wxDataViewModelNotifier::ItemAdded (const wxDataViewItem & *parent,* const wxDataViewItem & *item*) [pure virtual]

Called by owning model.

Returns

Always return true from this function in derived classes.

virtual bool wxDataViewModelNotifier::ItemChanged (const wxDataViewItem & *item*) [pure virtual]

Called by owning model.

Returns

Always return true from this function in derived classes.

virtual bool wxDataViewModelNotifier::ItemDeleted (const wxDataViewItem & *parent,* const wxDataViewItem & *item*) [pure virtual]

Called by owning model.

Returns

Always return true from this function in derived classes.

virtual bool wxDataViewModelNotifier::ItemsAdded (const wxDataViewItem & *parent,* const wxDataViewItemArray & *items*) [virtual]

Called by owning model.

Returns

Always return true from this function in derived classes.

virtual bool wxDataViewModelNotifier::ItemsChanged (const wxDataViewItemArray & *items*) `[virtual]`

Called by owning model.

Returns

Always return true from this function in derived classes.

virtual bool wxDataViewModelNotifier::ItemsDeleted (const wxDataViewItem & *parent,* const wxDataViewItemArray & *items*) `[virtual]`

Called by owning model.

Returns

Always return true from this function in derived classes.

virtual void wxDataViewModelNotifier::Resort ( ) `[pure virtual]`

Called by owning model.

void wxDataViewModelNotifier::SetOwner (wxDataViewModel * *owner*)

Set owner of this notifier.

Used internally.

virtual bool wxDataViewModelNotifier::ValueChanged (const wxDataViewItem & *item,* unsigned int *col*) `[pure virtual]`

Called by owning model.

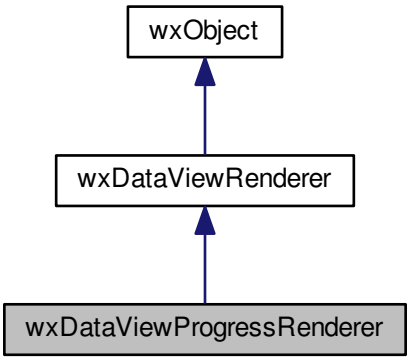
Returns

Always return true from this function in derived classes.

21.159 wxDataViewProgressRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewProgressRenderer:



21.159.1 Detailed Description

This class is used by wxDataViewCtrl to render progress bars.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewProgressRenderer (const wxString &label=wxEmptyString, const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Additional Inherited Members

21.159.2 Constructor & Destructor Documentation

wxDataViewProgressRenderer::wxDataViewProgressRenderer (const wxString & *label* = wxEmptyString, const wxString & *varianttype* = GetDefaultType (), wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.159.3 Member Function Documentation

static wxString wxDataViewProgressRenderer::GetDefaultType ( ) `[static]`

Returns the wxVariant type used with this renderer.

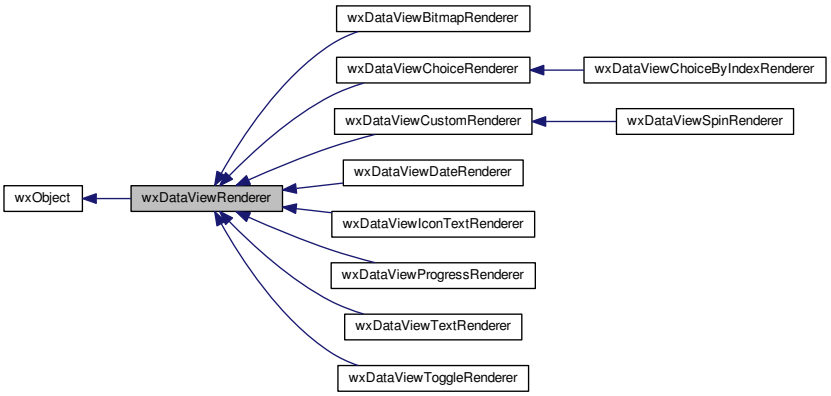
Since

3.1.0

21.160 wxDataViewRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewRenderer:



21.160.1 Detailed Description

This class is used by wxDataViewCtrl to render the individual cells.

One instance of a renderer class is owned by a wxDataViewColumn. There is a number of ready-to-use renderers provided:

- wxDataViewTextRenderer,
- wxDataViewIconTextRenderer,
- wxDataViewToggleRenderer,

- wxDataViewProgressRenderer,
- wxDataViewBitmapRenderer,
- wxDataViewDateRenderer,
- wxDataViewSpinRenderer.
- wxDataViewChoiceRenderer.

Additionally, the user can write their own renderers by deriving from wxDataViewCustomRenderer.

The wxDataViewCellMode and wxDataViewCellRenderState flags accepted by the constructors respectively controls what actions the cell data allows and how the renderer should display its contents in a cell.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewRenderer (const wxString &varianttype, wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  Constructor.
- void EnableEllipsize (wxEllipsizeMode mode=wxELLIPSIZE_MIDDLE)

  Enable or disable replacing parts of the item text with ellipsis to make it fit the column width.
- void DisableEllipsize ()

  Disable replacing parts of the item text with ellipsis.
- virtual int GetAlignment () const

  Returns the alignment.
- wxEllipsizeMode GetEllipsizeMode () const

  Returns the ellipsize mode used by the renderer.
- virtual wxDataViewCellMode GetMode () const

  Returns the cell mode.
- wxDataViewColumn * GetOwner () const

  Returns pointer to the owning wxDataViewColumn.
- virtual bool GetValue (wxVariant &value) const =0

  This methods retrieves the value from the renderer in order to transfer the value back to the data model.
- wxString GetVariantType () const

  Returns a string with the type of the wxVariant supported by this renderer.
- virtual void SetAlignment (int align)

  Sets the alignment of the renderer's content.
- void SetOwner (wxDataViewColumn *owner)

  Sets the owning wxDataViewColumn.
- virtual bool SetValue (const wxVariant &value)=0

  Set the value of the renderer (and thus its cell) to value.
- virtual bool Validate (wxVariant &value)

  Before data is committed to the data model, it is passed to this method where it can be checked for validity.
- virtual bool HasEditorCtrl () const
- virtual wxWindow * CreateEditorCtrl (wxWindow *parent, wxRect labelRect, const wxVariant &value)
- virtual bool GetValueFromEditorCtrl (wxWindow *editor, wxVariant &value)
- virtual bool StartEditing (const wxDataViewItem &item, wxRect labelRect)
- virtual void CancelEditing ()
- virtual bool FinishEditing ()
- wxWindow * GetEditorCtrl ()

Protected Member Functions

- wxDataViewCtrl * GetView () const

Additional Inherited Members

21.160.2 Constructor & Destructor Documentation

wxDataViewRenderer::wxDataViewRenderer (const wxString & *varianttype,* wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

Constructor.

21.160.3 Member Function Documentation

virtual void wxDataViewRenderer::CancelEditing ( ) [virtual]

virtual wxWindow* wxDataViewRenderer::CreateEditorCtrl (wxWindow * *parent,* wxRect *labelRect,* const wxVariant & *value*) [virtual]

Reimplemented in wxDataViewCustomRenderer.

void wxDataViewRenderer::DisableEllipsize ( )

Disable replacing parts of the item text with ellipsis.

If ellipsizing is disabled, the string will be truncated if it doesn't fit.

This is the same as

```
EnableEllipsize(wxELLIPSIZE_NONE)
```

.

Since

2.9.1

void wxDataViewRenderer::EnableEllipsize (wxEllipsizeMode *mode* = wxELLIPSIZE_MIDDLE)

Enable or disable replacing parts of the item text with ellipsis to make it fit the column width.

This method only makes sense for the renderers working with text, such as wxDataViewTextRenderer or wxDataViewIconTextRenderer.

By default wxELLIPSIZE_MIDDLE is used.

Parameters

mode	Ellipsization mode, use wxELLIPSIZE_NONE to disable.

Since

2.9.1

virtual bool wxDataViewRenderer::FinishEditing ( ) [virtual]

virtual int wxDataViewRenderer::GetAlignment ( ) const [virtual]

Returns the alignment.

See SetAlignment()

wxWindow* wxDataViewRenderer::GetEditorCtrl ( )

wxEllipsizeMode wxDataViewRenderer::GetEllipsizeMode ( ) const

Returns the ellipsize mode used by the renderer.

If the return value is wxELLIPSIZE_NONE, the text is simply truncated if it doesn't fit.

See also

EnableEllipsize()

virtual wxDataViewCellMode wxDataViewRenderer::GetMode ( ) const [virtual]

Returns the cell mode.

wxDataViewColumn* wxDataViewRenderer::GetOwner ( ) const

Returns pointer to the owning wxDataViewColumn.

virtual bool wxDataViewRenderer::GetValue (wxVariant & *value*) const [pure virtual]

This methods retrieves the value from the renderer in order to transfer the value back to the data model.

Returns false on failure.

virtual bool wxDataViewRenderer::GetValueFromEditorCtrl (wxWindow * *editor*, wxVariant & *value*) [virtual]

Reimplemented in wxDataViewCustomRenderer.

wxString wxDataViewRenderer::GetVariantType ( ) const

Returns a string with the type of the wxVariant supported by this renderer.

wxDataViewCtrl* wxDataViewRenderer::GetView ( ) const [protected]

virtual bool wxDataViewRenderer::HasEditorCtrl ( ) const [virtual]

Reimplemented in wxDataViewCustomRenderer.

virtual void wxDataViewRenderer::SetAlignment (int *align*) [virtual]

Sets the alignment of the renderer's content.

The default value of `wxDVR_DEFAULT_ALIGMENT` indicates that the content should have the same alignment as the column header.

The method is not implemented under OS X and the renderer always aligns its contents as the column header on that platform. The other platforms support both vertical and horizontal alignment.

void wxDataViewRenderer::SetOwner (wxDataViewColumn * *owner*)

Sets the owning wxDataViewColumn.

This is usually called from within wxDataViewColumn.

virtual bool wxDataViewRenderer::SetValue (const wxVariant & *value*) `[pure virtual]`

Set the value of the renderer (and thus its cell) to *value*.

The internal code will then render this cell with this data.

virtual bool wxDataViewRenderer::StartEditing (const wxDataViewItem & *item*, wxRect *labelRect*) `[virtual]`

virtual bool wxDataViewRenderer::Validate (wxVariant & *value*) `[virtual]`

Before data is committed to the data model, it is passed to this method where it can be checked for validity.

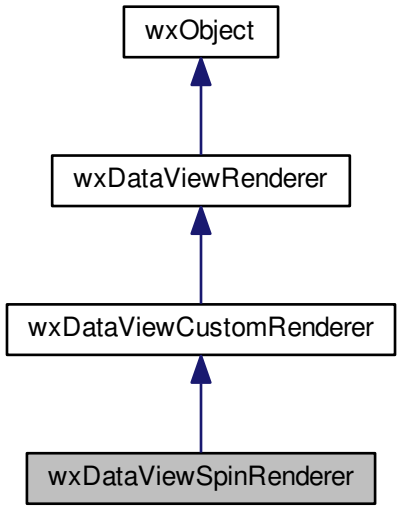
This can also be used for checking a valid range or limiting the user input in a certain aspect (e.g. max number of characters or only alphanumeric input, ASCII only etc.). Return false if the value is not valid.

Please note that due to implementation limitations, this validation is done after the editing control already is destroyed and the editing process finished.

21.161 wxDataViewSpinRenderer Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewSpinRenderer:



21.161.1 Detailed Description

This is a specialized renderer for rendering integer values.

It supports modifying the values in-place by using a wxSpinCtrl. The renderer only support variants of type *long*.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewSpinRenderer (int min, int max, wxDataViewCellMode mode=wxDATAVIEW_CELL_EDITABLE, int align=wxDVR_DEFAULT_ALIGNMENT)

  Constructor.

Additional Inherited Members

21.161.2 Constructor & Destructor Documentation

wxDataViewSpinRenderer::wxDataViewSpinRenderer (int *min,* int *max,* wxDataViewCellMode *mode =* wxDATAVIEW_CELL_EDITABLE, int *align =* wxDVR_DEFAULT_ALIGNMENT)

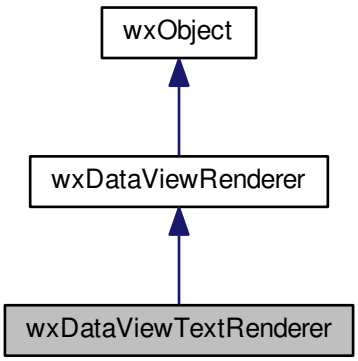
Constructor.

min and *max* indicate the minimum and maximum values for the wxSpinCtrl.

21.162 wxDataViewTextRenderer Class Reference

```
#include <wx/dataview.h>
```

Inheritance diagram for wxDataViewTextRenderer:



21.162.1 Detailed Description

wxDataViewTextRenderer is used for rendering text.

It supports in-place editing if desired.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewTextRenderer (const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Additional Inherited Members

21.162.2 Constructor & Destructor Documentation

wxDataViewTextRenderer::wxDataViewTextRenderer (const wxString & *varianttype* = GetDefaultType (), wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.162.3 Member Function Documentation

static wxString wxDataViewTextRenderer::GetDefaultType ( ) `[static]`

Returns the wxVariant type used with this renderer.

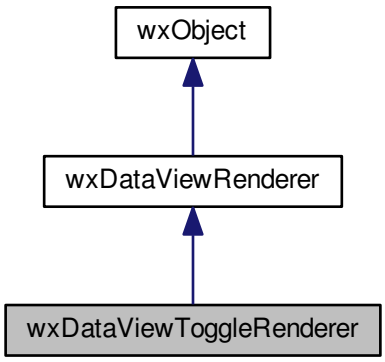
Since

3.1.0

21.163 wxDataViewToggleRenderer Class Reference

```
#include <wx/dataview.h>
```

Inheritance diagram for wxDataViewToggleRenderer:



21.163.1 Detailed Description

This class is used by wxDataViewCtrl to render toggle controls.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewToggleRenderer (const wxString &varianttype=GetDefaultType(), wxDataViewCellMode mode=wxDATAVIEW_CELL_INERT, int align=wxDVR_DEFAULT_ALIGNMENT)

  The ctor.

Static Public Member Functions

- static wxString GetDefaultType ()

  Returns the wxVariant type used with this renderer.

Additional Inherited Members

21.163.2 Constructor & Destructor Documentation

wxDataViewToggleRenderer::wxDataViewToggleRenderer (const wxString & *varianttype* = GetDefaultType ( ), wxDataViewCellMode *mode* = wxDATAVIEW_CELL_INERT, int *align* = wxDVR_DEFAULT_ALIGNMENT)

The ctor.

21.163.3 Member Function Documentation

static wxString wxDataViewToggleRenderer::GetDefaultType ( ) [static]

Returns the wxVariant type used with this renderer.

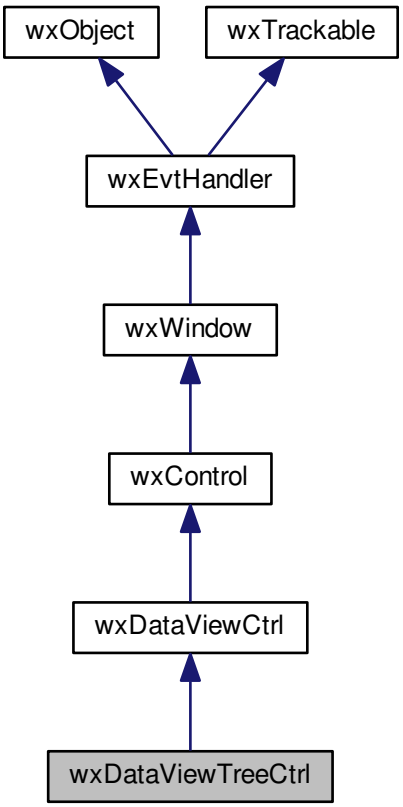
Since

3.1.0

21.164 wxDataViewTreeCtrl Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewTreeCtrl:



21.164.1 Detailed Description

This class is a wxDataViewCtrl which internally uses a wxDataViewTreeStore and forwards most of its API to that class.

Additionally, it uses a wxImageList to store a list of icons.

The main purpose of this class is to provide a simple upgrade path for code using wxTreeCtrl.

Styles

This class supports the following styles:

See wxDataViewCtrl for the list of supported styles.

Events emitted by this class

Event macros for events emitted by this class:

See wxDataViewCtrl for the list of events emitted by this class.

Library: wxAdvanced

Category: Controls, wxDataViewCtrl Related Classes

Since

2.9.0

Public Member Functions

- wxDataViewTreeCtrl ()

  Default ctor.
- wxDataViewTreeCtrl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDV_NO_HEADER|wxDV_ROW_LINES, const wxValidator &validator=wxDefaultValidator)

  Constructor.
- virtual ~wxDataViewTreeCtrl ()

  Destructor.
- wxDataViewItem AppendContainer (const wxDataViewItem &parent, const wxString &text, int icon=-1, int expanded=-1, wxClientData *data=NULL)

  Appends a container to the given parent.
- wxDataViewItem AppendItem (const wxDataViewItem &parent, const wxString &text, int icon=-1, wxClientData *data=NULL)

  Appends an item to the given parent.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDV_NO_HEADER|wxDV_ROW_LINES, const wxValidator &validator=wxDefaultValidator)

  Creates the control and a wxDataViewTreeStore as its internal model.
- void DeleteAllItems ()

  Calls the identical method from wxDataViewTreeStore.
- void DeleteChildren (const wxDataViewItem &item)

  Calls the identical method from wxDataViewTreeStore.
- void DeleteItem (const wxDataViewItem &item)

  Calls the identical method from wxDataViewTreeStore.
- int GetChildCount (const wxDataViewItem &parent) const

  Calls the identical method from wxDataViewTreeStore.
- wxImageList * GetImageList ()

  Returns the image list.
- wxClientData * GetItemData (const wxDataViewItem &item) const

  Calls the identical method from wxDataViewTreeStore.

- const wxIcon & GetItemExpandedIcon (const wxDataViewItem &item) const

  Calls the identical method from wxDataViewTreeStore.
- const wxIcon & GetItemIcon (const wxDataViewItem &item) const

  Calls the identical method from wxDataViewTreeStore.
- wxString GetItemText (const wxDataViewItem &item) const

  Calls the identical method from wxDataViewTreeStore.
- wxDataViewItem GetNthChild (const wxDataViewItem &parent, unsigned int pos) const

  Calls the identical method from wxDataViewTreeStore.
- wxDataViewItem InsertContainer (const wxDataViewItem &parent, const wxDataViewItem &previous, const wxString &text, int icon=-1, int expanded=-1, wxClientData *data=NULL)

  Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.
- wxDataViewItem InsertItem (const wxDataViewItem &parent, const wxDataViewItem &previous, const wxString &text, int icon=-1, wxClientData *data=NULL)

  Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.
- bool IsContainer (const wxDataViewItem &item)

  Returns true if item is a container.
- wxDataViewItem PrependContainer (const wxDataViewItem &parent, const wxString &text, int icon=-1, int expanded=-1, wxClientData *data=NULL)

  Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.
- wxDataViewItem PrependItem (const wxDataViewItem &parent, const wxString &text, int icon=-1, wxClientData *data=NULL)

  Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.
- void SetImageList (wxImageList *imagelist)

  Sets the image list.
- void SetItemData (const wxDataViewItem &item, wxClientData *data)

  Calls the identical method from wxDataViewTreeStore.
- void SetItemExpandedIcon (const wxDataViewItem &item, const wxIcon &icon)

  Calls the identical method from wxDataViewTreeStore.
- void SetItemIcon (const wxDataViewItem &item, const wxIcon &icon)

  Calls the identical method from wxDataViewTreeStore.
- void SetItemText (const wxDataViewItem &item, const wxString &text)

  Calls the identical method from wxDataViewTreeStore.

- wxDataViewTreeStore * GetStore ()

  Returns the store.
- const wxDataViewTreeStore * GetStore () const

  Returns the store.

Additional Inherited Members

21.164.2 Constructor & Destructor Documentation

wxDataViewTreeCtrl::wxDataViewTreeCtrl ( )

Default ctor.

wxDataViewTreeCtrl::wxDataViewTreeCtrl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDV_NO_HEADER|wxDV_ROW_LINES, const wxValidator & *validator* = wxDefaultValidator)

Constructor.

Calls Create().

virtual wxDataViewTreeCtrl::~wxDataViewTreeCtrl ( ) [virtual]

Destructor.

Deletes the image list if any.

21.164.3 Member Function Documentation

wxDataViewItem wxDataViewTreeCtrl::AppendContainer (const **wxDataViewItem** & *parent*, const **wxString** & *text*, int *icon* = −1, int *expanded* = −1, **wxClientData** ∗ *data* = NULL)

Appends a container to the given *parent*.

wxDataViewItem wxDataViewTreeCtrl::AppendItem (const **wxDataViewItem** & *parent*, const **wxString** & *text*, int *icon* = −1, **wxClientData** ∗ *data* = NULL)

Appends an item to the given *parent*.

bool wxDataViewTreeCtrl::Create (**wxWindow** ∗ *parent*, **wxWindowID** *id*, const **wxPoint** & *pos* = **wxDefaultPosition**, const **wxSize** & *size* = **wxDefaultSize**, long *style* = **wxDV_NO_HEADER**|**wxDV_ROW_LINES**, const **wxValidator** & *validator* = **wxDefaultValidator**)

Creates the control and a wxDataViewTreeStore as its internal model.

The default tree column created by this method is an editable column using wxDataViewIconTextRenderer as its renderer.

void wxDataViewTreeCtrl::DeleteAllItems ( )

Calls the identical method from wxDataViewTreeStore.

void wxDataViewTreeCtrl::DeleteChildren (const **wxDataViewItem** & *item*)

Calls the identical method from wxDataViewTreeStore.

void wxDataViewTreeCtrl::DeleteItem (const **wxDataViewItem** & *item*)

Calls the identical method from wxDataViewTreeStore.

int wxDataViewTreeCtrl::GetChildCount (const **wxDataViewItem** & *parent*) const

Calls the identical method from wxDataViewTreeStore.

wxImageList∗ wxDataViewTreeCtrl::GetImageList ( )

Returns the image list.

wxClientData∗ wxDataViewTreeCtrl::GetItemData (const **wxDataViewItem** & *item*) const

Calls the identical method from wxDataViewTreeStore.

const wxIcon& wxDataViewTreeCtrl::GetItemExpandedIcon (const wxDataViewItem & *item*) const

Calls the identical method from wxDataViewTreeStore.

const wxIcon& wxDataViewTreeCtrl::GetItemIcon (const wxDataViewItem & *item*) const

Calls the identical method from wxDataViewTreeStore.

wxString wxDataViewTreeCtrl::GetItemText (const wxDataViewItem & *item*) const

Calls the identical method from wxDataViewTreeStore.

wxDataViewItem wxDataViewTreeCtrl::GetNthChild (const wxDataViewItem & *parent,* unsigned int *pos*) const

Calls the identical method from wxDataViewTreeStore.

wxDataViewTreeStore* wxDataViewTreeCtrl::GetStore ( )

Returns the store.

const wxDataViewTreeStore* wxDataViewTreeCtrl::GetStore ( ) const

Returns the store.

wxDataViewItem wxDataViewTreeCtrl::InsertContainer (const wxDataViewItem & *parent,* const wxDataViewItem & *previous,* const wxString & *text,* int *icon* = -1, int *expanded* = -1, wxClientData * *data* = NULL)

Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.

wxDataViewItem wxDataViewTreeCtrl::InsertItem (const wxDataViewItem & *parent,* const wxDataViewItem & *previous,* const wxString & *text,* int *icon* = -1, wxClientData * *data* = NULL)

Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.

bool wxDataViewTreeCtrl::IsContainer (const wxDataViewItem & *item*)

Returns true if item is a container.

wxDataViewItem wxDataViewTreeCtrl::PrependContainer (const wxDataViewItem & *parent,* const wxString & *text,* int *icon* = -1, int *expanded* = -1, wxClientData * *data* = NULL)

Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.

wxDataViewItem wxDataViewTreeCtrl::PrependItem (const wxDataViewItem & *parent,* const wxString & *text,* int *icon* = -1, wxClientData * *data* = NULL)

Calls the same method from wxDataViewTreeStore but uses an index position in the image list instead of a wxIcon.

void wxDataViewTreeCtrl::SetImageList (wxImageList * *imagelist*)

Sets the image list.

void wxDataViewTreeCtrl::SetItemData (const wxDataViewItem & *item,* wxClientData * *data*)

Calls the identical method from wxDataViewTreeStore.

void wxDataViewTreeCtrl::SetItemExpandedIcon (const wxDataViewItem & *item,* const wxIcon & *icon*)

Calls the identical method from wxDataViewTreeStore.

void wxDataViewTreeCtrl::SetItemIcon (const wxDataViewItem & *item,* const wxIcon & *icon*)

Calls the identical method from wxDataViewTreeStore.

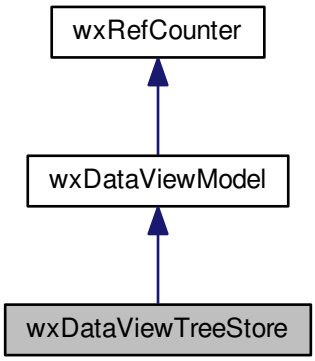
void wxDataViewTreeCtrl::SetItemText (const wxDataViewItem & *item,* const wxString & *text*)

Calls the identical method from wxDataViewTreeStore.

21.165 wxDataViewTreeStore Class Reference

`#include <wx/dataview.h>`

Inheritance diagram for wxDataViewTreeStore:



21.165.1 Detailed Description

wxDataViewTreeStore is a specialised wxDataViewModel for storing simple trees very much like wxTreeCtrl does and it offers a similar API.

This class actually stores the entire tree and the values (therefore its name) and implements all virtual methods from the base class so it can be used directly without having to derive any class from it, but it is mostly used from within wxDataViewTreeCtrl.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewTreeStore ()

  Constructor.
- virtual ~wxDataViewTreeStore ()

  Destructor.
- wxDataViewItem AppendContainer (const wxDataViewItem &parent, const wxString &text, const wxIcon &icon=wxNullIcon, const wxIcon &expanded=wxNullIcon, wxClientData *data=NULL)

  Append a container.
- wxDataViewItem AppendItem (const wxDataViewItem &parent, const wxString &text, const wxIcon &icon=wxNullIcon, wxClientData *data=NULL)

  Append an item.
- void DeleteAllItems ()

  Delete all item in the model.
- void DeleteChildren (const wxDataViewItem &item)

  Delete all children of the item, but not the item itself.
- void DeleteItem (const wxDataViewItem &item)

  Delete this item.
- int GetChildCount (const wxDataViewItem &parent) const

  Return the number of children of item.
- wxClientData * GetItemData (const wxDataViewItem &item) const

  Returns the client data associated with the item.
- const wxIcon & GetItemExpandedIcon (const wxDataViewItem &item) const

  Returns the icon to display in expanded containers.
- const wxIcon & GetItemIcon (const wxDataViewItem &item) const

  Returns the icon of the item.
- wxString GetItemText (const wxDataViewItem &item) const

  Returns the text of the item.
- wxDataViewItem GetNthChild (const wxDataViewItem &parent, unsigned int pos) const

  Returns the nth child item of item.
- wxDataViewItem InsertContainer (const wxDataViewItem &parent, const wxDataViewItem &previous, const wxString &text, const wxIcon &icon=wxNullIcon, const wxIcon &expanded=wxNullIcon, wxClientData *data=NULL)

  Inserts a container after previous.
- wxDataViewItem InsertItem (const wxDataViewItem &parent, const wxDataViewItem &previous, const wxString &text, const wxIcon &icon=wxNullIcon, wxClientData *data=NULL)

  Inserts an item after previous.
- wxDataViewItem PrependContainer (const wxDataViewItem &parent, const wxString &text, const wxIcon &icon=wxNullIcon, const wxIcon &expanded=wxNullIcon, wxClientData *data=NULL)

  Inserts a container before the first child item or parent.
- wxDataViewItem PrependItem (const wxDataViewItem &parent, const wxString &text, const wxIcon &icon=wxNullIcon, wxClientData *data=NULL)

  Inserts an item before the first child item or parent.
- void SetItemData (const wxDataViewItem &item, wxClientData *data)

  Sets the client data associated with the item.
- void SetItemExpandedIcon (const wxDataViewItem &item, const wxIcon &icon)

  Sets the expanded icon for the item.
- void SetItemIcon (const wxDataViewItem &item, const wxIcon &icon)

  Sets the icon for the item.

Additional Inherited Members

21.165.2 Constructor & Destructor Documentation

wxDataViewTreeStore::wxDataViewTreeStore ( )

Constructor.

Creates the invisible root node internally.

virtual wxDataViewTreeStore::~wxDataViewTreeStore ( ) [virtual]

Destructor.

21.165.3 Member Function Documentation

wxDataViewItem wxDataViewTreeStore::AppendContainer (const wxDataViewItem & *parent,* const wxString & *text,* const wxIcon & *icon* = wxNullIcon, const wxIcon & *expanded* = wxNullIcon, wxClientData * *data* = NULL)

Append a container.

wxDataViewItem wxDataViewTreeStore::AppendItem (const wxDataViewItem & *parent,* const wxString & *text,* const wxIcon & *icon* = wxNullIcon, wxClientData * *data* = NULL)

Append an item.

void wxDataViewTreeStore::DeleteAllItems ( )

Delete all item in the model.

void wxDataViewTreeStore::DeleteChildren (const wxDataViewItem & *item*)

Delete all children of the item, but not the item itself.

void wxDataViewTreeStore::DeleteItem (const wxDataViewItem & *item*)

Delete this item.

int wxDataViewTreeStore::GetChildCount (const wxDataViewItem & *parent*) const

Return the number of children of item.

wxClientData* wxDataViewTreeStore::GetItemData (const wxDataViewItem & *item*) const

Returns the client data associated with the item.

const wxIcon& wxDataViewTreeStore::GetItemExpandedIcon (const wxDataViewItem & *item*) const

Returns the icon to display in expanded containers.

const wxIcon& wxDataViewTreeStore::GetItemIcon (const wxDataViewItem & *item*) const

Returns the icon of the item.

wxString wxDataViewTreeStore::GetItemText (const wxDataViewItem & *item*) const

Returns the text of the item.

wxDataViewItem wxDataViewTreeStore::GetNthChild (const wxDataViewItem & *parent,* unsigned int *pos*) const

Returns the nth child item of item.

wxDataViewItem wxDataViewTreeStore::InsertContainer (const wxDataViewItem & *parent,* const wxDataViewItem & *previous,* const wxString & *text,* const wxIcon & *icon* = wxNullIcon, const wxIcon & *expanded* = wxNullIcon, wxClientData * *data* = NULL)

Inserts a container after *previous*.

wxDataViewItem wxDataViewTreeStore::InsertItem (const wxDataViewItem & *parent,* const wxDataViewItem & *previous,* const wxString & *text,* const wxIcon & *icon* = wxNullIcon, wxClientData * *data* = NULL)

Inserts an item after *previous*.

wxDataViewItem wxDataViewTreeStore::PrependContainer (const wxDataViewItem & *parent,* const wxString & *text,* const wxIcon & *icon* = wxNullIcon, const wxIcon & *expanded* = wxNullIcon, wxClientData * *data* = NULL)

Inserts a container before the first child item or *parent*.

wxDataViewItem wxDataViewTreeStore::PrependItem (const wxDataViewItem & *parent,* const wxString & *text,* const wxIcon & *icon* = wxNullIcon, wxClientData * *data* = NULL)

Inserts an item before the first child item or *parent*.

void wxDataViewTreeStore::SetItemData (const wxDataViewItem & *item,* wxClientData * *data*)

Sets the client data associated with the item.

void wxDataViewTreeStore::SetItemExpandedIcon (const wxDataViewItem & *item,* const wxIcon & *icon*)

Sets the expanded icon for the item.

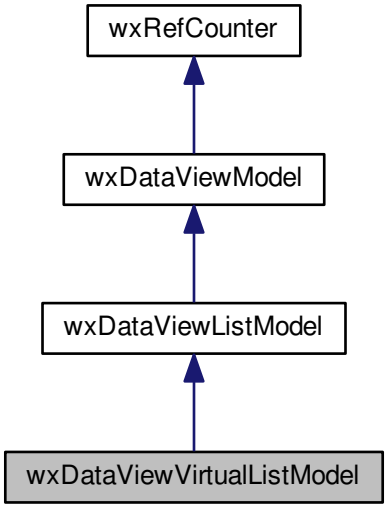
void wxDataViewTreeStore::SetItemIcon (const wxDataViewItem & *item,* const wxIcon & *icon*)

Sets the icon for the item.

21.166 wxDataViewVirtualListModel Class Reference

```
#include <wx/dataview.h>
```

Inheritance diagram for wxDataViewVirtualListModel:



21.166.1 Detailed Description

wxDataViewVirtualListModel is a specialized data model which lets you address an item by its position (row) rather than its wxDataViewItem and as such offers the exact same interface as wxDataViewIndexListModel.

The important difference is that under platforms other than OS X, using this model will result in a truly virtual control able to handle millions of items as the control doesn't store any item (a feature not supported by OS X).

See also

wxDataViewListModel for the API.

Library: wxAdvanced

Category: wxDataViewCtrl Related Classes

Public Member Functions

- wxDataViewVirtualListModel (unsigned int initial_size=0)

  Constructor.
- wxDataViewItem GetItem (unsigned int row) const

  Returns the wxDataViewItem at the given row.
- void Reset (unsigned int new_size)

  Call this after if the data has to be read again from the model.
- void RowAppended ()

  Call this after a row has been appended to the model.
- void RowChanged (unsigned int row)

Call this after a row has been changed.

- void RowDeleted (unsigned int row)

  Call this after a row has been deleted.

- void RowInserted (unsigned int before)

  Call this after a row has been inserted at the given position.

- void RowPrepended ()

  Call this after a row has been prepended to the model.

- void RowValueChanged (unsigned int row, unsigned int col)

  Call this after a value has been changed.

- void RowsDeleted (const wxArrayInt &rows)

  Call this after rows have been deleted.

Additional Inherited Members

21.166.2 Constructor & Destructor Documentation

wxDataViewVirtualListModel::wxDataViewVirtualListModel (unsigned int *initial_size* = 0)

Constructor.

21.166.3 Member Function Documentation

wxDataViewItem wxDataViewVirtualListModel::GetItem (unsigned int *row*) const

Returns the wxDataViewItem at the given *row*.

void wxDataViewVirtualListModel::Reset (unsigned int *new_size*)

Call this after if the data has to be read again from the model.

This is useful after major changes when calling the methods below (possibly thousands of times) doesn't make sense.

void wxDataViewVirtualListModel::RowAppended ( )

Call this after a row has been appended to the model.

void wxDataViewVirtualListModel::RowChanged (unsigned int *row*)

Call this after a row has been changed.

void wxDataViewVirtualListModel::RowDeleted (unsigned int *row*)

Call this after a row has been deleted.

void wxDataViewVirtualListModel::RowInserted (unsigned int *before*)

Call this after a row has been inserted at the given position.

void wxDataViewVirtualListModel::RowPrepended ( )

Call this after a row has been prepended to the model.

void wxDataViewVirtualListModel::RowsDeleted (const wxArrayInt & *rows*)

Call this after rows have been deleted.

The array will internally get copied and sorted in descending order so that the rows with the highest position will be deleted first.

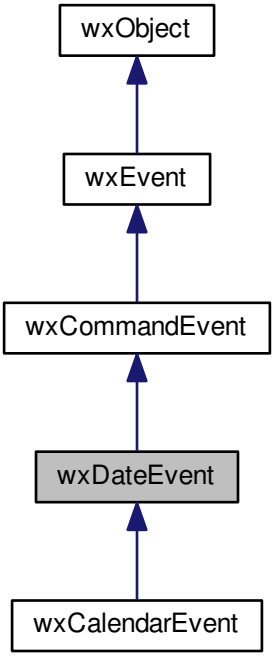
void wxDataViewVirtualListModel::RowValueChanged (unsigned int *row*, unsigned int *col*)

Call this after a value has been changed.

21.167 wxDateEvent Class Reference

`#include <wx/dateevt.h>`

Inheritance diagram for wxDateEvent:



21.167.1 Detailed Description

This event class holds information about a date change and is used together with wxDatePickerCtrl.

It also serves as a base class for wxCalendarEvent.

Library: wxAdvanced

Category: Events

Public Member Functions

- wxDateEvent ()
- wxDateEvent (wxWindow *win, const wxDateTime &dt, wxEventType type)
- const wxDateTime & GetDate () const

  Returns the date.
- void SetDate (const wxDateTime &date)

  Sets the date carried by the event, normally only used by the library internally.

Additional Inherited Members

21.167.2 Constructor & Destructor Documentation

wxDateEvent::wxDateEvent ( )

wxDateEvent::wxDateEvent (wxWindow * *win,* const wxDateTime & *dt,* wxEventType *type*)

21.167.3 Member Function Documentation

const wxDateTime& wxDateEvent::GetDate ( ) const

Returns the date.

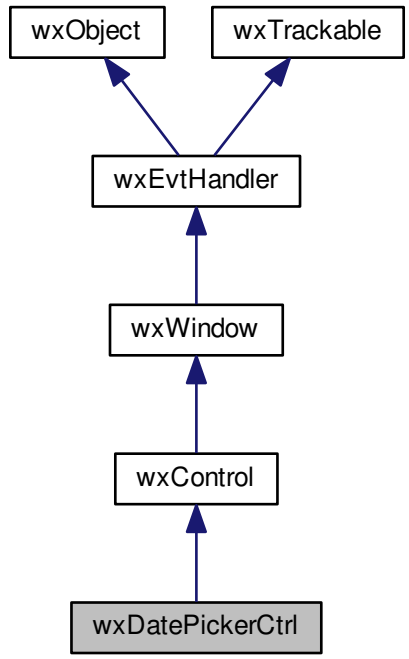
void wxDateEvent::SetDate (const wxDateTime & *date*)

Sets the date carried by the event, normally only used by the library internally.

21.168 wxDatePickerCtrl Class Reference

```
#include <wx/datectrl.h>
```

Inheritance diagram for wxDatePickerCtrl:



21.168.1 Detailed Description

This control allows the user to select a date.

Unlike wxCalendarCtrl, which is a relatively big control, wxDatePickerCtrl is implemented as a small window showing the currently selected date. The control can be edited using the keyboard, and can also display a popup window for more user-friendly date selection, depending on the styles used and the platform.

It is only available if `wxUSE_DATEPICKCTRL` is set to 1.

Styles

This class supports the following styles:

- wxDP_SPIN: Creates a control without a month calendar drop down but with spin-control-like arrows to change individual date components. This style is not supported by the generic version.
- wxDP_DROPDOWN: Creates a control with a month calendar drop-down part from which the user can select a date. This style is not supported in OSX/Cocoa native version.
- wxDP_DEFAULT: Creates a control with the style that is best supported for the current platform (currently wxDP_SPIN under Windows and OSX/Cocoa and wxDP_DROPDOWN elsewhere).
- wxDP_ALLOWNONE: With this style, the control allows the user to not enter any valid date at all. Without it - the default - the control always has some valid date. This style is not supported in OSX/Cocoa native version.
- wxDP_SHOWCENTURY: Forces display of the century in the default date format. Without this style the century could be displayed, or not, depending on the default date representation in the system. This style is not supported in OSX/Cocoa native version currently.

As can be seen from the remarks above, most of the control style are only supported in the native MSW implementation. In portable code it's recommended to use `wxDP_DEFAULT` style only, possibly combined with `wxDP_SHOWCENTURY` (this is also the style used by default if none is specified).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxDateEvent& event)

Event macros for events emitted by this class:

- EVT_DATE_CHANGED(id, func): This event fires when the user changes the current selection in the control.

Library: wxAdvanced

Category: Picker Controls

See also

wxCalendarCtrl, wxDateEvent

Public Member Functions

- wxDatePickerCtrl ()

  Default constructor.
- wxDatePickerCtrl (wxWindow *parent, wxWindowID id, const wxDateTime &dt=wxDefaultDateTime, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDP_DEFAULT|wxDP_SHOWCENTURY, const wxValidator &validator=wxDefaultValidator, const wxString &name="datectrl")

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxDateTime &dt=wxDefaultDateTime, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDP_DEFAULT|wxDP_SHOWCENTURY, const wxValidator &validator=wxDefaultValidator, const wxString &name="datectrl")

  Create the control window.
- virtual bool GetRange (wxDateTime *dt1, wxDateTime *dt2) const

  If the control had been previously limited to a range of dates using SetRange(), returns the lower and upper bounds of this range.
- virtual wxDateTime GetValue () const

  Returns the currently entered date.
- virtual void SetRange (const wxDateTime &dt1, const wxDateTime &dt2)

  Sets the valid range for the date selection.
- virtual void SetValue (const wxDateTime &dt)

  Changes the current value of the control.

Additional Inherited Members

21.168.2 Constructor & Destructor Documentation

wxDatePickerCtrl::wxDatePickerCtrl ( )

Default constructor.

wxDatePickerCtrl::wxDatePickerCtrl (wxWindow * *parent*, wxWindowID *id*, const wxDateTime & *dt* = wxDefaultDateTime, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDP_DEFAULT|wxDP_SHOWCENTURY, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "datectrl")

Initializes the object and calls Create() with all the parameters.

21.168.3 Member Function Documentation

bool wxDatePickerCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxDateTime & *dt* = wxDefaultDateTime, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDP_DEFAULT|wxDP_SHOWCENTURY, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "datectrl")

Create the control window.

This method should only be used for objects created using default constructor.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
dt	The initial value of the control, if an invalid date (such as the default value) is used, the control is set to today.
pos	Initial position.
size	Initial size. If left at default value, the control chooses its own best size by using the height approximately equal to a text control and width large enough to show the date string fully.
style	The window style, see the description of the styles in the class documentation.
validator	Validator which can be used for additional date checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

virtual bool wxDatePickerCtrl::GetRange (wxDateTime * *dt1*, wxDateTime * *dt2*) const `[virtual]`

If the control had been previously limited to a range of dates using SetRange(), returns the lower and upper bounds of this range.

If no range is set (or only one of the bounds is set), *dt1* and/or *dt2* are set to be invalid.

Notice that when using a native MSW implementation of this control the lower range is always set, even if SetRange() hadn't been called explicitly, as the native control only supports dates later than year 1601.

Parameters

dt1	Pointer to the object which receives the lower range limit or becomes invalid if it is not set. May be NULL if the caller is not interested in lower limit.
dt2	Same as above but for the upper limit.

Returns

false if no range limits are currently set, true if at least one bound is set.

virtual wxDateTime wxDatePickerCtrl::GetValue ( ) const `[virtual]`

Returns the currently entered date.

For a control with wxDP_ALLOWNONE style the returned value may be invalid if no date is entered, otherwise it is always valid.

virtual void wxDatePickerCtrl::SetRange (const wxDateTime & *dt1*, const wxDateTime & *dt2*) [virtual]

Sets the valid range for the date selection.

If *dt1* is valid, it becomes the earliest date (inclusive) accepted by the control. If *dt2* is valid, it becomes the latest possible date.

Remarks

If the current value of the control is outside of the newly set range bounds, the behaviour is undefined.

virtual void wxDatePickerCtrl::SetValue (const wxDateTime & *dt*) [virtual]

Changes the current value of the control.

The date should be valid unless the control was created with wxDP_ALLOWNONE style and included in the currently selected range, if any.

Calling this method does not result in a date change event.

21.169 wxDateSpan Class Reference

`#include <wx/datetime.h>`

21.169.1 Detailed Description

This class is a "logical time span" and is useful for implementing program logic for such things as "add one month to the date" which, in general, doesn't mean to add 60∗60∗24∗31 seconds to it, but to take the same date the next month (to understand that this is indeed different consider adding one month to Feb, 15 – we want to get Mar, 15, of course).

When adding a month to the date, all lesser components (days, hours, ...) won't be changed unless the resulting date would be invalid: for example, Jan 31 + 1 month will be Feb 28, not (non-existing) Feb 31.

Because of this feature, adding and subtracting back again the same wxDateSpan will **not**, in general, give back the original date: Feb 28 - 1 month will be Jan 28, not Jan 31!

wxDateSpan objects can be either positive or negative. They may be multiplied by scalars which multiply all deltas by the scalar: i.e. 2∗(1 month and 1 day) is 2 months and 2 days. They can be added together with wxDateTime or wxTimeSpan, but the type of result is different for each case.

Warning

If you specify both weeks and days, the total number of days added will be 7∗weeks + days! See also GetTotalDays().

Equality operators are defined for wxDateSpans. Two wxDateSpans are equal if and only if they both give the same target date when added to **every** source date. Thus wxDateSpan::Months(1) is not equal to wxDateSpan::Days(30), because they don't give the same date when added to Feb 1st. But wxDateSpan::Days(14) is equal to wxDateSpan::Weeks(2).

Finally, notice that for adding hours, minutes and so on you don't need this class at all: wxTimeSpan will do the job because there are no subtleties associated with those (we don't support leap seconds).

Library: wxBase

Category: Data Structures

See also

Date and Time, wxDateTime

Public Member Functions

- wxDateSpan (int years=0, int months=0, int weeks=0, int days=0)

  Constructs the date span object for the given number of years, months, weeks and days.
- wxDateSpan Add (const wxDateSpan &other) const

  Returns the sum of two date spans.
- wxDateSpan & Add (const wxDateSpan &other)

  Adds the given wxDateSpan to this wxDateSpan and returns a reference to itself.
- int GetDays () const

  Returns the number of days (not counting the weeks component) in this date span.
- int GetMonths () const

  Returns the number of the months (not counting the years) in this date span.
- int GetTotalMonths () const

  Returns the combined number of months in this date span, counting both years and months.
- int GetTotalDays () const

  Returns the combined number of days in this date span, counting both weeks and days.
- int GetWeeks () const

  Returns the number of weeks in this date span.
- int GetYears () const

  Returns the number of years in this date span.
- wxDateSpan Multiply (int factor) const

  Returns the product of the date span by the specified factor.
- wxDateSpan & Multiply (int factor)

  Multiplies this date span by the specified factor.
- wxDateSpan & Neg ()

  Changes the sign of this date span.
- wxDateSpan Negate () const

  Returns a date span with the opposite sign.
- wxDateSpan & SetDays (int n)

  Sets the number of days (without modifying any other components) in this date span.
- wxDateSpan & SetMonths (int n)

  Sets the number of months (without modifying any other components) in this date span.
- wxDateSpan & SetWeeks (int n)

  Sets the number of weeks (without modifying any other components) in this date span.
- wxDateSpan & SetYears (int n)

  Sets the number of years (without modifying any other components) in this date span.
- wxDateSpan Subtract (const wxDateSpan &other) const

  Returns the difference of two date spans.
- wxDateSpan & Subtract (const wxDateSpan &other)

  Subtracts the given wxDateSpan to this wxDateSpan and returns a reference to itself.
- wxDateSpan & operator+= (const wxDateSpan &other)

  Adds the given wxDateSpan to this wxDateSpan and returns the result.

- wxDateSpan & operator-= (const wxDateSpan &other)

  Subtracts the given wxDateSpan to this wxDateSpan and returns the result.
- wxDateSpan & operator- ()

  Changes the sign of this date span.
- wxDateSpan & operator*= (int factor)

  Multiplies this date span by the specified factor.
- bool operator!= (const wxDateSpan &other) const

  Returns true if this date span is different from the other one.
- bool operator== (const wxDateSpan &other) const

  Returns true if this date span is equal to the other one.

Static Public Member Functions

- static wxDateSpan Day ()

  Returns a date span object corresponding to one day.
- static wxDateSpan Days (int days)

  Returns a date span object corresponding to the given number of days.
- static wxDateSpan Month ()

  Returns a date span object corresponding to one month.
- static wxDateSpan Months (int mon)

  Returns a date span object corresponding to the given number of months.
- static wxDateSpan Week ()

  Returns a date span object corresponding to one week.
- static wxDateSpan Weeks (int weeks)

  Returns a date span object corresponding to the given number of weeks.
- static wxDateSpan Year ()

  Returns a date span object corresponding to one year.
- static wxDateSpan Years (int years)

  Returns a date span object corresponding to the given number of years.

21.169.2 Constructor & Destructor Documentation

wxDateSpan::wxDateSpan (int *years* = 0, int *months* = 0, int *weeks* = 0, int *days* = 0)

Constructs the date span object for the given number of years, months, weeks and days.

Note that the weeks and days add together if both are given.

21.169.3 Member Function Documentation

wxDateSpan wxDateSpan::Add (const wxDateSpan & *other*) const

Returns the sum of two date spans.

Returns

A new wxDateSpan object with the result.

wxDateSpan& wxDateSpan::Add (const wxDateSpan & *other*)

Adds the given wxDateSpan to this wxDateSpan and returns a reference to itself.

static wxDateSpan wxDateSpan::Day ( ) [static]

Returns a date span object corresponding to one day.

See also

Days()

static wxDateSpan wxDateSpan::Days (int *days*) [static]

Returns a date span object corresponding to the given number of days.

See also

Day()

int wxDateSpan::GetDays ( ) const

Returns the number of days (not counting the weeks component) in this date span.

See also

GetTotalDays()

int wxDateSpan::GetMonths ( ) const

Returns the number of the months (not counting the years) in this date span.

int wxDateSpan::GetTotalDays ( ) const

Returns the combined number of days in this date span, counting both weeks and days.

This doesn't take months or years into account.

See also

GetWeeks(), GetDays()

int wxDateSpan::GetTotalMonths ( ) const

Returns the combined number of months in this date span, counting both years and months.

See also

GetYears(), GetMonths()

Since

2.9.5

int wxDateSpan::GetWeeks ( ) const

Returns the number of weeks in this date span.

See also

GetTotalDays()

int wxDateSpan::GetYears ( ) const

Returns the number of years in this date span.

static wxDateSpan wxDateSpan::Month ( ) [static]

Returns a date span object corresponding to one month.

See also

Months()

static wxDateSpan wxDateSpan::Months (int *mon*) [static]

Returns a date span object corresponding to the given number of months.

See also

Month()

wxDateSpan wxDateSpan::Multiply (int *factor*) const

Returns the product of the date span by the specified *factor*.

The product is computed by multiplying each of the components by the *factor*.

Returns

A new wxDateSpan object with the result.

wxDateSpan& wxDateSpan::Multiply (int *factor*)

Multiplies this date span by the specified *factor*.

The product is computed by multiplying each of the components by the *factor*.

Returns

A reference to this wxDateSpan object modified in place.

wxDateSpan& wxDateSpan::Neg ( )

Changes the sign of this date span.

See also

Negate()

wxDateSpan wxDateSpan::Negate ( ) const

Returns a date span with the opposite sign.

See also

Neg()

bool wxDateSpan::operator!= (const wxDateSpan & *other*) const

Returns true if this date span is different from the other one.

wxDateSpan& wxDateSpan::operator*= (int *factor*)

Multiplies this date span by the specified *factor*.

The product is computed by multiplying each of the components by the *factor*.

Returns

A reference to this wxDateSpan object modified in place.

wxDateSpan& wxDateSpan::operator+= (const wxDateSpan & *other*)

Adds the given wxDateSpan to this wxDateSpan and returns the result.

wxDateSpan& wxDateSpan::operator- ( )

Changes the sign of this date span.

See also

Negate()

wxDateSpan& wxDateSpan::operator-= (const wxDateSpan & *other*)

Subtracts the given wxDateSpan to this wxDateSpan and returns the result.

bool wxDateSpan::operator== (const wxDateSpan & *other*) const

Returns true if this date span is equal to the other one.

Two date spans are considered equal if and only if they have the same number of years and months and the same total number of days (counting both days and weeks).

wxDateSpan& wxDateSpan::SetDays (int *n*)

Sets the number of days (without modifying any other components) in this date span.

wxDateSpan& wxDateSpan::SetMonths (int *n*)

Sets the number of months (without modifying any other components) in this date span.

wxDateSpan& wxDateSpan::SetWeeks (int *n*)

Sets the number of weeks (without modifying any other components) in this date span.

wxDateSpan& wxDateSpan::SetYears (int *n*)

Sets the number of years (without modifying any other components) in this date span.

wxDateSpan wxDateSpan::Subtract (const wxDateSpan & *other*) const

Returns the difference of two date spans.

Returns

A new wxDateSpan object with the result.

wxDateSpan& wxDateSpan::Subtract (const wxDateSpan & *other*)

Subtracts the given wxDateSpan to this wxDateSpan and returns a reference to itself.

static wxDateSpan wxDateSpan::Week ( ) [static]

Returns a date span object corresponding to one week.

See also

Weeks()

static wxDateSpan wxDateSpan::Weeks (int *weeks*) [static]

Returns a date span object corresponding to the given number of weeks.

See also

Week()

static wxDateSpan wxDateSpan::Year ( ) [static]

Returns a date span object corresponding to one year.

See also

Years()

static wxDateSpan wxDateSpan::Years (int *years*) [static]

Returns a date span object corresponding to the given number of years.

See also

Year()

21.170 wxDateTime Class Reference

`#include <wx/datetime.h>`

21.170.1 Detailed Description

wxDateTime class represents an absolute moment in time.

The type `wxDateTime_t` is typedefed as `unsigned short` and is used to contain the number of years, hours, minutes, seconds and milliseconds.

Global constant wxDefaultDateTime and synonym for it wxInvalidDateTime are defined. This constant will be different from any valid wxDateTime object.

21.170.2 Static Functions

All static functions either set or return the static variables of wxDateSpan (the country), return the current moment, year, month or number of days in it, or do some general calendar-related actions.

Please note that although several function accept an extra Calendar parameter, it is currently ignored as only the Gregorian calendar is supported. Future versions will support other calendars.

21.170.3 Date Formatting and Parsing

The date formatting and parsing functions convert wxDateTime objects to and from text. The conversions to text are mostly trivial: you can either do it using the default date and time representations for the current locale (FormatDate() and FormatTime()), using the international standard representation defined by ISO 8601 (FormatISODate(), FormatISOTime() and FormatISOCombined()) or by specifying any format at all and using Format() directly.

The conversions from text are more interesting, as there are much more possibilities to care about. The simplest cases can be taken care of with ParseFormat() which can parse any date in the given (rigid) format. ParseRfc822Date() is another function for parsing dates in predefined format – the one of RFC 822 which (still...) defines the format of email messages on the Internet. This format cannot be described with `strptime(3)-like` format strings used by Format(), hence the need for a separate function.

But the most interesting functions are ParseTime(), ParseDate() and ParseDateTime(). They try to parse the date and time (or only one of them) in 'free' format, i.e. allow them to be specified in any of possible ways. These functions will usually be used to parse the (interactive) user input which is not bound to be in any predefined format. As an example, ParseDate() can parse the strings such as "tomorrow", "March first" and even "next Sunday".

Finally notice that each of the parsing functions is available in several overloads: if the input string is a narrow (`char *`) string, then a narrow pointer is returned. If the input string is a wide string, a wide char pointer is returned. Finally, if the input parameter is a wxString, a narrow char pointer is also returned for backwards compatibility but there is also an additional argument of wxString::const_iterator type in which, if it is not NULL, an iterator pointing to the end of the scanned string part is returned.

Library: wxBase

Category: Data Structures

Predefined objects/pointers:

- wxDefaultDateTime

See also

Date and Time, wxTimeSpan, wxDateSpan, wxCalendarCtrl

Classes

- class TimeZone

  Class representing a time zone.

- struct Tm

  Contains broken down date-time representation.

Public Types

- enum TZ {
  Local,
  GMT_12,
  GMT_11,
  GMT_10,
  GMT_9,
  GMT_8,
  GMT_7,
  GMT_6,
  GMT_5,
  GMT_4,
  GMT_3,
  GMT_2,
  GMT_1,
  GMT0,
  GMT1,
  GMT2,
  GMT3,
  GMT4,
  GMT5,
  GMT6,
  GMT7,
  GMT8,
  GMT9,
  GMT10,
  GMT11,
  GMT12,
  GMT13,
  WET = GMT0,
  WEST = GMT1,
  CET = GMT1,
  CEST = GMT2,
  EET = GMT2,
  EEST = GMT3,
  MSK = GMT3,
  MSD = GMT4,
  AST = GMT_4,
  ADT = GMT_3,
  EST = GMT_5,
  EDT = GMT_4,
  CST = GMT_6,
  CDT = GMT_5,
  MST = GMT_7,
  MDT = GMT_6,
  PST = GMT_8,
  PDT = GMT_7,
  HST = GMT_10,
  AKST = GMT_9,
  AKDT = GMT_8,
  A_WST = GMT8,
  A_CST = GMT13 + 1,
  A_EST = GMT10,
  A_ESST = GMT11,
  NZST = GMT12,
  NZDT = GMT13,
  UTC = GMT0 }

Time zone symbolic names.

- enum Calendar {
  Gregorian,
  Julian }

  Several functions accept an extra parameter specifying the calendar to use (although most of them only support now the Gregorian calendar).
- enum Country {
  Country_Unknown,
  Country_Default,
  Country_WesternEurope_Start,
  Country_EEC = Country_WesternEurope_Start,
  France,
  Germany,
  UK,
  Country_WesternEurope_End = UK,
  Russia,
  USA }

  Date calculations often depend on the country and wxDateTime allows to set the country whose conventions should be used using SetCountry().
- enum Month {
  Jan,
  Feb,
  Mar,
  Apr,
  May,
  Jun,
  Jul,
  Aug,
  Sep,
  Oct,
  Nov,
  Dec,
  Inv_Month }

  symbolic names for the months
- enum WeekDay {
  Sun,
  Mon,
  Tue,
  Wed,
  Thu,
  Fri,
  Sat,
  Inv_WeekDay }

  symbolic names for the weekdays
- enum Year { Inv_Year = SHRT_MIN }

  invalid value for the year
- enum NameFlags {
  Name_Full = 0x01,
  Name_Abbr = 0x02 }

  Flags to be used with GetMonthName() and GetWeekDayName() functions.
- enum WeekFlags {
  Default_First,
  Monday_First,
  Sunday_First }

  Different parts of the world use different conventions for the week start.
- typedef unsigned short wxDateTime_t

  A small unsigned integer type for storing things like minutes, seconds &c.

Public Member Functions

Constructors, Assignment Operators and Setters

Constructors and various Set() methods are collected here.

If you construct a date object from separate values for day, month and year, you should use IsValid() method to check that the values were correct as constructors cannot return an error code.

- wxDateTime ()
  Default constructor.
- wxDateTime (const wxDateTime &date)
  Copy constructor.
- wxDateTime (time_t timet)
  Same as Set().
- wxDateTime (const struct tm &tm)
  Same as Set().
- wxDateTime (double jdn)
  Same as Set().
- wxDateTime (wxDateTime_t hour, wxDateTime_t minute=0, wxDateTime_t second=0, wxDateTime_t millisec=0)
  Same as Set().
- wxDateTime (wxDateTime_t day, Month month, int year=Inv_Year, wxDateTime_t hour=0, wxDateTime_t minute=0, wxDateTime_t second=0, wxDateTime_t millisec=0)
  Same as Set().
- wxDateTime (const struct _SYSTEMTIME &st)
  Same as SetFromMSWSysTime.
- wxDateTime & ResetTime ()
  Reset time to midnight (00:00:00) without changing the date.
- wxDateTime & Set (time_t timet)
  Constructs the object from timet value holding the number of seconds since Jan 1, 1970 UTC.
- wxDateTime & Set (const struct tm &tm)
  Sets the date and time from the broken down representation in the standard tm structure.
- wxDateTime & Set (const Tm &tm)
  Sets the date and time from the broken down representation in the wxDateTime::Tm structure.
- wxDateTime & Set (double jdn)
  Sets the date from the so-called Julian Day Number.
- wxDateTime & Set (wxDateTime_t hour, wxDateTime_t minute=0, wxDateTime_t second=0, wxDateTime_t millisec=0)
  Sets the date to be equal to Today() and the time from supplied parameters.
- wxDateTime & Set (wxDateTime_t day, Month month, int year=Inv_Year, wxDateTime_t hour=0, wxDateTime_t minute=0, wxDateTime_t second=0, wxDateTime_t millisec=0)
  Sets the date and time from the parameters.
- wxDateTime & SetDay (unsigned short day)
  Sets the day without changing other date components.
- wxDateTime & SetFromDOS (unsigned long ddt)
  Sets the date from the date and time in DOS format.
- wxDateTime & SetHour (unsigned short hour)
  Sets the hour without changing other date components.
- wxDateTime & SetMillisecond (unsigned short millisecond)
  Sets the millisecond without changing other date components.
- wxDateTime & SetMinute (unsigned short minute)
  Sets the minute without changing other date components.
- wxDateTime & SetMonth (Month month)
  Sets the month without changing other date components.
- wxDateTime & SetSecond (unsigned short second)
  Sets the second without changing other date components.
- wxDateTime & SetToCurrent ()
  Sets the date and time of to the current values.
- wxDateTime & SetYear (int year)
  Sets the year without changing other date components.

- wxDateTime & operator= (time_t timet)
  Same as Set().
- wxDateTime & operator= (const struct tm &tm)
  Same as Set().

Accessors

Here are the trivial accessors.

Other functions, which might have to perform some more complicated calculations to find the answer are under the "Date Arithmetics" section.

- unsigned long GetAsDOS () const
  Returns the date and time in DOS format.
- wxDateTime & SetFromMSWSysTime (const struct _SYSTEMTIME &st)
  Initialize using the Windows SYSTEMTIME structure.
- void GetAsMSWSysTime (struct _SYSTEMTIME *st) const
  Returns the date and time in the Windows SYSTEMTIME format.
- int GetCentury (const TimeZone &tz=Local) const
  Returns the century of this date.
- wxDateTime GetDateOnly () const
  Returns the object having the same date component as this one but time of 00:00:00.
- unsigned short GetDay (const TimeZone &tz=Local) const
  Returns the day in the given timezone (local one by default).
- unsigned short GetDayOfYear (const TimeZone &tz=Local) const
  Returns the day of the year (in 1-366 range) in the given timezone (local one by default).
- unsigned short GetHour (const TimeZone &tz=Local) const
  Returns the hour in the given timezone (local one by default).
- unsigned short GetMillisecond (const TimeZone &tz=Local) const
  Returns the milliseconds in the given timezone (local one by default).
- unsigned short GetMinute (const TimeZone &tz=Local) const
  Returns the minute in the given timezone (local one by default).
- Month GetMonth (const TimeZone &tz=Local) const
  Returns the month in the given timezone (local one by default).
- unsigned short GetSecond (const TimeZone &tz=Local) const
  Returns the seconds in the given timezone (local one by default).
- time_t GetTicks () const
  Returns the number of seconds since Jan 1, 1970 UTC.
- Tm GetTm (const TimeZone &tz=Local) const
  Returns broken down representation of the date and time.
- WeekDay GetWeekDay (const TimeZone &tz=Local) const
  Returns the week day in the given timezone (local one by default).
- int GetWeekBasedYear (const TimeZone &tz) const
  Returns the year to which the week containing this date belongs.
- wxDateTime_t GetWeekOfMonth (WeekFlags flags=Monday_First, const TimeZone &tz=Local) const
  Returns the ordinal number of the week in the month (in 1-5 range).
- wxDateTime_t GetWeekOfYear (WeekFlags flags=Monday_First, const TimeZone &tz=Local) const
  Returns the number of the week of the year this date is in.
- int GetYear (const TimeZone &tz=Local) const
  Returns the year in the given timezone (local one by default).
- bool IsValid () const
  Returns true if the object represents a valid time moment.
- bool IsWorkDay (Country country=Country_Default) const
  Returns true is this day is not a holiday in the given country.

Date Comparison

There are several functions to allow date comparison.

To supplement them, a few global operators, etc taking wxDateTime are defined.

- bool IsEarlierThan (const wxDateTime &datetime) const

Returns true if this date precedes the given one.

- bool IsEqualTo (const wxDateTime &datetime) const

  Returns true if the two dates are strictly identical.
- bool IsEqualUpTo (const wxDateTime &dt, const wxTimeSpan &ts) const

  Returns true if the date is equal to another one up to the given time interval, i.e. if the absolute difference between the two dates is less than this interval.
- bool IsLaterThan (const wxDateTime &datetime) const

  Returns true if this date is later than the given one.
- bool IsSameDate (const wxDateTime &dt) const

  Returns true if the date is the same without comparing the time parts.
- bool IsSameTime (const wxDateTime &dt) const

  Returns true if the time is the same (although dates may differ).
- bool IsStrictlyBetween (const wxDateTime &t1, const wxDateTime &t2) const

  Returns true if this date lies strictly between the two given dates.
- bool IsBetween (const wxDateTime &t1, const wxDateTime &t2) const

  Returns true if IsStrictlyBetween() is true or if the date is equal to one of the limit values.

Date Arithmetics

These functions carry out arithmetics on the wxDateTime objects.

As explained in the overview, either wxTimeSpan or wxDateSpan may be added to wxDateTime, hence all functions are overloaded to accept both arguments.

Also, both Add() and Subtract() have both const and non-const version. The first one returns a new object which represents the sum/difference of the original one with the argument while the second form modifies the object to which it is applied. The operators "-=" and "+=" are defined to be equivalent to the second forms of these functions.

- wxDateTime Add (const wxDateSpan &diff) const

  Adds the given date span to this object.
- wxDateTime & Add (const wxDateSpan &diff)

  Adds the given date span to this object.
- wxDateTime Add (const wxTimeSpan &diff) const

  Adds the given time span to this object.
- wxDateTime & Add (const wxTimeSpan &diff)

  Adds the given time span to this object.
- wxDateTime Subtract (const wxTimeSpan &diff) const

  Subtracts the given time span from this object.
- wxDateTime & Subtract (const wxTimeSpan &diff)

  Subtracts the given time span from this object.
- wxDateTime Subtract (const wxDateSpan &diff) const

  Subtracts the given date span from this object.
- wxDateTime & Subtract (const wxDateSpan &diff)

  Subtracts the given date span from this object.
- wxTimeSpan Subtract (const wxDateTime &dt) const

  Subtracts another date from this one and returns the difference between them as a wxTimeSpan.
- wxDateSpan DiffAsDateSpan (const wxDateTime &dt) const

  Returns the difference between this object and dt as a wxDateSpan.
- wxDateTime & operator+= (const wxDateSpan &diff)

  Adds the given date span to this object.
- wxDateTime operator+ (const wxDateSpan &ds) const

  Adds the given date span to this object.
- wxDateTime & operator-= (const wxDateSpan &diff)

  Subtracts the given date span from this object.
- wxDateTime operator- (const wxDateSpan &ds) const

  Subtracts the given date span from this object.
- wxDateTime & operator+= (const wxTimeSpan &diff)

  Adds the given time span to this object.
- wxDateTime operator+ (const wxTimeSpan &ts) const

  Adds the given time span to this object.

- wxDateTime & operator-= (const wxTimeSpan &diff)
  Subtracts the given time span from this object.
- wxDateTime operator- (const wxTimeSpan &ts) const
  Subtracts the given time span from this object.
- wxTimeSpan operator- (const wxDateTime &dt2) const
  Subtracts another date from this one and returns the difference between them as a wxTimeSpan.

Date Formatting and Parsing

See Date Formatting and Parsing

- wxString Format (const wxString &format=wxDefaultDateTimeFormat, const TimeZone &tz=Local) const
  This function does the same as the standard ANSI C `strftime(3)` function (http://www.cplusplus.com/reference/clibrary/ctime/strftime.html).
- wxString FormatDate () const
  Identical to calling Format() with `"%x"` argument (which means "preferred date representation for the current locale").
- wxString FormatISOCombined (char sep= 'T') const
  Returns the combined date-time representation in the ISO 8601 format `"YYYY-MM-DDTHH:MM:SS"`.
- wxString FormatISODate () const
  This function returns the date representation in the ISO 8601 format `"YYYY-MM-DD"`.
- wxString FormatISOTime () const
  This function returns the time representation in the ISO 8601 format `"HH:MM:SS"`.
- wxString FormatTime () const
  Identical to calling Format() with `"%X"` argument (which means "preferred time representation for the current locale").
- bool ParseDate (const wxString &date, wxString::const_iterator *end)
  This function is like ParseDateTime(), but it only allows the date to be specified.
- bool ParseDateTime (const wxString &datetime, wxString::const_iterator *end)
  Parses the string datetime containing the date and time in free format.
- bool ParseFormat (const wxString &date, const wxString &format, const wxDateTime &dateDef, wxString::const_iterator *end)
  This function parses the string date according to the given format.
- bool ParseFormat (const wxString &date, const wxString &format, wxString::const_iterator *end)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool ParseFormat (const wxString &date, wxString::const_iterator *end)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool ParseISOCombined (const wxString &date, char sep= 'T')
  This function parses the string containing the date and time in ISO 8601 combined format `"YYYY-MM-DDTHH:MM:SS"`.
- bool ParseISODate (const wxString &date)
  This function parses the date in ISO 8601 format `"YYYY-MM-DD"`.
- bool ParseISOTime (const wxString &date)
  This function parses the time in ISO 8601 format `"HH:MM:SS"`.
- bool ParseRfc822Date (const wxString &date, wxString::const_iterator *end)
  Parses the string date looking for a date formatted according to the RFC 822 in it.
- bool ParseTime (const wxString &time, wxString::const_iterator *end)
  This functions is like ParseDateTime(), but only allows the time to be specified in the input string.

Calendar Calculations

The functions in this section perform the basic calendar calculations, mostly related to the week days.

They allow to find the given week day in the week with given number (either in the month or in the year) and so on.

None of the functions in this section modify the time part of the wxDateTime, they only work with the date part of it.

- wxDateTime GetLastMonthDay (Month month=Inv_Month, int year=Inv_Year) const

Returns the copy of this object to which SetToLastMonthDay() was applied.

- wxDateTime GetLastWeekDay (WeekDay weekday, Month month=Inv_Month, int year=Inv_Year)

  Returns the copy of this object to which SetToLastWeekDay() was applied.
- wxDateTime GetNextWeekDay (WeekDay weekday) const

  Returns the copy of this object to which SetToNextWeekDay() was applied.
- wxDateTime GetPrevWeekDay (WeekDay weekday) const

  Returns the copy of this object to which SetToPrevWeekDay() was applied.
- wxDateTime GetWeekDay (WeekDay weekday, int n=1, Month month=Inv_Month, int year=Inv_Year) const

  Returns the copy of this object to which SetToWeekDay() was applied.
- wxDateTime GetWeekDayInSameWeek (WeekDay weekday, WeekFlags flags=Monday_First) const

  Returns the copy of this object to which SetToWeekDayInSameWeek() was applied.
- wxDateTime GetYearDay (wxDateTime_t yday) const

  Returns the copy of this object to which SetToYearDay() was applied.
- wxDateTime & SetToLastMonthDay (Month month=Inv_Month, int year=Inv_Year)

  Sets the date to the last day in the specified month (the current one by default).
- bool SetToLastWeekDay (WeekDay weekday, Month month=Inv_Month, int year=Inv_Year)

  The effect of calling this function is the same as of calling `SetToWeekDay`(-1, weekday, month, year).
- wxDateTime & SetToNextWeekDay (WeekDay weekday)

  Sets the date so that it will be the first weekday following the current date.
- wxDateTime & SetToPrevWeekDay (WeekDay weekday)

  Sets the date so that it will be the last weekday before the current date.
- bool SetToWeekDay (WeekDay weekday, int n=1, Month month=Inv_Month, int year=Inv_Year)

  Sets the date to the n-th weekday in the given month of the given year (the current month and year are used by default).
- wxDateTime & SetToWeekDayInSameWeek (WeekDay weekday, WeekFlags flags=Monday_First)

  Adjusts the date so that it will still lie in the same week as before, but its week day will be the given one.
- wxDateTime & SetToYearDay (wxDateTime_t yday)

  Sets the date to the day number yday in the same year (i.e. unlike the other functions, this one does not use the current year).

Astronomical/Historical Functions

Some degree of support for the date units used in astronomy and/or history is provided.

You can construct a wxDateTime object from a JDN and you may also get its JDN, MJD or Rata Die number from it.

Related functions in other groups: wxDateTime(double), Set(double)

- double GetJDN () const

  Synonym for GetJulianDayNumber().
- double GetJulianDayNumber () const

  Returns the JDN corresponding to this date.
- double GetMJD () const

  Synonym for GetModifiedJulianDayNumber().
- double GetModifiedJulianDayNumber () const

  Returns the "Modified Julian Day Number" (MJD) which is, by definition, is equal to JDN - 2400000.5.
- double GetRataDie () const

  Return the Rata Die number of this date.

Time Zone and DST Support

Please see the time zone overview for more information about time zones.

Normally, these functions should be rarely used.

Related functions in other groups: GetBeginDST(), GetEndDST()

- wxDateTime FromTimezone (const TimeZone &tz, bool noDST=false) const

  Transform the date from the given time zone to the local one.
- int IsDST (Country country=Country_Default) const

  Returns true if the DST is applied for this date in the given country.
- wxDateTime & MakeFromTimezone (const TimeZone &tz, bool noDST=false)

Same as FromTimezone() but modifies the object in place.

- wxDateTime & MakeTimezone (const TimeZone &tz, bool noDST=false)

  Modifies the object in place to represent the date in another time zone.
- wxDateTime & MakeUTC (bool noDST=false)

  This is the same as calling MakeTimezone() with the argument `GMT0`.
- wxDateTime ToTimezone (const TimeZone &tz, bool noDST=false) const

  Transform the date to the given time zone.
- wxDateTime ToUTC (bool noDST=false) const

  This is the same as calling ToTimezone() with the argument `GMT0`.

Static Public Member Functions

- static int ConvertYearToBC (int year)

  Converts the year in absolute notation (i.e. a number which can be negative, positive or zero) to the year in BC/AD notation.
- static void GetAmPmStrings (wxString *am, wxString *pm)

  Returns the translations of the strings `AM` and `PM` used for time formatting for the current locale.
- static wxDateTime GetBeginDST (int year=Inv_Year, Country country=Country_Default)

  Get the beginning of DST for the given country in the given year (current one by default).
- static wxDateTime GetEndDST (int year=Inv_Year, Country country=Country_Default)

  Returns the end of DST for the given country in the given year (current one by default).
- static int GetCentury (int year)

  Get the current century, i.e. first two digits of the year, in given calendar (only Gregorian is currently supported).
- static Country GetCountry ()

  Returns the current default country.
- static Month GetCurrentMonth (Calendar cal=Gregorian)

  Get the current month in given calendar (only Gregorian is currently supported).
- static int GetCurrentYear (Calendar cal=Gregorian)

  Get the current year in given calendar (only Gregorian is currently supported).
- static wxString GetEnglishMonthName (Month month, NameFlags flags=Name_Full)

  Return the standard English name of the given month.
- static wxString GetEnglishWeekDayName (WeekDay weekday, NameFlags flags=Name_Full)

  Return the standard English name of the given week day.
- static wxString GetMonthName (Month month, NameFlags flags=Name_Full)

  Gets the full (default) or abbreviated name of the given month.
- static wxDateTime_t GetNumberOfDays (int year, Calendar cal=Gregorian)

  Returns the number of days in the given year.
- static wxDateTime_t GetNumberOfDays (Month month, int year=Inv_Year, Calendar cal=Gregorian)

  Returns the number of days in the given month of the given year.
- static time_t GetTimeNow ()

  Returns the current time.
- static tm * GetTmNow (struct tm *tm)

  Returns the current time broken down using the buffer whose address is passed to the function with tm to store the result.
- static tm * GetTmNow ()

  Returns the current time broken down.
- static wxString GetWeekDayName (WeekDay weekday, NameFlags flags=Name_Full)

  Gets the full (default) or abbreviated name of the given week day.
- static bool IsDSTApplicable (int year=Inv_Year, Country country=Country_Default)

  Returns true if DST was used in the given year (the current one by default) in the given country.
- static bool IsLeapYear (int year=Inv_Year, Calendar cal=Gregorian)

  Returns true if the year is a leap one in the specified calendar.

- static bool IsWestEuropeanCountry (Country country=Country_Default)

  This function returns true if the specified (or default) country is one of Western European ones.
- static wxDateTime Now ()

  Returns the object corresponding to the current time.
- static void SetCountry (Country country)

  Sets the country to use by default.
- static wxDateTime SetToWeekOfYear (int year, wxDateTime_t numWeek, WeekDay weekday=Mon)

  Set the date to the given weekday in the week number numWeek of the given year .
- static wxDateTime Today ()

  Returns the object corresponding to the midnight of the current day (i.e. the same as Now(), but the time part is set to 0).
- static wxDateTime UNow ()

  Returns the object corresponding to the current UTC time including the milliseconds.

21.170.4 Member Typedef Documentation

typedef unsigned short wxDateTime::wxDateTime_t

A small unsigned integer type for storing things like minutes, seconds &c.

It should be at least short (i.e. not char) to contain the number of milliseconds - it may also be 'int' because there is no size penalty associated with it in our code, we don't store any data in this format.

21.170.5 Member Enumeration Documentation

enum wxDateTime::Calendar

Several functions accept an extra parameter specifying the calendar to use (although most of them only support now the Gregorian calendar).

This parameters is one of the following values.

Enumerator

Gregorian calendar currently in use in Western countries

Julian calendar in use since -45 until the 1582 (or later)

enum wxDateTime::Country

Date calculations often depend on the country and wxDateTime allows to set the country whose conventions should be used using SetCountry().

It takes one of the following values as parameter.

Enumerator

Country_Unknown no special information for this country

Country_Default set the default country with SetCountry() method or use the default country with any other

Country_WesternEurope_Start

Country_EEC

France

Germany

UK

Country_WesternEurope_End

Russia

USA

enum wxDateTime::Month

symbolic names for the months

Enumerator

Jan

Feb

Mar

Apr

May

Jun

Jul

Aug

Sep

Oct

Nov

Dec

Inv_Month Invalid month value.

enum wxDateTime::NameFlags

Flags to be used with GetMonthName() and GetWeekDayName() functions.

Enumerator

Name_Full return full name

Name_Abbr return abbreviated name

enum wxDateTime::TZ

Time zone symbolic names.

Enumerator

Local the time in the current time zone

GMT_12 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_11 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_10 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_9 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_8 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_7 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_6 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_5 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_4 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_3 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_2 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT_1 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT0 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT1 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT2 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT3 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT4 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT5 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT6 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT7 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT8 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT9 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT10 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT11 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT12 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

GMT13 zones from GMT (= Greenwich Mean Time): they're guaranteed to be consequent numbers, so writing something like 'GMT0 + offset' is safe if abs(offset) <= 12

WET Western Europe Time.

WEST Western Europe Summer Time.

CET Central Europe Time.

CEST Central Europe Summer Time.

EET Eastern Europe Time.

EEST Eastern Europe Summer Time.

MSK Moscow Time.

MSD Moscow Summer Time.

AST Atlantic Standard Time.

ADT Atlantic Daylight Time.

EST Eastern Standard Time.

EDT Eastern Daylight Saving Time.

CST Central Standard Time.

CDT Central Daylight Saving Time.

MST Mountain Standard Time.

MDT Mountain Daylight Saving Time.

PST Pacific Standard Time.

PDT Pacific Daylight Saving Time.

HST Hawaiian Standard Time.

AKST Alaska Standard Time.

AKDT Alaska Daylight Saving Time.

A_WST Western Standard Time.

A_CST Central Standard Time (+9.5)

A_EST Eastern Standard Time.

A_ESST Eastern Summer Time.

NZST Standard Time.

NZDT Daylight Saving Time.

UTC Universal Coordinated Time = the new and politically correct name for GMT.

enum wxDateTime::WeekDay

symbolic names for the weekdays

Enumerator

Sun

Mon

Tue

Wed

Thu

Fri

Sat

Inv_WeekDay Invalid week day value.

enum wxDateTime::WeekFlags

Different parts of the world use different conventions for the week start.

In some countries, the week starts on Sunday, while in others – on Monday. The ISO standard doesn't address this issue, so we support both conventions in the functions whose result depends on it (GetWeekOfYear() and GetWeekOfMonth()).

The desired behaviour may be specified by giving one of the following constants as argument to these functions.

Enumerator

Default_First Sunday_First for US, Monday_First for the rest.

Monday_First week starts with a Monday

Sunday_First week starts with a Sunday

enum wxDateTime::Year

invalid value for the year

Enumerator

Inv_Year

21.170.6 Constructor & Destructor Documentation

wxDateTime::wxDateTime ( )

Default constructor.

Use one of the Set() functions to initialize the object later.

wxDateTime::wxDateTime (const wxDateTime & *date*)

Copy constructor.

wxDateTime::wxDateTime (time_t *timet*)

Same as Set().

wxDateTime::wxDateTime (const struct tm & *tm*)

Same as Set().

wxDateTime::wxDateTime (double *jdn*)

Same as Set().

wxDateTime::wxDateTime (wxDateTime_t *hour*, wxDateTime_t *minute* = 0, wxDateTime_t *second* = 0, wxDateTime_t *millisec* = 0)

Same as Set().

wxDateTime::wxDateTime (wxDateTime_t *day*, Month *month*, int *year* = Inv_Year, wxDateTime_t *hour* = 0, wxDateTime_t *minute* = 0, wxDateTime_t *second* = 0, wxDateTime_t *millisec* = 0)

Same as Set().

wxDateTime::wxDateTime (const struct _SYSTEMTIME & *st*)

Same as SetFromMSWSysTime.

Parameters

st	Input, Windows SYSTEMTIME reference

Since

2.9.0

Remarks

MSW only Availability: only available for the wxMSW port.

21.170.7 Member Function Documentation

wxDateTime wxDateTime::Add (const wxDateSpan & *diff*) const

Adds the given date span to this object.

wxDateTime& wxDateTime::Add (const wxDateSpan & *diff*)

Adds the given date span to this object.

wxDateTime wxDateTime::Add (const wxTimeSpan & *diff*) const

Adds the given time span to this object.

wxDateTime& wxDateTime::Add (const wxTimeSpan & *diff*)

Adds the given time span to this object.

static int wxDateTime::ConvertYearToBC (int *year*) [static]

Converts the year in absolute notation (i.e. a number which can be negative, positive or zero) to the year in BC/AD notation.

For the positive years, nothing is done, but the year 0 is year 1 BC and so for other years there is a difference of 1.

This function should be used like this:

```
wxDateTime dt(...);
int y = dt.GetYear();
printf("The year is %d%s", wxDateTime::ConvertYearToBC(y), y > 0 ? "AD" : "BC");
```

wxDateSpan wxDateTime::DiffAsDateSpan (const wxDateTime & *dt*) const

Returns the difference between this object and *dt* as a wxDateSpan.

This method allows to find the number of entire years, months, weeks and days between *dt* and this date.

Since

2.9.5

wxString wxDateTime::Format (const wxString & *format* = wxDefaultDateTimeFormat, const TimeZone & *tz* = Local) const

This function does the same as the standard ANSI C `strftime(3)` function (http://www.cplusplus.com/reference/clibrary/ctime/strftime.html).

Please see its description for the meaning of *format* parameter.

Notice that POSIX `"%g"`, `"%G"`, `"%V"` and `"%z"` format specifiers are supported even if the standard library doesn't support them (e.g. MSVC).

It also accepts a few wxWidgets-specific extensions: you can optionally specify the width of the field to follow using `printf(3)`-like syntax and the format specification `"%l"` can be used to get the number of milliseconds.

See also

ParseFormat()

wxString wxDateTime::FormatDate ( ) const

Identical to calling Format() with "%x" argument (which means "preferred date representation for the current locale").

wxString wxDateTime::FormatISOCombined (char *sep* = 'T') const

Returns the combined date-time representation in the ISO 8601 format "YYYY-MM-DDTHH:MM:SS".

The *sep* parameter default value produces the result exactly corresponding to the ISO standard, but it can also be useful to use a space as separator if a more human-readable combined date-time representation is needed.

See also

FormatISODate(), FormatISOTime(), ParseISOCombined()

wxString wxDateTime::FormatISODate ( ) const

This function returns the date representation in the ISO 8601 format "YYYY-MM-DD".

wxString wxDateTime::FormatISOTime ( ) const

This function returns the time representation in the ISO 8601 format "HH:MM:SS".

wxString wxDateTime::FormatTime ( ) const

Identical to calling Format() with "%X" argument (which means "preferred time representation for the current locale").

wxDateTime wxDateTime::FromTimezone (const TimeZone & *tz*, bool *noDST* = false) const

Transform the date from the given time zone to the local one.

If *noDST* is true, no DST adjustments will be made.

Returns

The date in the local time zone.

static void wxDateTime::GetAmPmStrings (wxString * *am*, wxString * *pm*) [static]

Returns the translations of the strings AM and PM used for time formatting for the current locale.

Either of the pointers may be NULL if the corresponding value is not needed.

unsigned long wxDateTime::GetAsDOS ( ) const

Returns the date and time in DOS format.

void wxDateTime::GetAsMSWSysTime (struct _SYSTEMTIME * *st*) const

Returns the date and time in the Windows SYSTEMTIME format.

Parameters

st	Output, pointer to Windows SYSTEMTIME

Since

2.9.0

Remarks

MSW only Availability: only available for the wxMSW port.

static wxDateTime wxDateTime::GetBeginDST (int *year* = Inv_Year, Country *country* = Country_Default) [static]

Get the beginning of DST for the given country in the given year (current one by default).

This function suffers from limitations described in the DST overview.

See also

GetEndDST()

int wxDateTime::GetCentury (const TimeZone & *tz* = Local) const

Returns the century of this date.

static int wxDateTime::GetCentury (int *year*) [static]

Get the current century, i.e. first two digits of the year, in given calendar (only Gregorian is currently supported).

static Country wxDateTime::GetCountry ( ) [static]

Returns the current default country.

The default country is used for DST calculations, for example.

See also

SetCountry()

static Month wxDateTime::GetCurrentMonth (Calendar *cal* = Gregorian) [static]

Get the current month in given calendar (only Gregorian is currently supported).

static int wxDateTime::GetCurrentYear (Calendar *cal* = Gregorian) [static]

Get the current year in given calendar (only Gregorian is currently supported).

wxDateTime wxDateTime::GetDateOnly ( ) const

Returns the object having the same date component as this one but time of 00:00:00.

Since

2.8.2

See also

ResetTime()

unsigned short wxDateTime::GetDay (const TimeZone & *tz* = Local) const

Returns the day in the given timezone (local one by default).

unsigned short wxDateTime::GetDayOfYear (const TimeZone & *tz* = Local) const

Returns the day of the year (in 1-366 range) in the given timezone (local one by default).

static wxDateTime wxDateTime::GetEndDST (int *year* = Inv_Year, Country *country* = Country_Default) [static]

Returns the end of DST for the given country in the given year (current one by default).

See also

GetBeginDST()

static wxString wxDateTime::GetEnglishMonthName (Month *month*, NameFlags *flags* = Name_Full) [static]

Return the standard English name of the given month.

This function always returns "January" or "Jan" for January, use GetMonthName() to retrieve the name of the month in the users current locale.

Parameters

month	One of wxDateTime::Jan, ..., wxDateTime::Dec values.
flags	Either Name_Full (default) or Name_Abbr.

See also

GetEnglishWeekDayName()

Since

2.9.0

static wxString wxDateTime::GetEnglishWeekDayName (WeekDay *weekday*, NameFlags *flags* = Name_Full) [static]

Return the standard English name of the given week day.

This function always returns "Monday" or "Mon" for Monday, use GetWeekDayName() to retrieve the name of the month in the users current locale.

Parameters

weekday	One of wxDateTime::Sun, ..., wxDateTime::Sat values.
flags	Either Name_Full (default) or Name_Abbr.

See also

GetEnglishMonthName()

Since

2.9.0

unsigned short wxDateTime::GetHour (const TimeZone & *tz* = Local) const

Returns the hour in the given timezone (local one by default).

double wxDateTime::GetJDN ( ) const

Synonym for GetJulianDayNumber().

double wxDateTime::GetJulianDayNumber ( ) const

Returns the JDN corresponding to this date.

Beware of rounding errors!

See also

GetModifiedJulianDayNumber()

wxDateTime wxDateTime::GetLastMonthDay (Month *month* = Inv_Month, int *year* = Inv_Year) const

Returns the copy of this object to which SetToLastMonthDay() was applied.

wxDateTime wxDateTime::GetLastWeekDay (WeekDay *weekday*, Month *month* = Inv_Month, int *year* = Inv_Year)

Returns the copy of this object to which SetToLastWeekDay() was applied.

unsigned short wxDateTime::GetMillisecond (const TimeZone & *tz* = Local) const

Returns the milliseconds in the given timezone (local one by default).

unsigned short wxDateTime::GetMinute (const TimeZone & *tz* = Local) const

Returns the minute in the given timezone (local one by default).

double wxDateTime::GetMJD ( ) const

Synonym for GetModifiedJulianDayNumber().

double wxDateTime::GetModifiedJulianDayNumber ( ) const

Returns the *"Modified Julian Day Number"* (MJD) which is, by definition, is equal to JDN - 2400000.5.

The MJDs are simpler to work with as the integral MJDs correspond to midnights of the dates in the Gregorian calendar and not the noons like JDN. The MJD 0 represents Nov 17, 1858.

Month wxDateTime::GetMonth (const TimeZone & *tz* = Local) const

Returns the month in the given timezone (local one by default).

static wxString wxDateTime::GetMonthName (Month *month*, NameFlags *flags* = Name_Full) [static]

Gets the full (default) or abbreviated name of the given month.

This function returns the name in the current locale, use GetEnglishMonthName() to get the untranslated name if necessary.

Parameters

month	One of wxDateTime::Jan, ..., wxDateTime::Dec values.
flags	Either Name_Full (default) or Name_Abbr.

See also

GetWeekDayName()

wxDateTime wxDateTime::GetNextWeekDay (WeekDay *weekday*) const

Returns the copy of this object to which SetToNextWeekDay() was applied.

static wxDateTime_t wxDateTime::GetNumberOfDays (int *year*, Calendar *cal* = Gregorian) [static]

Returns the number of days in the given year.

The only supported value for *cal* currently is `Gregorian`.

static wxDateTime_t wxDateTime::GetNumberOfDays (Month *month*, int *year* = Inv_Year, Calendar *cal* = Gregorian) [static]

Returns the number of days in the given month of the given year.

The only supported value for *cal* currently is `Gregorian`.

wxDateTime wxDateTime::GetPrevWeekDay (WeekDay *weekday*) const

Returns the copy of this object to which SetToPrevWeekDay() was applied.

double wxDateTime::GetRataDie ( ) const

Return the *Rata* Die number of this date.

By definition, the Rata Die number is a date specified as the number of days relative to a base date of December 31 of the year 0. Thus January 1 of the year 1 is Rata Die day 1.

unsigned short wxDateTime::GetSecond (const TimeZone & *tz* = Local) const

Returns the seconds in the given timezone (local one by default).

time_t wxDateTime::GetTicks ( ) const

Returns the number of seconds since Jan 1, 1970 UTC.

An assert failure will occur if the date is not in the range covered by `time_t` type, use GetValue() if you work with dates outside of it.

static time_t wxDateTime::GetTimeNow ( ) [static]

Returns the current time.

Tm wxDateTime::GetTm (const TimeZone & *tz* = Local) const

Returns broken down representation of the date and time.

static tm* wxDateTime::GetTmNow (struct tm * *tm*) [static]

Returns the current time broken down using the buffer whose address is passed to the function with *tm* to store the result.

static tm* wxDateTime::GetTmNow ( ) [static]

Returns the current time broken down.

Note that this function returns a pointer to a static buffer that's reused by calls to this function and certain C library functions (e.g. localtime). If there is any chance your code might be used in a multi-threaded application, you really should use GetTmNow(struct tm *) instead.

int wxDateTime::GetWeekBasedYear (const TimeZone & *tz*) const

Returns the year to which the week containing this date belongs.

The value returned by this function is the same as the year, except, possibly, for a few days at the very beginning and very end of the year if they belong to a week which is mostly (i.e. at least 4 days) is in another year in which case that other (previous or next) year is returned.

For example, January 1 in 2015 belongs to the first year of 2015, hence GetWeekOfYear() for it returns 1 and this function returns 2015. However January 1 in 2016 belongs to the last week of 2015 according to ISO 8601 standard rules and so GetWeekOfYear() returns 53 and this function returns 2015, although GetYear() returns 2016.

Since

3.1.0

WeekDay wxDateTime::GetWeekDay (const TimeZone & *tz* = Local) const

Returns the week day in the given timezone (local one by default).

wxDateTime wxDateTime::GetWeekDay (WeekDay *weekday,* int *n* = 1, Month *month* = Inv_Month, int *year* = Inv_Year) const

Returns the copy of this object to which SetToWeekDay() was applied.

wxDateTime wxDateTime::GetWeekDayInSameWeek (WeekDay *weekday,* WeekFlags *flags* = Monday_First) const

Returns the copy of this object to which SetToWeekDayInSameWeek() was applied.

static wxString wxDateTime::GetWeekDayName (WeekDay *weekday,* NameFlags *flags* = Name_Full) [static]

Gets the full (default) or abbreviated name of the given week day.

This function returns the name in the current locale, use GetEnglishWeekDayName() to get the untranslated name if necessary.

Parameters

weekday	One of wxDateTime::Sun, ..., wxDateTime::Sat values.
flags	Either Name_Full (default) or Name_Abbr.

See also

GetMonthName()

wxDateTime_t wxDateTime::GetWeekOfMonth (WeekFlags *flags* = Monday_First, const TimeZone & *tz* = Local) const

Returns the ordinal number of the week in the month (in 1-5 range).

As GetWeekOfYear(), this function supports both conventions for the week start.

wxDateTime_t wxDateTime::GetWeekOfYear (WeekFlags *flags* = Monday_First, const TimeZone & *tz* = Local) const

Returns the number of the week of the year this date is in.

The first week of the year is, according to international standards, the one containing Jan 4 or, equivalently, the first week which has Thursday in this year. Both of these definitions are the same as saying that the first week of the year must contain more than half of its days in this year. Accordingly, the week number will always be in 1-53 range (52 for non-leap years).

The function depends on the week start convention specified by the *flags* argument but its results for `Sunday_First` are not well-defined as the ISO definition quoted above applies to the weeks starting on Monday only.

See also

GetWeekBasedYear()

int wxDateTime::GetYear (const TimeZone & *tz* = Local) const

Returns the year in the given timezone (local one by default).

wxDateTime wxDateTime::GetYearDay (wxDateTime_t *yday*) const

Returns the copy of this object to which SetToYearDay() was applied.

bool wxDateTime::IsBetween (const wxDateTime & *t1*, const wxDateTime & *t2*) const

Returns true if IsStrictlyBetween() is true or if the date is equal to one of the limit values.

See also

IsStrictlyBetween()

int wxDateTime::IsDST (Country *country* = Country_Default) const

Returns true if the DST is applied for this date in the given country.

See also

GetBeginDST(), GetEndDST()

static bool wxDateTime::IsDSTApplicable (int *year* = Inv_Year, Country *country* = Country_Default) [static]

Returns true if DST was used in the given year (the current one by default) in the given country.

bool wxDateTime::IsEarlierThan (const wxDateTime & *datetime*) const

Returns true if this date precedes the given one.

bool wxDateTime::IsEqualTo (const wxDateTime & *datetime*) const

Returns true if the two dates are strictly identical.

bool wxDateTime::IsEqualUpTo (const wxDateTime & *dt*, const wxTimeSpan & *ts*) const

Returns true if the date is equal to another one up to the given time interval, i.e. if the absolute difference between the two dates is less than this interval.

bool wxDateTime::IsLaterThan (const wxDateTime & *datetime*) const

Returns true if this date is later than the given one.

static bool wxDateTime::IsLeapYear (int *year* = Inv_Year, Calendar *cal* = Gregorian) [static]

Returns true if the *year* is a leap one in the specified calendar.

This functions supports Gregorian and Julian calendars.

bool wxDateTime::IsSameDate (const wxDateTime & *dt*) const

Returns true if the date is the same without comparing the time parts.

bool wxDateTime::IsSameTime (const wxDateTime & *dt*) const

Returns true if the time is the same (although dates may differ).

bool wxDateTime::IsStrictlyBetween (const wxDateTime & *t1*, const wxDateTime & *t2*) const

Returns true if this date lies strictly between the two given dates.

See also

IsBetween()

bool wxDateTime::IsValid ( ) const

Returns true if the object represents a valid time moment.

static bool wxDateTime::IsWestEuropeanCountry (Country *country* = Country_Default) [static]

This function returns true if the specified (or default) country is one of Western European ones.

It is used internally by wxDateTime to determine the DST convention and date and time formatting rules.

bool wxDateTime::IsWorkDay (Country *country* = Country_Default) const

Returns true is this day is not a holiday in the given country.

wxDateTime& wxDateTime::MakeFromTimezone (const TimeZone & *tz*, bool *noDST* = false)

Same as FromTimezone() but modifies the object in place.

wxDateTime& wxDateTime::MakeTimezone (const TimeZone & *tz*, bool *noDST* = false)

Modifies the object in place to represent the date in another time zone.

If *noDST* is true, no DST adjustments will be made.

wxDateTime& wxDateTime::MakeUTC (bool *noDST* = false)

This is the same as calling MakeTimezone() with the argument GMT0.

static wxDateTime wxDateTime::Now ( ) [static]

Returns the object corresponding to the current time.

Example:

```
wxDateTime now = wxDateTime::Now();
printf("Current time in Paris:\t%s\n", now.Format("%c", wxDateTime::CET).
      c_str());
```

Note

This function is accurate up to seconds. UNow() can be used if better precision is required.

See also

Today()

wxDateTime wxDateTime::operator+ (const wxDateSpan & *ds*) const

Adds the given date span to this object.

wxDateTime wxDateTime::operator+ (const wxTimeSpan & *ts*) const

Adds the given time span to this object.

wxDateTime& wxDateTime::operator+= (const wxDateSpan & *diff*)

Adds the given date span to this object.

wxDateTime& wxDateTime::operator+= (const wxTimeSpan & *diff*)

Adds the given time span to this object.

wxDateTime wxDateTime::operator- (const wxDateSpan & *ds*) const

Subtracts the given date span from this object.

wxDateTime wxDateTime::operator- (const wxTimeSpan & *ts*) const

Subtracts the given time span from this object.

wxTimeSpan wxDateTime::operator- (const wxDateTime & *dt2*) const

Subtracts another date from this one and returns the difference between them as a wxTimeSpan.

wxDateTime& wxDateTime::operator-= (const wxDateSpan & *diff*)

Subtracts the given date span from this object.

wxDateTime& wxDateTime::operator-= (const wxTimeSpan & *diff*)

Subtracts the given time span from this object.

wxDateTime& wxDateTime::operator= (time_t *timet*)

Same as Set().

wxDateTime& wxDateTime::operator= (const struct tm & *tm*)

Same as Set().

bool wxDateTime::ParseDate (const wxString & *date,* wxString::const_iterator * *end*)

This function is like ParseDateTime(), but it only allows the date to be specified.

It is thus less flexible then ParseDateTime(), but also has less chances to misinterpret the user input.

See ParseFormat() for the description of function parameters and return value.

See also

Format()

bool wxDateTime::ParseDateTime (const wxString & *datetime,* wxString::const_iterator * *end*)

Parses the string *datetime* containing the date and time in free format.

This function tries as hard as it can to interpret the given string as date and time. Unlike ParseRfc822Date(), it will accept anything that may be accepted and will only reject strings which cannot be parsed in any way at all. Notice that the function will fail if either date or time part is present but not both, use ParseDate() or ParseTime() to parse strings containing just the date or time component.

See ParseFormat() for the description of function parameters and return value.

bool wxDateTime::ParseFormat (const wxString & *date,* const wxString & *format,* const wxDateTime & *dateDef,* wxString::const_iterator * *end*)

This function parses the string *date* according to the given *format*.

The system `strptime(3)` function is used whenever available, but even if it is not, this function is still implemented, although support for locale-dependent format specifiers such as `"%c"`, `"%x"` or `"%X"` may not be perfect and GNU extensions such as `"%z"` and `"%Z"` are not implemented. This function does handle the month and weekday names in the current locale on all platforms, however.

Please see the description of the ANSI C function `strftime(3)` for the syntax of the format string.

The *dateDef* parameter is used to fill in the fields which could not be determined from the format string. For example, if the format is `"%d"` (the day of the month), the month and the year are taken from *dateDef*. If it is not specified, Today() is used as the default date.

Example of using this function:

```
wxDateTime dt;
wxString str = "...";
wxString::const_iterator end;
if ( !dt.ParseFormat(str, "%Y-%m-%d", &end) )
    ... parsing failed ...
else if ( end == str.end() )
    ... entire string parsed ...
else
    ... wxString(end, str.end()) left over ...
```

Parameters

date	The string to be parsed.
format	strptime()-like format string.
dateDef	Used to fill in the date components not specified in the *date* string.
end	Will be filled with the iterator pointing to the location where the parsing stopped if the function returns true. If the entire string was consumed, it is set to `date.end()`. Notice that this argument must be non-NULL.

Returns

true if at least part of the string was parsed successfully, false otherwise.

See also

Format()

bool wxDateTime::ParseFormat (const wxString & *date,* const wxString & *format,* wxString::const_iterator * *end*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxDateTime::ParseFormat (const wxString & *date,* wxString::const_iterator * *end*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxDateTime::ParseISOCombined (const wxString & *date,* char *sep* = 'T')

This function parses the string containing the date and time in ISO 8601 combined format "YYYY-MM-DDTHH:↩MM:SS".

The separator between the date and time parts must be equal to *sep* for the function to succeed.

Returns

true if the entire string was parsed successfully, false otherwise.

bool wxDateTime::ParseISODate (const wxString & *date*)

This function parses the date in ISO 8601 format "YYYY-MM-DD".

Returns

true if the entire string was parsed successfully, false otherwise.

bool wxDateTime::ParseISOTime (const wxString & *date*)

This function parses the time in ISO 8601 format "HH:MM:SS".

Returns

true if the entire string was parsed successfully, false otherwise.

bool wxDateTime::ParseRfc822Date (const wxString & *date,* wxString::const_iterator * *end*)

Parses the string *date* looking for a date formatted according to the RFC 822 in it.

The exact description of this format may, of course, be found in the RFC (section 5), but, briefly, this is the format used in the headers of Internet email messages and one of the most common strings expressing date in this format may be something like "Sat, 18 Dec 1999 00:48:30 +0100".

Returns NULL if the conversion failed, otherwise return the pointer to the character immediately following the part of the string which could be parsed. If the entire string contains only the date in RFC 822 format, the returned pointer will be pointing to a NUL character.

This function is intentionally strict, it will return an error for any string which is not RFC 822 compliant. If you need to parse date formatted in more free ways, you should use ParseDateTime() or ParseDate() instead.

See ParseFormat() for the description of function parameters and return value.

bool wxDateTime::ParseTime (const wxString & *time*, wxString::const_iterator * *end*)

This functions is like ParseDateTime(), but only allows the time to be specified in the input string.

See ParseFormat() for the description of function parameters and return value.

wxDateTime& wxDateTime::ResetTime ( )

Reset time to midnight (00:00:00) without changing the date.

wxDateTime& wxDateTime::Set (time_t *timet*)

Constructs the object from *timet* value holding the number of seconds since Jan 1, 1970 UTC.

If *timet* is invalid, i.e.

```
(time_t)-1
```

, wxDateTime becomes invalid too, i.e. its IsValid() will return false.

wxDateTime& wxDateTime::Set (const struct tm & *tm*)

Sets the date and time from the broken down representation in the standard *tm* structure.

wxDateTime& wxDateTime::Set (const Tm & *tm*)

Sets the date and time from the broken down representation in the *wxDateTime::Tm* structure.

wxDateTime& wxDateTime::Set (double *jdn*)

Sets the date from the so-called Julian Day Number.

By definition, the Julian Day Number, usually abbreviated as JDN, of a particular instant is the fractional number of days since 12 hours Universal Coordinated Time (Greenwich mean noon) on January 1 of the year -4712 in the Julian proleptic calendar.

wxDateTime& wxDateTime::Set (wxDateTime_t *hour*, wxDateTime_t *minute* = 0, wxDateTime_t *second* = 0, wxDateTime_t *millisec* = 0)

Sets the date to be equal to Today() and the time from supplied parameters.

See the full Set() overload for the remarks about DST.

wxDateTime& wxDateTime::Set (wxDateTime_t *day*, Month *month*, int *year* = Inv_Year, wxDateTime_t *hour* = 0, wxDateTime_t *minute* = 0, wxDateTime_t *second* = 0, wxDateTime_t *millisec* = 0)

Sets the date and time from the parameters.

If the function parameters are invalid, e.g. *month* is February and *day* is 30, the object is left in an invalid state, i.e. IsValid() method will return false.

If the specified time moment is invalid due to DST, i.e. it falls into the "missing" hour on the date on which the DST starts, a valid wxDateTime object is still constructed but its hour component is moved forward to ensure that it corresponds to a valid moment in the local time zone. For example, in the CET time zone the DST started on 2013-03-31T02:00:00 in 2013 and so setting the object to 2:30 at this date actually sets the hour to 3, and not 2.

static void wxDateTime::SetCountry (Country *country*) `[static]`

Sets the country to use by default.

This setting influences the DST calculations, date formatting and other things.

See also

GetCountry()

wxDateTime& wxDateTime::SetDay (unsigned short *day*)

Sets the day without changing other date components.

wxDateTime& wxDateTime::SetFromDOS (unsigned long *ddt*)

Sets the date from the date and time in DOS format.

wxDateTime& wxDateTime::SetFromMSWSysTime (const struct _SYSTEMTIME & *st*)

Initialize using the Windows SYSTEMTIME structure.

Parameters

st	Input, Windows SYSTEMTIME reference

Since

2.9.0

Remarks

MSW only Availability: only available for the wxMSW port.

wxDateTime& wxDateTime::SetHour (unsigned short *hour*)

Sets the hour without changing other date components.

wxDateTime& wxDateTime::SetMillisecond (unsigned short *millisecond*)

Sets the millisecond without changing other date components.

wxDateTime& wxDateTime::SetMinute (unsigned short *minute*)

Sets the minute without changing other date components.

wxDateTime& wxDateTime::SetMonth (Month *month*)

Sets the month without changing other date components.

wxDateTime& wxDateTime::SetSecond (unsigned short *second*)

Sets the second without changing other date components.

wxDateTime& wxDateTime::SetToCurrent ( )

Sets the date and time of to the current values.

Same as assigning the result of Now() to this object.

wxDateTime& wxDateTime::SetToLastMonthDay (Month *month* = Inv_Month, int *year* = Inv_Year)

Sets the date to the last day in the specified month (the current one by default).

Returns

The reference to the modified object itself.

bool wxDateTime::SetToLastWeekDay (WeekDay *weekday*, Month *month* = Inv_Month, int *year* = Inv_Year)

The effect of calling this function is the same as of calling `SetToWeekDay`(-1, weekday, month, year).

The date will be set to the last *weekday* in the given month and year (the current ones by default). Always returns true.

wxDateTime& wxDateTime::SetToNextWeekDay (WeekDay *weekday*)

Sets the date so that it will be the first *weekday* following the current date.

Returns

The reference to the modified object itself.

wxDateTime& wxDateTime::SetToPrevWeekDay (WeekDay *weekday*)

Sets the date so that it will be the last *weekday* before the current date.

Returns

The reference to the modified object itself.

bool wxDateTime::SetToWeekDay (WeekDay *weekday*, int *n* = `1`, Month *month* = Inv_Month, int *year* = Inv_Year)

Sets the date to the *n-th weekday* in the given month of the given year (the current month and year are used by default).

The parameter *n* may be either positive (counting from the beginning of the month) or negative (counting from the end of it).

For example, SetToWeekDay(2, wxDateTime::Wed) will set the date to the second Wednesday in the current month and SetToWeekDay(-1, wxDateTime::Sun) will set the date to the last Sunday in the current month.

Returns

true if the date was modified successfully, false otherwise meaning that the specified date doesn't exist.

wxDateTime& wxDateTime::SetToWeekDayInSameWeek (WeekDay *weekday,* WeekFlags *flags* = Monday_First)

Adjusts the date so that it will still lie in the same week as before, but its week day will be the given one.

Returns

The reference to the modified object itself.

static wxDateTime wxDateTime::SetToWeekOfYear (int *year,* wxDateTime_t *numWeek,* WeekDay *weekday* = Mon) [static]

Set the date to the given *weekday* in the week number *numWeek* of the given *year* .

The number should be in range 1-53.

Note that the returned date may be in a different year than the one passed to this function because both the week 1 and week 52 or 53 (for leap years) contain days from different years. See GetWeekOfYear() for the explanation of how the year weeks are counted.

wxDateTime& wxDateTime::SetToYearDay (wxDateTime_t *yday*)

Sets the date to the day number *yday* in the same year (i.e. unlike the other functions, this one does not use the current year).

The day number should be in the range 1-366 for the leap years and 1-365 for the other ones.

Returns

The reference to the modified object itself.

wxDateTime& wxDateTime::SetYear (int *year*)

Sets the year without changing other date components.

wxDateTime wxDateTime::Subtract (const wxTimeSpan & *diff*) const

Subtracts the given time span from this object.

wxDateTime& wxDateTime::Subtract (const wxTimeSpan & *diff*)

Subtracts the given time span from this object.

wxDateTime wxDateTime::Subtract (const wxDateSpan & *diff*) const

Subtracts the given date span from this object.

wxDateTime& wxDateTime::Subtract (const wxDateSpan & *diff*)

Subtracts the given date span from this object.

wxTimeSpan wxDateTime::Subtract (const wxDateTime & *dt*) const

Subtracts another date from this one and returns the difference between them as a wxTimeSpan.

static wxDateTime wxDateTime::Today ( ) [static]

Returns the object corresponding to the midnight of the current day (i.e. the same as Now(), but the time part is set to 0).

See also

Now()

wxDateTime wxDateTime::ToTimezone (const TimeZone & *tz*, bool *noDST* = false) const

Transform the date to the given time zone.

If *noDST* is true, no DST adjustments will be made.

Returns

The date in the new time zone.

wxDateTime wxDateTime::ToUTC (bool *noDST* = false) const

This is the same as calling ToTimezone() with the argument GMT0.

static wxDateTime wxDateTime::UNow ( ) [static]

Returns the object corresponding to the current UTC time including the milliseconds.

Notice that unlike Now(), this method creates a wxDateTime object corresponding to UTC, not local, time.

See also

Now(), wxGetUTCTimeMillis()

21.171 wxDateTimeHolidayAuthority Class Reference

`#include <wx/datetime.h>`

21.171.1 Detailed Description

Todo Write wxDateTimeHolidayAuthority documentation.

Library: wxBase

Category: Data Structures

21.172 wxDateTimeWorkDays Class Reference

`#include <wx/datetime.h>`

21.172.1 Detailed Description

Todo Write wxDateTimeWorkDays documentation.

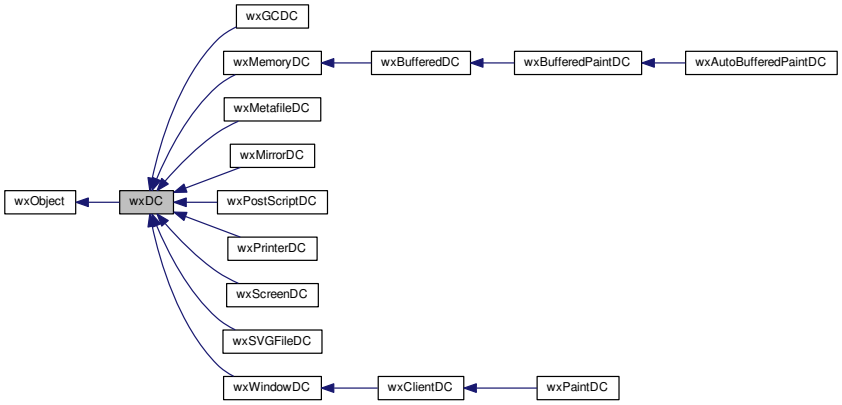
Library: wxBase

Category: Data Structures

21.173 wxDC Class Reference

`#include <wx/dc.h>`

Inheritance diagram for wxDC:



21.173.1 Detailed Description

A wxDC is a *"device context"* onto which graphics and text can be drawn.

It is intended to represent different output devices and offers a common abstract API for drawing on any of them.

wxWidgets offers an alternative drawing API based on the modern drawing backends GDI+, CoreGraphics and Cairo. See wxGraphicsContext, wxGraphicsRenderer and related classes. There is also a wxGCDC linking the APIs by offering the wxDC API on top of a wxGraphicsContext.

wxDC is an abstract base class and cannot be created directly. Use wxPaintDC, wxClientDC, wxWindowDC, wxScreenDC, wxMemoryDC or wxPrinterDC. Notice that device contexts which are associated with windows (i.e. wxClientDC, wxWindowDC and wxPaintDC) use the window font and colours by default (starting with wxWidgets 2.9.0) but the other device context classes use system-default values so you always must set the appropriate fonts and colours before using them.

In addition to the versions of the methods documented below, there are also versions which accept single wxPoint parameter instead of the two wxCoord ones or wxPoint and wxSize instead of the four wxCoord parameters.

Beginning with wxWidgets 2.9.0 the entire wxDC code has been reorganized. All platform dependent code (actually all drawing code) has been moved into backend classes which derive from a common wxDCImpl class. The user-visible classes such as wxClientDC and wxPaintDC merely forward all calls to the backend implementation.

21.173.2 Device and logical units

In the wxDC context there is a distinction between *logical* units and *device* units.

Device units are the units native to the particular device; e.g. for a screen, a device unit is a *pixel*. For a printer, the device unit is defined by the resolution of the printer (usually given in `DPI:` dot-per-inch).

All wxDC functions use instead **logical** units, unless where explicitly stated. Logical units are arbitrary units mapped to device units using the current mapping mode (see wxDC::SetMapMode).

This mechanism allows to reuse the same code which prints on e.g. a window on the screen to print on e.g. a paper.

21.173.3 Support for Transparency / Alpha Channel

In general wxDC methods don't support alpha transparency and the alpha component of wxColour is simply ignored and you need to use wxGraphicsContext for full transparency support. There are, however, a few exceptions: first, under Mac OS X colours with alpha channel are supported in all the normal wxDC-derived classes as they use wxGraphicsContext internally. Second, under all platforms wxSVGFileDC also fully supports alpha channel. In both of these cases the instances of wxPen or wxBrush that are built from wxColour use the colour's alpha values when stroking or filling.

21.173.4 for Transformation Matrix

On some platforms (currently only under MSW and only on Windows NT, i.e. not Windows 9x/ME, systems) wxDC has support for applying an arbitrary affine transformation matrix to its coordinate system. Call CanUseTransform↩ Matrix() to check if this support is available and then call SetTransformMatrix() if it is. If the transformation matrix is not supported, SetTransformMatrix() always simply returns false and doesn't do anything.

Library: wxCore

Category: Device Contexts, Graphics Device Interface (GDI)

See also

Device Contexts, wxGraphicsContext, wxDCFontChanger, wxDCTextColourChanger, wxDCPenChanger, wxDCBrushChanger, wxDCClipper

Todo Precise definition of default/initial state.

Pixelwise definition of operations (e.g. last point of a line not drawn).

Public Member Functions

- void CopyAttributes (const wxDC &dc)

  Copy attributes from another DC.
- int GetDepth () const

  Returns the depth (number of bits/pixel) of this DC.
- wxPoint GetDeviceOrigin () const

  Returns the current device origin.
- wxRasterOperationMode GetLogicalFunction () const

  Gets the current logical function.
- wxMappingMode GetMapMode () const

  Gets the current mapping mode for the device context.

- bool GetPixel (wxCoord x, wxCoord y, wxColour *colour) const

  Gets in colour the colour at the specified location.
- wxSize GetPPI () const

  Returns the resolution of the device in pixels per inch.
- void GetSize (wxCoord *width, wxCoord *height) const

  Gets the horizontal and vertical extent of this device context in device units.
- wxSize GetSize () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void GetSizeMM (wxCoord *width, wxCoord *height) const

  Returns the horizontal and vertical resolution in millimetres.
- wxSize GetSizeMM () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void GetUserScale (double *x, double *y) const

  Gets the current user scale factor.
- bool IsOk () const

  Returns true if the DC is ok to use.
- void SetAxisOrientation (bool xLeftRight, bool yBottomUp)

  Sets the x and y axis orientation (i.e. the direction from lowest to highest values on the axis).
- void SetDeviceOrigin (wxCoord x, wxCoord y)

  Sets the device origin (i.e. the origin in pixels after scaling has been applied).
- void SetLogicalFunction (wxRasterOperationMode function)

  Sets the current logical function for the device context.
- void SetMapMode (wxMappingMode mode)

  The mapping mode of the device context defines the unit of measurement used to convert logical units to device units.
- void SetPalette (const wxPalette &palette)

  If this is a window DC or memory DC, assigns the given palette to the window or bitmap associated with the DC.
- void SetUserScale (double xScale, double yScale)

  Sets the user scaling factor, useful for applications which require 'zooming'.
- void * GetHandle () const

  Returns a value that can be used as a handle to the native drawing context, if this wxDC has something that could be thought of in that way.
- wxBitmap GetAsBitmap (const wxRect *subrect=NULL) const

  If supported by the platform and the type of DC, fetch the contents of the DC, or a subset of it, as a bitmap.
- void SetLogicalScale (double x, double y)
- void GetLogicalScale (double *x, double *y) const
- void SetLogicalOrigin (wxCoord x, wxCoord y)
- void GetLogicalOrigin (wxCoord *x, wxCoord *y) const
- wxPoint GetLogicalOrigin () const

Coordinate conversion functions

- wxCoord DeviceToLogicalX (wxCoord x) const

  Convert device X coordinate to logical coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.
- wxCoord DeviceToLogicalXRel (wxCoord x) const

  Convert device X coordinate to relative logical coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.
- wxCoord DeviceToLogicalY (wxCoord y) const

  Converts device Y coordinate to logical coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.
- wxCoord DeviceToLogicalYRel (wxCoord y) const

Convert device Y coordinate to relative logical coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.

- wxCoord LogicalToDeviceX (wxCoord x) const

  Converts logical X coordinate to device coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.
- wxCoord LogicalToDeviceXRel (wxCoord x) const

  Converts logical X coordinate to relative device coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.
- wxCoord LogicalToDeviceY (wxCoord y) const

  Converts logical Y coordinate to device coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.
- wxCoord LogicalToDeviceYRel (wxCoord y) const

  Converts logical Y coordinate to relative device coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.

Drawing functions

- void Clear ()

  Clears the device context using the current background brush.
- void DrawArc (wxCoord xStart, wxCoord yStart, wxCoord xEnd, wxCoord yEnd, wxCoord xc, wxCoord yc)

  Draws an arc from the given start to the given end point.
- void DrawArc (const wxPoint &ptStart, const wxPoint &ptEnd, const wxPoint &centre)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawBitmap (const wxBitmap &bitmap, wxCoord x, wxCoord y, bool useMask=false)

  Draw a bitmap on the device context at the specified point.
- void DrawBitmap (const wxBitmap &bmp, const wxPoint &pt, bool useMask=false)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawCheckMark (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Draws a check mark inside the given rectangle.
- void DrawCheckMark (const wxRect &rect)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawCircle (wxCoord x, wxCoord y, wxCoord radius)

  Draws a circle with the given centre and radius.
- void DrawCircle (const wxPoint &pt, wxCoord radius)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawEllipse (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Draws an ellipse contained in the rectangle specified either with the given top left corner and the given size or directly.
- void DrawEllipse (const wxPoint &pt, const wxSize &size)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawEllipse (const wxRect &rect)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawEllipticArc (wxCoord x, wxCoord y, wxCoord width, wxCoord height, double start, double end)

  Draws an arc of an ellipse.
- void DrawEllipticArc (const wxPoint &pt, const wxSize &sz, double sa, double ea)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawIcon (const wxIcon &icon, wxCoord x, wxCoord y)

  Draw an icon on the display (does nothing if the device context is PostScript).
- void DrawIcon (const wxIcon &icon, const wxPoint &pt)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawLabel (const wxString &text, const wxBitmap &bitmap, const wxRect &rect, int alignment=wxALIGN_LEFT|wxALIGN_TOP, int indexAccel=-1, wxRect *rectBounding=NULL)

Draw optional bitmap and the text into the given rectangle and aligns it as specified by alignment parameter; it also will emphasize the character with the given index if it is != -1 and return the bounding rectangle if required.

- void DrawLabel (const wxString &text, const wxRect &rect, int alignment=wxALIGN_LEFT|wxALIGN_TOP, int indexAccel=-1)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawLine (wxCoord x1, wxCoord y1, wxCoord x2, wxCoord y2)

  Draws a line from the first point to the second.
- void DrawLine (const wxPoint &pt1, const wxPoint &pt2)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawLines (int n, const wxPoint points[], wxCoord xoffset=0, wxCoord yoffset=0)

  Draws lines using an array of points of size n adding the optional offset coordinate.
- void DrawLines (const wxPointList *points, wxCoord xoffset=0, wxCoord yoffset=0)

  This method uses a list of wxPoints, adding the optional offset coordinate.
- void DrawPoint (wxCoord x, wxCoord y)

  Draws a point using the color of the current pen.
- void DrawPoint (const wxPoint &pt)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawPolygon (int n, const wxPoint points[], wxCoord xoffset=0, wxCoord yoffset=0, wxPolygonFill↩ Mode fill_style=wxODDEVEN_RULE)

  Draws a filled polygon using an array of points of size n, adding the optional offset coordinate.
- void DrawPolygon (const wxPointList *points, wxCoord xoffset=0, wxCoord yoffset=0, wxPolygonFillMode fill_style=wxODDEVEN_RULE)

  This method draws a filled polygon using a list of wxPoints, adding the optional offset coordinate.
- void DrawPolyPolygon (int n, const int count[], const wxPoint points[], wxCoord xoffset=0, wxCoord yoffset=0, wxPolygonFillMode fill_style=wxODDEVEN_RULE)

  Draws two or more filled polygons using an array of points, adding the optional offset coordinates.
- void DrawRectangle (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Draws a rectangle with the given top left corner, and with the given size.
- void DrawRectangle (const wxPoint &pt, const wxSize &sz)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawRectangle (const wxRect &rect)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawRotatedText (const wxString &text, wxCoord x, wxCoord y, double angle)

  Draws the text rotated by angle degrees (positive angles are counterclockwise; the full angle is 360 degrees).
- void DrawRotatedText (const wxString &text, const wxPoint &point, double angle)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawRoundedRectangle (wxCoord x, wxCoord y, wxCoord width, wxCoord height, double radius)

  Draws a rectangle with the given top left corner, and with the given size.
- void DrawRoundedRectangle (const wxPoint &pt, const wxSize &sz, double radius)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawRoundedRectangle (const wxRect &rect, double radius)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawSpline (int n, const wxPoint points[])

  Draws a spline between all given points using the current pen.
- void DrawSpline (const wxPointList *points)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawSpline (wxCoord x1, wxCoord y1, wxCoord x2, wxCoord y2, wxCoord x3, wxCoord y3)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void DrawText (const wxString &text, wxCoord x, wxCoord y)

Draws a text string at the specified point, using the current text font, and the current text foreground and background colours.

- void DrawText (const wxString &text, const wxPoint &pt)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void GradientFillConcentric (const wxRect &rect, const wxColour &initialColour, const wxColour &destColour)

  Fill the area specified by rect with a radial gradient, starting from initialColour at the centre of the circle and fading to destColour on the circle outside.
- void GradientFillConcentric (const wxRect &rect, const wxColour &initialColour, const wxColour &destColour, const wxPoint &circleCenter)

  Fill the area specified by rect with a radial gradient, starting from initialColour at the centre of the circle and fading to destColour on the circle outside.
- void GradientFillLinear (const wxRect &rect, const wxColour &initialColour, const wxColour &destColour, wxDirection nDirection=wxRIGHT)

  Fill the area specified by rect with a linear gradient, starting from initialColour and eventually fading to destColour.
- bool FloodFill (wxCoord x, wxCoord y, const wxColour &colour, wxFloodFillStyle style=wxFLOOD_SURFACE)

  Flood fills the device context starting from the given point, using the current brush colour, and using a style:
- bool FloodFill (const wxPoint &pt, const wxColour &col, wxFloodFillStyle style=wxFLOOD_SURFACE)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void CrossHair (wxCoord x, wxCoord y)

  Displays a cross hair using the current pen.
- void CrossHair (const wxPoint &pt)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Clipping region functions

- void DestroyClippingRegion ()

  Destroys the current clipping region so that none of the DC is clipped.
- void GetClippingBox (wxCoord *x, wxCoord *y, wxCoord *width, wxCoord *height) const

  Gets the rectangle surrounding the current clipping region.
- void SetClippingRegion (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Sets the clipping region for this device context to the intersection of the given region described by the parameters of this method and the previously set clipping region.
- void SetClippingRegion (const wxPoint &pt, const wxSize &sz)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetClippingRegion (const wxRect &rect)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetDeviceClippingRegion (const wxRegion &region)

  Sets the clipping region for this device context.

Text/character extent functions

- wxCoord GetCharHeight () const

  Gets the character height of the currently set font.
- wxCoord GetCharWidth () const

  Gets the average character width of the currently set font.
- wxFontMetrics GetFontMetrics () const

  Returns the various font characteristics.
- void GetMultiLineTextExtent (const wxString &string, wxCoord *w, wxCoord *h, wxCoord *heightLine=NULL, const wxFont *font=NULL) const

  Gets the dimensions of the string using the currently selected font.
- wxSize GetMultiLineTextExtent (const wxString &string) const

  Gets the dimensions of the string using the currently selected font.

- bool GetPartialTextExtents (const wxString &text, wxArrayInt &widths) const

  Fills the widths array with the widths from the beginning of text to the corresponding character of text.
- void GetTextExtent (const wxString &string, wxCoord *w, wxCoord *h, wxCoord *descent=NULL, wxCoord *externalLeading=NULL, const wxFont *font=NULL) const

  Gets the dimensions of the string using the currently selected font.
- wxSize GetTextExtent (const wxString &string) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Text properties functions

- int GetBackgroundMode () const

  Returns the current background mode: `wxPENSTYLE_SOLID` or `wxPENSTYLE_TRANSPARENT`.
- const wxFont & GetFont () const

  Gets the current font.
- wxLayoutDirection GetLayoutDirection () const

  Gets the current layout direction of the device context.
- const wxColour & GetTextBackground () const

  Gets the current text background colour.
- const wxColour & GetTextForeground () const

  Gets the current text foreground colour.
- void SetBackgroundMode (int mode)

  mode may be one of `wxPENSTYLE_SOLID` and `wxPENSTYLE_TRANSPARENT`.
- void SetFont (const wxFont &font)

  Sets the current font for the DC.
- void SetTextBackground (const wxColour &colour)

  Sets the current text background colour for the DC.
- void SetTextForeground (const wxColour &colour)

  Sets the current text foreground colour for the DC.
- void SetLayoutDirection (wxLayoutDirection dir)

  Sets the current layout direction for the device context.

Bounding box functions

- void CalcBoundingBox (wxCoord x, wxCoord y)

  Adds the specified point to the bounding box which can be retrieved with MinX(), MaxX() and MinY(), MaxY() functions.
- wxCoord MaxX () const

  Gets the maximum horizontal extent used in drawing commands so far.
- wxCoord MaxY () const

  Gets the maximum vertical extent used in drawing commands so far.
- wxCoord MinX () const

  Gets the minimum horizontal extent used in drawing commands so far.
- wxCoord MinY () const

  Gets the minimum vertical extent used in drawing commands so far.
- void ResetBoundingBox ()

  Resets the bounding box: after a call to this function, the bounding box doesn't contain anything.

Page and document start/end functions

- bool StartDoc (const wxString &message)

  Starts a document (only relevant when outputting to a printer).
- void StartPage ()

  Starts a document page (only relevant when outputting to a printer).
- void EndDoc ()

  Ends a document (only relevant when outputting to a printer).
- void EndPage ()

  Ends a document page (only relevant when outputting to a printer).

Bit-Block Transfer operations (blit)

- bool Blit (wxCoord xdest, wxCoord ydest, wxCoord width, wxCoord height, wxDC *source, wxCoord xsrc, wxCoord ysrc, wxRasterOperationMode logicalFunc=wxCOPY, bool useMask=false, wxCoord xsrcMask=wxDefaultCoord, wxCoord ysrcMask=wxDefaultCoord)
  Copy from a source DC to this DC.
- bool StretchBlit (wxCoord xdest, wxCoord ydest, wxCoord dstWidth, wxCoord dstHeight, wxDC *source, wxCoord xsrc, wxCoord ysrc, wxCoord srcWidth, wxCoord srcHeight, wxRasterOperationMode logicalFunc=wxCOPY, bool useMask=false, wxCoord xsrcMask=wxDefaultCoord, wxCoord ysrcMask=wxDefaultCoord)
  Copy from a source DC to this DC possibly changing the scale.

Background/foreground brush and pen

- const wxBrush & GetBackground () const
  Gets the brush used for painting the background.
- const wxBrush & GetBrush () const
  Gets the current brush.
- const wxPen & GetPen () const
  Gets the current pen.
- void SetBackground (const wxBrush &brush)
  Sets the current background brush for the DC.
- void SetBrush (const wxBrush &brush)
  Sets the current brush for the DC.
- void SetPen (const wxPen &pen)
  Sets the current pen for the DC.

Transformation matrix

See the notes about the availability of these functions in the class documentation.

- bool CanUseTransformMatrix () const
  Check if the use of transformation matrix is supported by the current system.
- bool SetTransformMatrix (const wxAffineMatrix2D &matrix)
  Set the transformation matrix.
- wxAffineMatrix2D GetTransformMatrix () const
  Return the transformation matrix used by this device context.
- void ResetTransformMatrix ()
  Revert the transformation matrix to identity matrix.

query capabilities

- bool CanDrawBitmap () const
  Does the DC support drawing bitmaps?
- bool CanGetTextExtent () const
  Does the DC support calculating the size required to draw text?

Additional Inherited Members

21.173.5 Member Function Documentation

bool wxDC::Blit (wxCoord *xdest*, wxCoord *ydest*, wxCoord *width*, wxCoord *height*, wxDC * *source*, wxCoord *xsrc*, wxCoord *ysrc*, wxRasterOperationMode *logicalFunc* = wxCOPY, bool *useMask* = `false`, wxCoord *xsrcMask* = wxDefaultCoord, wxCoord *ysrcMask* = wxDefaultCoord)

Copy from a source DC to this DC.

With this method you can specify the destination coordinates and the size of area to copy which will be the same for both the source and target DCs. If you need to apply scaling while copying, use StretchBlit().

Notice that source DC coordinates *xsrc* and *ysrc* are interpreted using the current source DC coordinate system, i.e. the scale, origin position and axis directions are taken into account when transforming them to physical (pixel) coordinates.

Parameters

xdest	Destination device context x position.
ydest	Destination device context y position.
width	Width of source area to be copied.
height	Height of source area to be copied.
source	Source device context.
xsrc	Source device context x position.
ysrc	Source device context y position.
logicalFunc	Logical function to use, see SetLogicalFunction().
useMask	If true, Blit does a transparent blit using the mask that is associated with the bitmap selected into the source device context. The Windows implementation does the following if MaskBlt cannot be used: 1. Creates a temporary bitmap and copies the destination area into it. 2. Copies the source area into the temporary bitmap using the specified logical function. 3. Sets the masked area in the temporary bitmap to BLACK by ANDing the mask bitmap with the temp bitmap with the foreground colour set to WHITE and the bg colour set to BLACK. 4. Sets the unmasked area in the destination area to BLACK by ANDing the mask bitmap with the destination area with the foreground colour set to BLACK and the background colour set to WHITE. 5. ORs the temporary bitmap with the destination area. 6. Deletes the temporary bitmap. This sequence of operations ensures that the source's transparent area need not be black, and logical functions are supported. **Note:** on Windows, blitting with masks can be speeded up considerably by compiling wx↩ Widgets with the wxUSE_DC_CACHEING option enabled. You can also influence whether MaskBlt or the explicit mask blitting code above is used, by using wxSystemOptions and setting the `no-maskblt` option to 1.

xsrcMask	Source x position on the mask. If both xsrcMask and ysrcMask are `-1`, xsrc and ysrc will be assumed for the mask source position. Currently only implemented on Windows.
ysrcMask	Source y position on the mask. If both xsrcMask and ysrcMask are `-1`, xsrc and ysrc will be assumed for the mask source position. Currently only implemented on Windows.

Remarks

There is partial support for Blit() in wxPostScriptDC, under X.

See also

StretchBlit(), wxMemoryDC, wxBitmap, wxMask

void wxDC::CalcBoundingBox (wxCoord *x*, wxCoord *y*)

Adds the specified point to the bounding box which can be retrieved with MinX(), MaxX() and MinY(), MaxY() functions.

See also

ResetBoundingBox()

bool wxDC::CanDrawBitmap ( ) const

Does the DC support drawing bitmaps?

bool wxDC::CanGetTextExtent ( ) const

Does the DC support calculating the size required to draw text?

bool wxDC::CanUseTransformMatrix ( ) const

Check if the use of transformation matrix is supported by the current system.

Currently this function always returns false for non-MSW platforms and may return false for old (Windows 9x/ME) Windows systems. Normally support for the transformation matrix is always available in any relatively recent Windows versions.

Since

2.9.2

void wxDC::Clear ( )

Clears the device context using the current background brush.

void wxDC::CopyAttributes (const wxDC & *dc*)

Copy attributes from another DC.

The copied attributes currently are:

- Font

- Text foreground and background colours
- Background brush
- Layout direction

Parameters

dc	A valid (i.e. its IsOk() must return true) source device context.

void wxDC::CrossHair (wxCoord *x*, wxCoord *y*)

Displays a cross hair using the current pen.

This is a vertical and horizontal line the height and width of the window, centred on the given point.

void wxDC::CrossHair (const wxPoint & *pt*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DestroyClippingRegion ( )

Destroys the current clipping region so that none of the DC is clipped.

See also

SetClippingRegion()

wxCoord wxDC::DeviceToLogicalX (wxCoord *x*) const

Convert *device* X coordinate to logical coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.

wxCoord wxDC::DeviceToLogicalXRel (wxCoord *x*) const

Convert *device* X coordinate to relative logical coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.

Use this for converting a width, for example.

wxCoord wxDC::DeviceToLogicalY (wxCoord *y*) const

Converts *device* Y coordinate to logical coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.

wxCoord wxDC::DeviceToLogicalYRel (wxCoord *y*) const

Convert *device* Y coordinate to relative logical coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.

Use this for converting a height, for example.

void wxDC::DrawArc (wxCoord *xStart,* wxCoord *yStart,* wxCoord *xEnd,* wxCoord *yEnd,* wxCoord *xc,* wxCoord *yc*)

Draws an arc from the given start to the given end point.

Note

DrawEllipticArc() has more clear semantics and it is recommended to use it instead of this function.

The arc drawn is an arc of the circle centered at (*xc*, *yc*). Its start point is (*xStart*, *yStart*) whereas its end point is the point of intersection of the line passing by (*xc*, *yc*) and (*xEnd*, *yEnd*) with the circle passing by (*xStart*, *yStart*).

The arc is drawn in a counter-clockwise direction between the start and the end points.

The current pen is used for the outline and the current brush for filling the shape. Notice that unless the brush is transparent, the lines connecting the centre of the circle to the end points of the arc are drawn as well.

void wxDC::DrawArc (const wxPoint & *ptStart,* const wxPoint & *ptEnd,* const wxPoint & *centre*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawBitmap (const wxBitmap & *bitmap,* wxCoord *x,* wxCoord *y,* bool *useMask* = `false`)

Draw a bitmap on the device context at the specified point.

If *transparent* is true and the bitmap has a transparency mask, the bitmap will be drawn transparently.

When drawing a mono-bitmap, the current text foreground colour will be used to draw the foreground of the bitmap (all bits set to 1), and the current text background colour to draw the background (all bits set to 0).

See also

SetTextForeground(), SetTextBackground(), wxMemoryDC

void wxDC::DrawBitmap (const wxBitmap & *bmp,* const wxPoint & *pt,* bool *useMask* = `false`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawCheckMark (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Draws a check mark inside the given rectangle.

void wxDC::DrawCheckMark (const wxRect & *rect*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawCircle (wxCoord *x,* wxCoord *y,* wxCoord *radius*)

Draws a circle with the given centre and radius.

See also

DrawEllipse()

void wxDC::DrawCircle (const wxPoint & *pt,* wxCoord *radius*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawEllipse (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Draws an ellipse contained in the rectangle specified either with the given top left corner and the given size or directly.

The current pen is used for the outline and the current brush for filling the shape.

See also

DrawCircle()

void wxDC::DrawEllipse (const wxPoint & *pt,* const wxSize & *size*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawEllipse (const wxRect & *rect*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawEllipticArc (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height,* double *start,* double *end*)

Draws an arc of an ellipse.

The current pen is used for drawing the arc and the current brush is used for drawing the pie.

x and *y* specify the x and y coordinates of the upper-left corner of the rectangle that contains the ellipse.

width and *height* specify the width and height of the rectangle that contains the ellipse.

start and *end* specify the end points of the arc relative to the three-o'clock position from the center of the rectangle. Angles are specified in degrees with 0 degree angle corresponding to the positive horizontal axis (3 o'clock) direction.

Independently of whether *start* is greater than or less than *end*, the arc is drawn in the counter-clockwise direction. Also, if *start* is equal to *end*, a complete ellipse is drawn.

Notice that unlike DrawArc(), this function does not draw the lines to the arc ends, even when using non-transparent brush.

void wxDC::DrawEllipticArc (const wxPoint & *pt,* const wxSize & *sz,* double *sa,* double *ea*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawIcon (const wxIcon & *icon,* wxCoord *x,* wxCoord *y*)

Draw an icon on the display (does nothing if the device context is PostScript).

This can be the simplest way of drawing bitmaps on a window.

void wxDC::DrawIcon (const wxIcon & *icon,* const wxPoint & *pt*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawLabel (const wxString & *text,* const wxBitmap & *bitmap,* const wxRect & *rect,* int *alignment* = wxALIGN_LEFT|wxALIGN_TOP, int *indexAccel* = -1, wxRect * *rectBounding* = NULL)

Draw optional bitmap and the text into the given rectangle and aligns it as specified by alignment parameter; it also will emphasize the character with the given index if it is != -1 and return the bounding rectangle if required.

void wxDC::DrawLabel (const wxString & *text,* const wxRect & *rect,* int *alignment* = wxALIGN_LEFT|wxALIGN_TOP, int *indexAccel* = -1)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawLine (wxCoord *x1,* wxCoord *y1,* wxCoord *x2,* wxCoord *y2*)

Draws a line from the first point to the second.

The current pen is used for drawing the line. Note that the point (*x2*, *y2*) is not part of the line and is not drawn by this function (this is consistent with the behaviour of many other toolkits).

void wxDC::DrawLine (const wxPoint & *pt1,* const wxPoint & *pt2*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawLines (int *n,* const wxPoint *points[],* wxCoord *xoffset* = 0, wxCoord *yoffset* = 0)

Draws lines using an array of points of size *n* adding the optional offset coordinate.

The current pen is used for drawing the lines.

wxPerl Note: Not supported by wxPerl.

void wxDC::DrawLines (const wxPointList * *points,* wxCoord *xoffset* = 0, wxCoord *yoffset* = 0)

This method uses a list of wxPoints, adding the optional offset coordinate.

The programmer is responsible for deleting the list of points.

wxPerl Note: The wxPerl version of this method accepts as its first parameter a reference to an array of wxPoint objects.

void wxDC::DrawPoint (wxCoord *x,* wxCoord *y*)

Draws a point using the color of the current pen.

Note that the other properties of the pen are not used, such as width.

void wxDC::DrawPoint (const wxPoint & *pt*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawPolygon (int *n*, const wxPoint *points[]*, wxCoord *xoffset* = 0, wxCoord *yoffset* = 0, wxPolygonFillMode *fill_style* = wxODDEVEN_RULE)

Draws a filled polygon using an array of points of size *n*, adding the optional offset coordinate.

The first and last points are automatically closed.

The last argument specifies the fill rule: **wxODDEVEN_RULE** (the default) or **wxWINDING_RULE**.

The current pen is used for drawing the outline, and the current brush for filling the shape. Using a transparent brush suppresses filling.

wxPerl Note: Not supported by wxPerl.

void wxDC::DrawPolygon (const wxPointList * *points*, wxCoord *xoffset* = 0, wxCoord *yoffset* = 0, wxPolygonFillMode *fill_style* = wxODDEVEN_RULE)

This method draws a filled polygon using a list of wxPoints, adding the optional offset coordinate.

The first and last points are automatically closed.

The last argument specifies the fill rule: **wxODDEVEN_RULE** (the default) or **wxWINDING_RULE**.

The current pen is used for drawing the outline, and the current brush for filling the shape. Using a transparent brush suppresses filling.

The programmer is responsible for deleting the list of points.

wxPerl Note: The wxPerl version of this method accepts as its first parameter a reference to an array of wxPoint objects.

void wxDC::DrawPolyPolygon (int *n*, const int *count[]*, const wxPoint *points[]*, wxCoord *xoffset* = 0, wxCoord *yoffset* = 0, wxPolygonFillMode *fill_style* = wxODDEVEN_RULE)

Draws two or more filled polygons using an array of *points*, adding the optional offset coordinates.

Notice that for the platforms providing a native implementation of this function (Windows and PostScript-based wxDC currently), this is more efficient than using DrawPolygon() in a loop.

n specifies the number of polygons to draw, the array *count* of size *n* specifies the number of points in each of the polygons in the *points* array.

The last argument specifies the fill rule: **wxODDEVEN_RULE** (the default) or **wxWINDING_RULE**.

The current pen is used for drawing the outline, and the current brush for filling the shape. Using a transparent brush suppresses filling.

The polygons maybe disjoint or overlapping. Each polygon specified in a call to DrawPolyPolygon() must be closed. Unlike polygons created by the DrawPolygon() member function, the polygons created by this method are not closed automatically.

void wxDC::DrawRectangle (wxCoord *x*, wxCoord *y*, wxCoord *width*, wxCoord *height*)

Draws a rectangle with the given top left corner, and with the given size.

The current pen is used for the outline and the current brush for filling the shape.

void wxDC::DrawRectangle (const wxPoint & *pt,* const wxSize & *sz*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawRectangle (const wxRect & *rect*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawRotatedText (const wxString & *text,* wxCoord *x,* wxCoord *y,* double *angle*)

Draws the text rotated by *angle* degrees (positive angles are counterclockwise; the full angle is 360 degrees).

Notice that, as with DrawText(), the *text* can contain multiple lines separated by the new line (`'\n'`) characters.

Note

Under Win9x only TrueType fonts can be drawn by this function. In particular, a font different from `wxNORMAL_FONT` should be used as the latter is not a TrueType font. `wxSWISS_FONT` is an example of a font which is.

See also

DrawText()

void wxDC::DrawRotatedText (const wxString & *text,* const wxPoint & *point,* double *angle*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawRoundedRectangle (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height,* double *radius*)

Draws a rectangle with the given top left corner, and with the given size.

The corners are quarter-circles using the given radius. The current pen is used for the outline and the current brush for filling the shape.

If *radius* is positive, the value is assumed to be the radius of the rounded corner. If *radius* is negative, the absolute value is assumed to be the *proportion* of the smallest dimension of the rectangle. This means that the corner can be a sensible size relative to the size of the rectangle, and also avoids the strange effects X produces when the corners are too big for the rectangle.

void wxDC::DrawRoundedRectangle (const wxPoint & *pt,* const wxSize & *sz,* double *radius*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawRoundedRectangle (const wxRect & *rect,* double *radius*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::DrawSpline (int *n,* const wxPoint *points[]*)

Draws a spline between all given points using the current pen.

wxPerl Note: Not supported by wxPerl.

void wxDC::DrawSpline (const wxPointList * *points*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxPerl Note: The wxPerl version of this method accepts as its first parameter a reference to an array of wxPoint objects.

void wxDC::DrawSpline (wxCoord *x1,* wxCoord *y1,* wxCoord *x2,* wxCoord *y2,* wxCoord *x3,* wxCoord *y3*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxPerl Note: Not supported by wxPerl.

void wxDC::DrawText (const wxString & *text,* wxCoord *x,* wxCoord *y*)

Draws a text string at the specified point, using the current text font, and the current text foreground and background colours.

The coordinates refer to the top-left corner of the rectangle bounding the string. See GetTextExtent() for how to get the dimensions of a text string, which can be used to position the text more precisely and DrawLabel() if you need to align the string differently.

Starting from wxWidgets 2.9.2 *text* parameter can be a multi-line string, i.e. contain new line characters, and will be rendered correctly.

Note

The current logical function is ignored by this function.

void wxDC::DrawText (const wxString & *text,* const wxPoint & *pt*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::EndDoc ( )

Ends a document (only relevant when outputting to a printer).

void wxDC::EndPage ( )

Ends a document page (only relevant when outputting to a printer).

bool wxDC::FloodFill (wxCoord *x,* wxCoord *y,* const wxColour & *colour,* wxFloodFillStyle *style* = wxFLOOD_SURFACE)

Flood fills the device context starting from the given point, using the current brush colour, and using a style:

- wxFLOOD_SURFACE: The flooding occurs until a colour other than the given colour is encountered.
- wxFLOOD_BORDER: The area to be flooded is bounded by the given colour.

Currently this method is not implemented in wxOSX and does nothing there.

Returns

false if the operation failed.

Note

The present implementation for non-Windows platforms may fail to find colour borders if the pixels do not match the colour exactly. However the function will still return true.
This method shouldn't be used with wxPaintDC under non-Windows platforms as it uses GetPixel() internally and this may give wrong results, notably in wxGTK. If you need to flood fill wxPaintDC, create a temporary wxMemoryDC, flood fill it and then blit it to, or draw as a bitmap on, wxPaintDC. See the example of doing this in the drawing sample and wxBufferedPaintDC class.

bool wxDC::FloodFill (const wxPoint & *pt,* const wxColour & *col,* wxFloodFillStyle *style* = wxFLOOD_SURFACE)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxBitmap wxDC::GetAsBitmap (const wxRect * *subrect* = NULL) const

If supported by the platform and the type of DC, fetch the contents of the DC, or a subset of it, as a bitmap.

const wxBrush& wxDC::GetBackground ( ) const

Gets the brush used for painting the background.

See also

wxDC::SetBackground()

int wxDC::GetBackgroundMode ( ) const

Returns the current background mode: wxPENSTYLE_SOLID or wxPENSTYLE_TRANSPARENT.

See also

SetBackgroundMode()

const wxBrush& wxDC::GetBrush ( ) const

Gets the current brush.

See also

wxDC::SetBrush()

wxCoord wxDC::GetCharHeight ( ) const

Gets the character height of the currently set font.

wxCoord wxDC::GetCharWidth ( ) const

Gets the average character width of the currently set font.

void wxDC::GetClippingBox (wxCoord * *x*, wxCoord * *y*, wxCoord * *width*, wxCoord * *height*) const

Gets the rectangle surrounding the current clipping region.

int wxDC::GetDepth ( ) const

Returns the depth (number of bits/pixel) of this DC.

See also

wxDisplayDepth()

wxPoint wxDC::GetDeviceOrigin ( ) const

Returns the current device origin.

See also

SetDeviceOrigin()

const wxFont& wxDC::GetFont ( ) const

Gets the current font.

Notice that even although each device context object has some default font after creation, this method would return a wxNullFont initially and only after calling SetFont() a valid font is returned.

wxFontMetrics wxDC::GetFontMetrics ( ) const

Returns the various font characteristics.

This method allows to retrieve some of the font characteristics not returned by GetTextExtent(), notably internal leading and average character width.

Currently this method returns correct results only under wxMSW, in the other ports the internal leading will always be 0 and the average character width will be computed as the width of the character 'x'.

Since

2.9.2

void* wxDC::GetHandle ( ) const

Returns a value that can be used as a handle to the native drawing context, if this wxDC has something that could be thought of in that way.

(Not all of them do.)

For example, on Windows the return value is an HDC, on OSX it is a CGContextRef and on wxGTK it will be a GdkDrawable. If the DC is a wxGCDC then the return value will be the value returned from wxGraphicsContext::GetNativeContext. A value of NULL is returned if the DC does not have anything that fits the handle concept.

Since

2.9.5

wxLayoutDirection wxDC::GetLayoutDirection ( ) const

Gets the current layout direction of the device context.

On platforms where RTL layout is supported, the return value will either be `wxLayout_LeftToRight` or `wxLayout_RightToLeft`. If RTL layout is not supported, the return value will be `wxLayout_Default`.

See also

SetLayoutDirection()

wxRasterOperationMode wxDC::GetLogicalFunction ( ) const

Gets the current logical function.

See also

SetLogicalFunction()

void wxDC::GetLogicalOrigin (wxCoord * *x*, wxCoord * *y*) const

wxPoint wxDC::GetLogicalOrigin ( ) const

void wxDC::GetLogicalScale (double * *x*, double * *y*) const

wxMappingMode wxDC::GetMapMode ( ) const

Gets the current mapping mode for the device context.

See also

SetMapMode()

void wxDC::GetMultiLineTextExtent (const wxString & *string*, wxCoord * *w*, wxCoord * *h*, wxCoord * *heightLine* = `NULL`, const wxFont * *font* = `NULL`) const

Gets the dimensions of the string using the currently selected font.

string is the text string to measure, *heightLine*, if non NULL, is where to store the height of a single line.

The text extent is set in the given *w* and *h* pointers.

If the optional parameter *font* is specified and valid, then it is used for the text extent calculation, otherwise the currently selected font is used.

Note

This function works with both single-line and multi-line strings.

wxPerl Note: In wxPerl this method is implemented as GetMultiLineTextExtent(string, font = undef) returning a 3-element list (width, height, line_height)

See also

wxFont, SetFont(), GetPartialTextExtents(), GetTextExtent()

wxSize wxDC::GetMultiLineTextExtent (const wxString & *string*) const

Gets the dimensions of the string using the currently selected font.

string is the text string to measure, *heightLine*, if non NULL, is where to store the height of a single line.

Returns

The text extent as a wxSize object.

Note

This function works with both single-line and multi-line strings.

wxPerl Note: Not supported by wxPerl.

See also

wxFont, SetFont(), GetPartialTextExtents(), GetTextExtent()

bool wxDC::GetPartialTextExtents (const wxString & *text,* wxArrayInt & *widths*) const

Fills the *widths* array with the widths from the beginning of *text* to the corresponding character of *text*.

The generic version simply builds a running total of the widths of each character using GetTextExtent(), however if the various platforms have a native API function that is faster or more accurate than the generic implementation then it should be used instead.

wxPerl Note: In wxPerl this method only takes the *text* parameter and returns the widths as a list of integers.

See also

GetMultiLineTextExtent(), GetTextExtent()

const wxPen& wxDC::GetPen ( ) const

Gets the current pen.

See also

SetPen()

bool wxDC::GetPixel (wxCoord *x*, wxCoord *y*, wxColour * *colour*) const

Gets in *colour* the colour at the specified location.

Not available for wxPostScriptDC or wxMetafileDC.

Note

Setting a pixel can be done using DrawPoint().
This method shouldn't be used with wxPaintDC as accessing the DC while drawing can result in unexpected results, notably in wxGTK.

wxSize wxDC::GetPPI ( ) const

Returns the resolution of the device in pixels per inch.

void wxDC::GetSize (wxCoord * *width*, wxCoord * *height*) const

Gets the horizontal and vertical extent of this device context in *device* units.

It can be used to scale graphics to fit the page.

For example, if *maxX* and *maxY* represent the maximum horizontal and vertical 'pixel' values used in your application, the following code will scale the graphic to fit on the printer page:

```
wxCoord w, h;
dc.GetSize(&w, &h);
double scaleX = (double)(maxX / w);
double scaleY = (double)(maxY / h);
dc.SetUserScale(min(scaleX, scaleY),min(scaleX, scaleY));
```

wxPerl Note: In wxPerl there are two methods instead of a single overloaded method:

- GetSize(): returns a Wx::Size object.
- GetSizeWH(): returns a 2-element list (width, height).

wxSize wxDC::GetSize ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::GetSizeMM (wxCoord * *width*, wxCoord * *height*) const

Returns the horizontal and vertical resolution in millimetres.

wxSize wxDC::GetSizeMM ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

const wxColour& wxDC::GetTextBackground ( ) const

Gets the current text background colour.

See also

SetTextBackground()

void wxDC::GetTextExtent (const wxString & *string*, wxCoord * *w*, wxCoord * *h*, wxCoord * *descent* = NULL, wxCoord * *externalLeading* = NULL, const wxFont * *font* = NULL) const

Gets the dimensions of the string using the currently selected font.

string is the text string to measure, *descent* is the dimension from the baseline of the font to the bottom of the descender, and *externalLeading* is any extra vertical space added to the font by the font designer (usually is zero).

The text extent is returned in *w* and *h* pointers or as a wxSize object depending on which version of this function is used.

If the optional parameter *font* is specified and valid, then it is used for the text extent calculation. Otherwise the currently selected font is.

Note

This function only works with single-line strings.

wxPerl Note: In wxPerl this method is implemented as GetTextExtent(string, font = undef) returning a 4-element list (width, height, descent, externalLeading)

See also

wxFont, SetFont(), GetPartialTextExtents(), GetMultiLineTextExtent()

wxSize wxDC::GetTextExtent (const wxString & *string*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxPerl Note: Not supported by wxPerl.

const wxColour& wxDC::GetTextForeground ( ) const

Gets the current text foreground colour.

See also

SetTextForeground()

wxAffineMatrix2D wxDC::GetTransformMatrix ( ) const

Return the transformation matrix used by this device context.

By default the transformation matrix is the identity matrix.

Since

2.9.2

void wxDC::GetUserScale (double * *x*, double * *y*) const

Gets the current user scale factor.

wxPerl Note: In wxPerl this method takes no arguments and return a two element array (x, y).

See also

SetUserScale()

void wxDC::GradientFillConcentric (const wxRect & *rect,* const wxColour & *initialColour,* const wxColour & *destColour*)

Fill the area specified by rect with a radial gradient, starting from *initialColour* at the centre of the circle and fading to *destColour* on the circle outside.

The circle is placed at the centre of *rect*.

Note

Currently this function is very slow, don't use it for real-time drawing.

void wxDC::GradientFillConcentric (const wxRect & *rect,* const wxColour & *initialColour,* const wxColour & *destColour,* const wxPoint & *circleCenter*)

Fill the area specified by rect with a radial gradient, starting from *initialColour* at the centre of the circle and fading to *destColour* on the circle outside.

circleCenter are the relative coordinates of centre of the circle in the specified *rect*.

Note

Currently this function is very slow, don't use it for real-time drawing.

void wxDC::GradientFillLinear (const wxRect & *rect,* const wxColour & *initialColour,* const wxColour & *destColour,* wxDirection *nDirection* = wxRIGHT)

Fill the area specified by *rect* with a linear gradient, starting from *initialColour* and eventually fading to *destColour*.

The *nDirection* specifies the direction of the colour change, default is to use *initialColour* on the left part of the rectangle and *destColour* on the right one.

bool wxDC::IsOk ( ) const

Returns true if the DC is ok to use.

wxCoord wxDC::LogicalToDeviceX (wxCoord *x*) const

Converts logical X coordinate to device coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.

wxCoord wxDC::LogicalToDeviceXRel (wxCoord *x*) const

Converts logical X coordinate to relative device coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.

Use this for converting a width, for example.

wxCoord wxDC::LogicalToDeviceY (wxCoord *y*) const

Converts logical Y coordinate to device coordinate, using the current mapping mode, user scale factor, device origin and axis orientation.

wxCoord wxDC::LogicalToDeviceYRel (wxCoord *y*) const

Converts logical Y coordinate to relative device coordinate, using the current mapping mode and user scale factor but ignoring the axis orientation.

Use this for converting a height, for example.

wxCoord wxDC::MaxX ( ) const

Gets the maximum horizontal extent used in drawing commands so far.

wxCoord wxDC::MaxY ( ) const

Gets the maximum vertical extent used in drawing commands so far.

wxCoord wxDC::MinX ( ) const

Gets the minimum horizontal extent used in drawing commands so far.

wxCoord wxDC::MinY ( ) const

Gets the minimum vertical extent used in drawing commands so far.

void wxDC::ResetBoundingBox ( )

Resets the bounding box: after a call to this function, the bounding box doesn't contain anything.

See also

CalcBoundingBox()

void wxDC::ResetTransformMatrix ( )

Revert the transformation matrix to identity matrix.

Since

2.9.2

void wxDC::SetAxisOrientation (bool *xLeftRight*, bool *yBottomUp*)

Sets the x and y axis orientation (i.e. the direction from lowest to highest values on the axis).

The default orientation is x axis from left to right and y axis from top down.

Parameters

xLeftRight	True to set the x axis orientation to the natural left to right orientation, false to invert it.
yBottomUp	True to set the y axis orientation to the natural bottom up orientation, false to invert it.

void wxDC::SetBackground (const wxBrush & *brush*)

Sets the current background brush for the DC.

void wxDC::SetBackgroundMode (int *mode*)

mode may be one of `wxPENSTYLE_SOLID` and `wxPENSTYLE_TRANSPARENT`.

This setting determines whether text will be drawn with a background colour or not.

void wxDC::SetBrush (const wxBrush & *brush*)

Sets the current brush for the DC.

If the argument is wxNullBrush (or another invalid brush; see wxBrush::IsOk), the current brush is selected out of the device context (leaving wxDC without any valid brush), allowing the current brush to be destroyed safely.

See also

wxBrush, wxMemoryDC (for the interpretation of colours when drawing into a monochrome bitmap)

void wxDC::SetClippingRegion (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Sets the clipping region for this device context to the intersection of the given region described by the parameters of this method and the previously set clipping region.

The clipping region is an area to which drawing is restricted. Possible uses for the clipping region are for clipping text or for speeding up window redraws when only a known area of the screen is damaged.

Notice that you need to call DestroyClippingRegion() if you want to set the clipping region exactly to the region specified.

Also note that if the clipping region is empty, any previously set clipping region is destroyed, i.e. it is equivalent to calling DestroyClippingRegion(), and not to clipping out all drawing on the DC as might be expected.

See also

DestroyClippingRegion(), wxRegion

void wxDC::SetClippingRegion (const wxPoint & *pt,* const wxSize & *sz*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::SetClippingRegion (const wxRect & *rect*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxDC::SetDeviceClippingRegion (const wxRegion & *region*)

Sets the clipping region for this device context.

Unlike SetClippingRegion(), this function works with physical coordinates and not with the logical ones.

void wxDC::SetDeviceOrigin (wxCoord *x,* wxCoord *y*)

Sets the device origin (i.e. the origin in pixels after scaling has been applied).

This function may be useful in Windows printing operations for placing a graphic on a page.

void wxDC::SetFont (const wxFont & *font*)

Sets the current font for the DC.

If the argument is wxNullFont (or another invalid font; see wxFont::IsOk), the current font is selected out of the device context (leaving wxDC without any valid font), allowing the current font to be destroyed safely.

See also

wxFont

void wxDC::SetLayoutDirection (wxLayoutDirection *dir*)

Sets the current layout direction for the device context.

Parameters

dir	May be either `wxLayout_Default`, `wxLayout_LeftToRight` or `wxLayout_RightToLeft`.

See also

GetLayoutDirection()

void wxDC::SetLogicalFunction (wxRasterOperationMode *function*)

Sets the current logical function for the device context.

It determines how a *source* pixel (from a pen or brush colour, or source device context if using Blit()) combines with a *destination* pixel in the current device context. Text drawing is not affected by this function.

See wxRasterOperationMode enumeration values for more info.

The default is `wxCOPY`, which simply draws with the current colour. The others combine the current colour and the background using a logical operation. `wxINVERT` is commonly used for drawing rubber bands or moving outlines, since drawing twice reverts to the original colour.

void wxDC::SetLogicalOrigin (wxCoord *x*, wxCoord *y*)

void wxDC::SetLogicalScale (double *x*, double *y*)

void wxDC::SetMapMode (wxMappingMode *mode*)

The mapping mode of the device context defines the unit of measurement used to convert *logical* units to *device* units.

Note that in X, text drawing isn't handled consistently with the mapping mode; a font is always specified in point size. However, setting the user scale (see SetUserScale()) scales the text appropriately. In Windows, scalable TrueType fonts are always used; in X, results depend on availability of fonts, but usually a reasonable match is found.

The coordinate origin is always at the top left of the screen/printer.

Drawing to a Windows printer device context uses the current mapping mode, but mapping mode is currently ignored for PostScript output.

void wxDC::SetPalette (const wxPalette & *palette*)

If this is a window DC or memory DC, assigns the given palette to the window or bitmap associated with the DC.

If the argument is wxNullPalette, the current palette is selected out of the device context, and the original palette restored.

See also

wxPalette

void wxDC::SetPen (const wxPen & *pen*)

Sets the current pen for the DC.

If the argument is wxNullPen (or another invalid pen; see wxPen::IsOk), the current pen is selected out of the device context (leaving wxDC without any valid pen), allowing the current pen to be destroyed safely.

See also

wxMemoryDC for the interpretation of colours when drawing into a monochrome bitmap.

void wxDC::SetTextBackground (const wxColour & *colour*)

Sets the current text background colour for the DC.

void wxDC::SetTextForeground (const wxColour & *colour*)

Sets the current text foreground colour for the DC.

See also

wxMemoryDC for the interpretation of colours when drawing into a monochrome bitmap.

bool wxDC::SetTransformMatrix (const wxAffineMatrix2D & *matrix*)

Set the transformation matrix.

If transformation matrix is supported on the current system, the specified *matrix* will be used to transform between wxDC and physical coordinates. Otherwise the function returns false and doesn't change the coordinate mapping.

Since

2.9.2

void wxDC::SetUserScale (double *xScale,* double *yScale*)

Sets the user scaling factor, useful for applications which require 'zooming'.

bool wxDC::StartDoc (const wxString & *message*)

Starts a document (only relevant when outputting to a printer).

message is a message to show while printing.

void wxDC::StartPage ( )

Starts a document page (only relevant when outputting to a printer).

bool wxDC::StretchBlit (wxCoord *xdest*, wxCoord *ydest*, wxCoord *dstWidth*, wxCoord *dstHeight*, wxDC * *source*, wxCoord *xsrc*, wxCoord *ysrc*, wxCoord *srcWidth*, wxCoord *srcHeight*, wxRasterOperationMode *logicalFunc* = wxCOPY, bool *useMask* = `false`, wxCoord *xsrcMask* = wxDefaultCoord, wxCoord *ysrcMask* = wxDefaultCoord)

Copy from a source DC to this DC possibly changing the scale.

Unlike Blit(), this method allows to specify different source and destination region sizes, meaning that it can stretch or shrink it while copying. The same can be achieved by changing the scale of the source or target DC but calling this method is simpler and can also be more efficient if the platform provides a native implementation of it.

The meaning of its other parameters is the same as with Blit(), in particular all source coordinates are interpreted using the source DC coordinate system, i.e. are affected by its scale, origin translation and axis direction.

Parameters

xdest	Destination device context x position.
ydest	Destination device context y position.
dstWidth	Width of destination area.
dstHeight	Height of destination area.
source	Source device context.
xsrc	Source device context x position.
ysrc	Source device context y position.
srcWidth	Width of source area to be copied.
srcHeight	Height of source area to be copied.
logicalFunc	Logical function to use, see SetLogicalFunction().
useMask	If true, Blit does a transparent blit using the mask that is associated with the bitmap selected into the source device context. The Windows implementation does the following if MaskBlt cannot be used: 1. Creates a temporary bitmap and copies the destination area into it. 2. Copies the source area into the temporary bitmap using the specified logical function. 3. Sets the masked area in the temporary bitmap to BLACK by ANDing the mask bitmap with the temp bitmap with the foreground colour set to WHITE and the bg colour set to BLACK. 4. Sets the unmasked area in the destination area to BLACK by ANDing the mask bitmap with the destination area with the foreground colour set to BLACK and the background colour set to WHITE. 5. ORs the temporary bitmap with the destination area. 6. Deletes the temporary bitmap. This sequence of operations ensures that the source's transparent area need not be black, and logical functions are supported. **Note:** on Windows, blitting with masks can be speeded up considerably by compiling wx↩ Widgets with the wxUSE_DC_CACHEING option enabled. You can also influence whether MaskBlt or the explicit mask blitting code above is used, by using wxSystemOptions and setting the `no-maskblt` option to 1.

xsrcMask	Source x position on the mask. If both xsrcMask and ysrcMask are wxDefaultCoord, *xsrc* and *ysrc* will be assumed for the mask source position. Currently only implemented on Windows.
ysrcMask	Source y position on the mask. If both xsrcMask and ysrcMask are wxDefaultCoord, *xsrc* and *ysrc* will be assumed for the mask source position. Currently only implemented on Windows.

There is partial support for Blit() in wxPostScriptDC, under X.

See wxMemoryDC for typical usage.

Since

2.9.0

See also

Blit(), wxMemoryDC, wxBitmap, wxMask

21.174 wxDCBrushChanger Class Reference

`#include <wx/dc.h>`

21.174.1 Detailed Description

wxDCBrushChanger is a small helper class for setting a brush on a wxDC and unsetting it automatically in the destructor, restoring the previous one.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxDC::SetBrush(), wxDCFontChanger, wxDCTextColourChanger, wxDCPenChanger, wxDCClipper

Public Member Functions

- wxDCBrushChanger (wxDC &dc, const wxBrush &brush)

  Sets brush on the given dc, storing the old one.
- ~wxDCBrushChanger ()

  Restores the brush originally selected in the DC passed to the ctor.

21.174.2 Constructor & Destructor Documentation

wxDCBrushChanger::wxDCBrushChanger (wxDC & *dc,* const wxBrush & *brush*)

Sets *brush* on the given *dc*, storing the old one.

Parameters

dc	The DC where the brush must be temporary set.
brush	The brush to set.

wxDCBrushChanger::~wxDCBrushChanger ( )

Restores the brush originally selected in the DC passed to the ctor.

21.175 wxDCClipper Class Reference

`#include <wx/dc.h>`

21.175.1 Detailed Description

wxDCClipper is a helper class for setting a clipping region on a wxDC during its lifetime.

An object of wxDCClipper class is typically created on the stack so that it is automatically destroyed when the object goes out of scope. A typical usage example:

```
void MyFunction(wxDC& dc)
{
    wxDCClipper clip(dc, rect);
    // ... drawing functions here are affected by clipping rect ...
}

void OtherFunction()
{
    wxDC dc;
    MyFunction(dc);
    // ... drawing functions here are not affected by clipping rect ...
}
```

Note

Unlike other similar classes such as wxDCFontChanger, wxDCClipper currently doesn't restore the previously active clipping region when it is destroyed but simply resets clipping on the associated wxDC. This may be changed in the future wxWidgets versions but has to be taken into account explicitly in the current one.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxDC::SetClippingRegion(), wxDCFontChanger, wxDCTextColourChanger, wxDCPenChanger, wxDCBrushChanger

Public Member Functions

- ~wxDCClipper ()

  Destroys the clipping region associated with the DC passed to the ctor.

- wxDCClipper (wxDC &dc, const wxRegion &region)

  Sets the clipping region to the specified region/coordinates.

- wxDCClipper (wxDC &dc, const wxRect &rect)

Sets the clipping region to the specified region/coordinates.

- wxDCClipper (wxDC &dc, wxCoord x, wxCoord y, wxCoord w, wxCoord h)

  Sets the clipping region to the specified region/coordinates.

21.175.2 Constructor & Destructor Documentation

wxDCClipper::wxDCClipper (wxDC & *dc,* const wxRegion & *region*)

Sets the clipping region to the specified region/coordinates.

The clipping region is automatically unset when this object is destroyed.

wxDCClipper::wxDCClipper (wxDC & *dc,* const wxRect & *rect*)

Sets the clipping region to the specified region/coordinates.

The clipping region is automatically unset when this object is destroyed.

wxDCClipper::wxDCClipper (wxDC & *dc,* wxCoord *x,* wxCoord *y,* wxCoord *w,* wxCoord *h*)

Sets the clipping region to the specified region/coordinates.

The clipping region is automatically unset when this object is destroyed.

wxDCClipper::~wxDCClipper ( )

Destroys the clipping region associated with the DC passed to the ctor.

21.176 wxDCFontChanger Class Reference

```
#include <wx/dc.h>
```

21.176.1 Detailed Description

wxDCFontChanger is a small helper class for setting a font on a wxDC and unsetting it automatically in the destructor, restoring the previous one.

Since

2.9.0

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxDC::SetFont(), wxDCTextColourChanger, wxDCPenChanger, wxDCBrushChanger, wxDCClipper

Public Member Functions

- wxDCFontChanger (wxDC &dc)

  Trivial constructor not changing anything.
- wxDCFontChanger (wxDC &dc, const wxFont &font)

  Sets font on the given dc, storing the old one.
- void Set (const wxFont &font)

  Set the font to use.
- ~wxDCFontChanger ()

  Restores the font originally selected in the DC passed to the ctor.

21.176.2 Constructor & Destructor Documentation

wxDCFontChanger::wxDCFontChanger (wxDC & *dc*)

Trivial constructor not changing anything.

This constructor is useful if you don't know beforehand if the font needs to be changed or not. It simply creates the object which won't do anything in its destructor unless Set() is called – in which case it would reset the previous font.

Since

2.9.1

wxDCFontChanger::wxDCFontChanger (wxDC & *dc,* const wxFont & *font*)

Sets *font* on the given *dc*, storing the old one.

Parameters

dc	The DC where the font must be temporary set.
font	The font to set.

wxDCFontChanger::~wxDCFontChanger ( )

Restores the font originally selected in the DC passed to the ctor.

21.176.3 Member Function Documentation

void wxDCFontChanger::Set (const wxFont & *font*)

Set the font to use.

This method is meant to be called once only and only on the objects created with the constructor overload not taking wxColour argument and has the same effect as the other constructor, i.e. sets the font to the given *font* and ensures that the old value is restored when this object is destroyed.

21.177 wxDCOverlay Class Reference

`#include <wx/overlay.h>`

21.177.1 Detailed Description

Connects an overlay with a drawing DC.

Library: wxCore

See also

wxOverlay, wxDC

Public Member Functions

- wxDCOverlay (wxOverlay &overlay, wxDC *dc, int x, int y, int width, int height)
  Connects this overlay to the corresponding drawing dc, if the overlay is not initialized yet this call will do so.
- wxDCOverlay (wxOverlay &overlay, wxDC *dc)
  Convenience wrapper that behaves the same using the entire area of the dc.
- virtual ~wxDCOverlay ()
  Removes the connection between the overlay and the dc.
- void Clear ()
  Clears the layer, restoring the state at the last init.

21.177.2 Constructor & Destructor Documentation

wxDCOverlay::wxDCOverlay (wxOverlay & *overlay,* wxDC * *dc,* int *x,* int *y,* int *width,* int *height*)

Connects this overlay to the corresponding drawing dc, if the overlay is not initialized yet this call will do so.

wxDCOverlay::wxDCOverlay (wxOverlay & *overlay,* wxDC * *dc*)

Convenience wrapper that behaves the same using the entire area of the dc.

virtual wxDCOverlay::~wxDCOverlay ( ) [virtual]

Removes the connection between the overlay and the dc.

21.177.3 Member Function Documentation

void wxDCOverlay::Clear ( )

Clears the layer, restoring the state at the last init.

21.178 wxDCPenChanger Class Reference

```
#include <wx/dc.h>
```

21.178.1 Detailed Description

wxDCPenChanger is a small helper class for setting a pen on a wxDC and unsetting it automatically in the destructor, restoring the previous one.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxDC::SetPen(), wxDCFontChanger, wxDCTextColourChanger, wxDCBrushChanger, wxDCClipper

Public Member Functions

- wxDCPenChanger (wxDC &dc, const wxPen &pen)

  Sets pen on the given dc, storing the old one.
- ~wxDCPenChanger ()

  Restores the pen originally selected in the DC passed to the ctor.

21.178.2 Constructor & Destructor Documentation

wxDCPenChanger::wxDCPenChanger (wxDC & *dc,* const wxPen & *pen*)

Sets *pen* on the given *dc*, storing the old one.

Parameters

dc	The DC where the pen must be temporary set.
pen	The pen to set.

wxDCPenChanger::~wxDCPenChanger ( )

Restores the pen originally selected in the DC passed to the ctor.

21.179 wxDCTextColourChanger Class Reference

```
#include <wx/dc.h>
```

21.179.1 Detailed Description

wxDCTextColourChanger is a small helper class for setting a foreground text colour on a wxDC and unsetting it automatically in the destructor, restoring the previous one.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxDC::SetTextForeground(), wxDCFontChanger, wxDCPenChanger, wxDCBrushChanger, wxDCClipper

Public Member Functions

- wxDCTextColourChanger (wxDC &dc)

  Trivial constructor not changing anything.
- wxDCTextColourChanger (wxDC &dc, const wxColour &col)

  Sets col on the given dc, storing the old one.
- void Set (const wxColour &col)

  Set the colour to use.
- ~wxDCTextColourChanger ()

  Restores the colour originally selected in the DC passed to the ctor.

21.179.2 Constructor & Destructor Documentation

wxDCTextColourChanger::wxDCTextColourChanger (wxDC & *dc*)

Trivial constructor not changing anything.

This constructor is useful if you don't know beforehand if the colour needs to be changed or not. It simply creates the object which won't do anything in its destructor unless Set() is called – in which case it would reset the previous colour.

wxDCTextColourChanger::wxDCTextColourChanger (wxDC & *dc,* const wxColour & *col*)

Sets *col* on the given *dc*, storing the old one.

Parameters

dc	The DC where the colour must be temporary set.
col	The colour to set.

wxDCTextColourChanger::~wxDCTextColourChanger ( )

Restores the colour originally selected in the DC passed to the ctor.

21.179.3 Member Function Documentation

void wxDCTextColourChanger::Set (const wxColour & *col*)

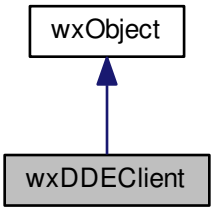
Set the colour to use.

This method is meant to be called once only and only on the objects created with the constructor overload not taking wxColour argument and has the same effect as the other constructor, i.e. sets the colour to the given *col* and ensures that the old value is restored when this object is destroyed.

21.180 wxDDEClient Class Reference

```
#include <wx/dde.h>
```

Inheritance diagram for wxDDEClient:



21.180.1 Detailed Description

A wxDDEClient object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation.

To create a client which can communicate with a suitable server, you need to derive a class from wxDDEConnection and another from wxDDEClient. The custom wxDDEConnection class will intercept communications in a "conversation" with a server, and the custom wxDDEServer is required so that a user-overridden OnMakeConnection() member can return a wxDDEConnection of the required class, when a connection is made.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPClient.

Library: wxBase

Category: Interprocess Communication Availability: only available for the wxMSW port.

See also

wxDDEServer, wxDDEConnection, Interprocess Communication

Public Member Functions

- wxDDEClient ()

  Constructs a client object.
- wxConnectionBase * MakeConnection (const wxString &host, const wxString &service, const wxString &topic)

  Tries to make a connection with a server specified by the host (machine name under UNIX, ignored under Windows), service name (must contain an integer port number under UNIX), and topic string.
- wxConnectionBase * OnMakeConnection ()

  The type of wxDDEConnection returned from a MakeConnection() call can be altered by deriving the OnMake↩ Connection() member to return your own derived connection object.
- bool ValidHost (const wxString &host)

  Returns true if this is a valid host name, false otherwise.

Additional Inherited Members

21.180.2 Constructor & Destructor Documentation

wxDDEClient::wxDDEClient ( )

Constructs a client object.

21.180.3 Member Function Documentation

wxConnectionBase* wxDDEClient::MakeConnection (const wxString & *host,* const wxString & *service,* const wxString & *topic*)

Tries to make a connection with a server specified by the host (machine name under UNIX, ignored under Windows), service name (must contain an integer port number under UNIX), and topic string.

If the server allows a connection, a wxDDEConnection object will be returned.

The type of wxDDEConnection returned can be altered by overriding the OnMakeConnection() member to return your own derived connection object.

wxConnectionBase* wxDDEClient::OnMakeConnection ( )

The type of wxDDEConnection returned from a MakeConnection() call can be altered by deriving the OnMake↩Connection() member to return your own derived connection object.

By default, a wxDDEConnection object is returned.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as wxDDEConnection::OnAdvise(). You may also want to store application-specific data in instances of the new class.

bool wxDDEClient::ValidHost (const wxString & *host*)

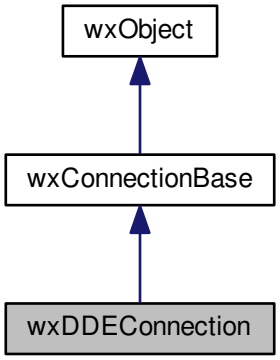
Returns true if this is a valid host name, false otherwise.

This always returns true under MS Windows.

21.181 wxDDEConnection Class Reference

```
#include <wx/dde.h>
```

Inheritance diagram for wxDDEConnection:



21.181.1 Detailed Description

A wxDDEConnection object represents the connection between a client and a server.

It can be created by making a connection using a wxDDEClient object, or by the acceptance of a connection by a wxDDEServer object. The bulk of a DDE (Dynamic Data Exchange) conversation is controlled by calling members in a wxDDEConnection object or by overriding its members.

An application should normally derive a new connection class from wxDDEConnection, in order to override the communication event handlers to do something interesting.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPConnection.

Library: wxBase

Category: Interprocess Communication Availability: only available for the wxMSW port.

See also

wxConnectionBase, wxDDEClient, wxDDEServer, Interprocess Communication

Public Member Functions

- wxDDEConnection ()

  Constructs a connection object.
- wxDDEConnection (void ∗buffer, size_t size)

  Constructs a connection object.
- bool Disconnect ()

  Called by the client or server application to disconnect from the other program; it causes the OnDisconnect() message to be sent to the corresponding connection object in the other program.
- virtual bool OnAdvise (const wxString &topic, const wxString &item, const void ∗data, size_t size, wxIPC↩ Format format)

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

- virtual bool OnDisconnect ()

  Message sent to the client or server application when the other application notifies it to delete the connection.
- virtual bool OnExecute (const wxString &topic, const void *data, size_t size, wxIPCFormat format)

  Message sent to the server application when the client notifies it to execute the given data.
- virtual bool OnPoke (const wxString &topic, const wxString &item, const void *data, size_t size, wxIPCFormat format)

  Message sent to the server application when the client notifies it to accept the given data.
- virtual const void * OnRequest (const wxString &topic, const wxString &item, size_t *size, wxIPCFormat format)

  Message sent to the server application when the client calls Request().
- virtual bool OnStartAdvise (const wxString &topic, const wxString &item)

  Message sent to the server application by the client, when the client wishes to start an "advise loop" for the given topic and item.
- virtual bool OnStopAdvise (const wxString &topic, const wxString &item)

  Message sent to the server application by the client, when the client wishes to stop an "advise loop" for the given topic and item.
- const void * Request (const wxString &item, size_t *size, wxIPCFormat format=wxIPC_TEXT)

  Called by the client application to request data from the server.
- bool StartAdvise (const wxString &item)

  Called by the client application to ask if an advise loop can be started with the server.
- bool StopAdvise (const wxString &item)

  Called by the client application to ask if an advise loop can be stopped.

- bool Advise (const wxString &item, const void *data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const char *data)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const wchar_t *data)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const wxString data)

  Called by the server application to advise the client of a change in the data associated with the given item.

- bool Execute (const void *data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the client application to execute a command on the server.
- bool Execute (const char *data)

  Called by the client application to execute a command on the server.
- bool Execute (const wchar_t *data)

  Called by the client application to execute a command on the server.
- bool Execute (const wxString data)

  Called by the client application to execute a command on the server.

- bool Poke (const wxString &item, const void *data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const char *data)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const wchar_t *data)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const wxString data)

  Called by the client application to poke data into the server.

Additional Inherited Members

21.181.2 Constructor & Destructor Documentation

wxDDEConnection::wxDDEConnection ( )

Constructs a connection object.

If no user-defined connection object is to be derived from wxDDEConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the wxDDEServer::OnAcceptConnection() and/or wxDDEClient::OnMakeConnection() members should be replaced by functions which construct the new connection object.

A default buffer will be associated with this connection.

wxDDEConnection::wxDDEConnection (void * *buffer*, size_t *size*)

Constructs a connection object.

If no user-defined connection object is to be derived from wxDDEConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection. However, if the user defines his or her own derived connection object, the wxDDEServer::OnAcceptConnection() and/or wxDDEClient::OnMakeConnection() members should be replaced by functions which construct the new connection object.

Parameters

buffer	Buffer for this connection object to use in transactions.
size	Size of the buffer given.

21.181.3 Member Function Documentation

bool wxDDEConnection::Advise (const wxString & *item*, const void * *data*, size_t *size*, wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxDDEConnection::Advise (const wxString & *item*, const char * *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxDDEConnection::Advise (const wxString & *item*, const wchar_t * *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxDDEConnection::Advise (const wxString & *item,* const wxString *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns

true if successful.

bool wxDDEConnection::Disconnect ( )

Called by the client or server application to disconnect from the other program; it causes the OnDisconnect() message to be sent to the corresponding connection object in the other program.

The default behaviour of OnDisconnect() is to delete the connection, but the calling application must explicitly delete its side of the connection having called Disconnect().

Returns

true if successful.

bool wxDDEConnection::Execute (const void * *data,* size_t *size,* wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns

true if successful.

bool wxDDEConnection::Execute (const char * *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns

true if successful.

bool wxDDEConnection::Execute (const wchar_t * *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns

true if successful.

bool wxDDEConnection::Execute (const wxString *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns

true if successful.

virtual bool wxDDEConnection::OnAdvise (const wxString & *topic,* const wxString & *item,* const void * *data,* size_t *size,* wxIPCFormat *format*) [virtual]

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

virtual bool wxDDEConnection::OnDisconnect ( ) [virtual]

Message sent to the client or server application when the other application notifies it to delete the connection.

Default behaviour is to delete the connection object.

virtual bool wxDDEConnection::OnExecute (const wxString & *topic,* const void * *data,* size_t *size,* wxIPCFormat *format*) [virtual]

Message sent to the server application when the client notifies it to execute the given data.

Note that there is no item associated with this message.

virtual bool wxDDEConnection::OnPoke (const wxString & *topic,* const wxString & *item,* const void * *data,* size_t *size,* wxIPCFormat *format*) [virtual]

Message sent to the server application when the client notifies it to accept the given data.

virtual const void* wxDDEConnection::OnRequest (const wxString & *topic,* const wxString & *item,* size_t * *size,* wxIPCFormat *format*) [virtual]

Message sent to the server application when the client calls Request().

The server should respond by returning a character string from OnRequest(), or NULL to indicate no data.

virtual bool wxDDEConnection::OnStartAdvise (const wxString & *topic,* const wxString & *item*) [virtual]

Message sent to the server application by the client, when the client wishes to start an "advise loop" for the given topic and item.

The server can refuse to participate by returning false.

virtual bool wxDDEConnection::OnStopAdvise (const wxString & *topic,* const wxString & *item*) [virtual]

Message sent to the server application by the client, when the client wishes to stop an "advise loop" for the given topic and item.

The server can refuse to stop the advise loop by returning false, although this doesn't have much meaning in practice.

bool wxDDEConnection::Poke (const wxString & *item*, const void * *data*, size_t *size*, wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called.

Returns

true if successful.

bool wxDDEConnection::Poke (const wxString & *item*, const char * *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called.

Returns

true if successful.

bool wxDDEConnection::Poke (const wxString & *item*, const wchar_t * *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called.

Returns

true if successful.

bool wxDDEConnection::Poke (const wxString & *item*, const wxString *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called.

Returns

true if successful.

const void* wxDDEConnection::Request (const wxString & *item*, size_t * *size*, wxIPCFormat *format* = wxIPC_TEXT)

Called by the client application to request data from the server.

Causes the server connection's OnRequest() member to be called.

Returns

A character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

bool wxDDEConnection::StartAdvise (const wxString & *item*)

Called by the client application to ask if an advise loop can be started with the server.

Causes the server connection's OnStartAdvise() member to be called.

Returns

true if the server okays it, false otherwise.

bool wxDDEConnection::StopAdvise (const wxString & *item*)

Called by the client application to ask if an advise loop can be stopped.

Causes the server connection's OnStopAdvise() member to be called.

Returns

true if the server okays it, false otherwise.

21.182 wxDDEServer Class Reference

`#include <wx/dde.h>`

21.182.1 Detailed Description

A wxDDEServer object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation.

This DDE-based implementation is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPServer.

Library: wxBase

Category: Interprocess Communication Availability: only available for the wxMSW port.

See also

wxDDEClient, wxDDEConnection, Interprocess Communication

Public Member Functions

- wxDDEServer ()

  Constructs a server object.
- bool Create (const wxString &service)

  Registers the server using the given service name.
- virtual wxConnectionBase * OnAcceptConnection (const wxString &topic)

  When a client calls wxDDEClient::MakeConnection(), the server receives the message and this member is called.

21.182.2 Constructor & Destructor Documentation

wxDDEServer::wxDDEServer ( )

Constructs a server object.

21.182.3 Member Function Documentation

bool wxDDEServer::Create (const wxString & *service*)

Registers the server using the given service name.

Under UNIX, the string must contain an integer id which is used as an Internet port number. false is returned if the call failed (for example, if the port number is already in use).

virtual wxConnectionBase* wxDDEServer::OnAcceptConnection (const wxString & *topic*) `[virtual]`

When a client calls wxDDEClient::MakeConnection(), the server receives the message and this member is called.

The application should derive a member to intercept this message and return a connection object of either the standard wxDDEConnection type, or of a user-derived type.

If the *topic* is "STDIO", the application may wish to refuse the connection. Under UNIX, when a server is created the OnAcceptConnection() message is always sent for standard input and output, but in the context of DDE messages it doesn't make a lot of sense.

21.183 wxDebugContext Class Reference

`#include <wx/memory.h>`

21.183.1 Detailed Description

A class for performing various debugging and memory tracing operations.

Full functionality (such as printing out objects currently allocated) is only present in a debugging build of wxWidgets, i.e. if the **WXDEBUG** symbol is defined. wxDebugContext and related functions and macros can be compiled out by setting wxUSE_DEBUG_CONTEXT to 0 is setup.h

Library: wxBase

Category: Debugging

See also

Debugging

Static Public Member Functions

- static int Check (bool checkAll=false)

  Checks the memory blocks for errors, starting from the currently set checkpoint.
- static bool Dump ()

  Performs a memory dump from the currently set checkpoint, writing to the current debug stream.
- static bool GetCheckPrevious ()

  Returns true if the memory allocator checks all previous memory blocks for errors.
- static bool GetDebugMode ()

  Returns true if debug mode is on.
- static int GetLevel ()

  Gets the debug level (default 1).
- static bool PrintClasses ()

  Prints a list of the classes declared in this application, giving derivation and whether instances of this class can be dynamically created.
- static bool PrintStatistics (bool detailed=true)

  Performs a statistics analysis from the currently set checkpoint, writing to the current debug stream.
- static void SetCheckPrevious (bool check)

  Tells the memory allocator to check all previous memory blocks for errors.
- static void SetCheckpoint (bool all=false)

  Sets the current checkpoint: Dump and PrintStatistics operations will be performed from this point on.

- static void SetDebugMode (bool debug)

  Sets the debug mode on or off.
- static void SetLevel (int level)

  Sets the debug level (default 1).
- static void SetShutdownNotifyFunction (wxShutdownNotifyFunction func)

  Installs a function to be called at the end of wxWidgets shutdown.

21.183.2 Member Function Documentation

static int wxDebugContext::Check (bool *checkAll* = `false`) `[static]`

Checks the memory blocks for errors, starting from the currently set checkpoint.

Returns

Returns the number of errors, so a value of zero represents success. Returns -1 if an error was detected that prevents further checking.

static bool wxDebugContext::Dump ( ) `[static]`

Performs a memory dump from the currently set checkpoint, writing to the current debug stream.

Calls the **Dump** member function for each wxObject derived instance.

Returns

true if the function succeeded, false otherwise.

static bool wxDebugContext::GetCheckPrevious ( ) `[static]`

Returns true if the memory allocator checks all previous memory blocks for errors.

By default, this is false since it slows down execution considerably.

See also

SetCheckPrevious()

static bool wxDebugContext::GetDebugMode ( ) `[static]`

Returns true if debug mode is on.

If debug mode is on, the wxObject new and delete operators store or use information about memory allocation. Otherwise, a straight malloc and free will be performed by these operators.

See also

SetDebugMode()

static int wxDebugContext::GetLevel ( ) [static]

Gets the debug level (default 1).

The debug level is used by the wxTraceLevel function and the WXTRACELEVEL macro to specify how detailed the trace information is; setting a different level will only have an effect if trace statements in the application specify a value other than one.

Deprecated This is obsolete, replaced by wxLog functionality.

See also

SetLevel()

static bool wxDebugContext::PrintClasses ( ) [static]

Prints a list of the classes declared in this application, giving derivation and whether instances of this class can be dynamically created.

See also

PrintStatistics()

static bool wxDebugContext::PrintStatistics (bool *detailed* = true) [static]

Performs a statistics analysis from the currently set checkpoint, writing to the current debug stream.

The number of object and non-object allocations is printed, together with the total size.

Parameters

detailed	If true, the function will also print how many objects of each class have been allocated, and the space taken by these class instances.

See also

PrintStatistics()

static void wxDebugContext::SetCheckpoint (bool *all* = false) [static]

Sets the current checkpoint: Dump and PrintStatistics operations will be performed from this point on.

This allows you to ignore allocations that have been performed up to this point.

Parameters

all	If true, the checkpoint is reset to include all memory allocations since the program started.

static void wxDebugContext::SetCheckPrevious (bool *check*) [static]

Tells the memory allocator to check all previous memory blocks for errors.

By default, this is false since it slows down execution considerably.

See also

GetCheckPrevious()

static void wxDebugContext::SetDebugMode (bool *debug*) `[static]`

Sets the debug mode on or off.

If debug mode is on, the wxObject new and delete operators store or use information about memory allocation. Otherwise, a straight malloc and free will be performed by these operators.

By default, debug mode is on if **WXDEBUG** is defined. If the application uses this function, it should make sure that all object memory allocated is deallocated with the same value of debug mode. Otherwise, the delete operator might try to look for memory information that does not exist.

See also

GetDebugMode()

static void wxDebugContext::SetLevel (int *level*) `[static]`

Sets the debug level (default 1).

The debug level is used by the wxTraceLevel function and the WXTRACELEVEL macro to specify how detailed the trace information is; setting a different level will only have an effect if trace statements in the application specify a value other than one.

Deprecated This is obsolete, replaced by wxLog functionality.

See also

GetLevel()

static void wxDebugContext::SetShutdownNotifyFunction (wxShutdownNotifyFunction *func*) `[static]`

Installs a function to be called at the end of wxWidgets shutdown.

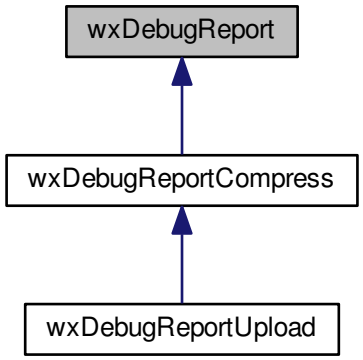
It will be called after all files with global instances of wxDebugContextDumpDelayCounter have run their destructors.

The shutdown function must be take no parameters and return nothing.

21.184 wxDebugReport Class Reference

`#include <wx/debugrpt.h>`

Inheritance diagram for wxDebugReport:



21.184.1 Detailed Description

wxDebugReport is used to generate a debug report, containing information about the program current state.

It is usually used from wxApp::OnFatalException() as shown in the Debug Reporter Sample.

A wxDebugReport object contains one or more files. A few of them can be created by the class itself but more can be created from the outside and then added to the report. Also note that several virtual functions may be overridden to further customize the class behaviour.

Once a report is fully assembled, it can simply be left in the temporary directory so that the user can email it to the developers (in which case you should still use wxDebugReportCompress to compress it in a single file) or uploaded to a Web server using wxDebugReportUpload (setting up the Web server to accept uploads is your responsibility, of course). Other handlers, for example for automatically emailing the report, can be defined as well but are not currently included in wxWidgets.

A typical usage example:

```
wxDebugReport report;
wxDebugReportPreviewStd preview;

report.AddCurrentContext();  // could also use AddAll()
report.AddCurrentDump();     // to do both at once

if ( preview.Show(report) )
    report.Process();
```

Library: wxQA

Category: Debugging

Public Types

- enum Context {
  Context_Current,
  Context_Exception }

  This enum is used for functions that report either the current state or the state during the last (fatal) exception.

Public Member Functions

- wxDebugReport ()

  The constructor creates a temporary directory where the files that will be included in the report are created.
- virtual ~wxDebugReport ()

  The destructor normally destroys the temporary directory created in the constructor with all the files it contains.
- void AddAll (Context context=Context_Exception)

  Adds all available information to the report.
- virtual bool AddContext (Context ctx)

  Add an XML file containing the current or exception context and the stack trace.
- bool AddCurrentContext ()

  The same as calling AddContext(Context_Current).
- bool AddCurrentDump ()

  The same as calling AddDump(Context_Current).
- virtual bool AddDump (Context ctx)

  Adds the minidump file to the debug report.
- bool AddExceptionContext ()

  The same as calling AddContext(Context_Exception).
- bool AddExceptionDump ()

  The same as calling AddDump(Context_Exception).
- virtual void AddFile (const wxString &filename, const wxString &description)

  Add another file to the report.
- bool AddText (const wxString &filename, const wxString &text, const wxString &description)

  This is a convenient wrapper around AddFile().
- const wxString & GetDirectory () const

  This method should be used to construct the full name of the files which you wish to add to the report using AddFile().
- bool GetFile (size_t n, wxString *name, wxString *desc) const

  Retrieves the name (relative to GetDirectory()) and the description of the file with the given index.
- size_t GetFilesCount () const

  Gets the current number files in this report.
- virtual wxString GetReportName () const

  Gets the name used as a base name for various files, by default wxApp::GetAppName() is used.
- bool IsOk () const

  Returns true if the object was successfully initialized.
- bool Process ()

  Processes this report: the base class simply notifies the user that the report has been generated.
- void RemoveFile (const wxString &name)

  Removes the file from report: this is used by wxDebugReportPreview to allow the user to remove files potentially containing private information from the report.
- void Reset ()

  Resets the directory name we use.

Protected Member Functions

- virtual void DoAddCustomContext (wxXmlNode *nodeRoot)

  This function may be overridden to add arbitrary custom context to the XML context file created by AddContext().
- virtual bool DoAddExceptionInfo (wxXmlNode *nodeContext)

  This function may be overridden to modify the contents of the exception tag in the XML context file.
- virtual bool DoAddLoadedModules (wxXmlNode *nodeModules)

  This function may be overridden to modify the contents of the modules tag in the XML context file.
- virtual bool DoAddSystemInfo (wxXmlNode *nodeSystemInfo)

  This function may be overridden to modify the contents of the system tag in the XML context file.

21.184.2 Member Enumeration Documentation

enum wxDebugReport::Context

This enum is used for functions that report either the current state or the state during the last (fatal) exception.

Enumerator

Context_Current

Context_Exception

21.184.3 Constructor & Destructor Documentation

wxDebugReport::wxDebugReport ( )

The constructor creates a temporary directory where the files that will be included in the report are created.

Use IsOk() to check for errors.

virtual wxDebugReport::~wxDebugReport ( ) [virtual]

The destructor normally destroys the temporary directory created in the constructor with all the files it contains.

Call Reset() to prevent this from happening.

21.184.4 Member Function Documentation

void wxDebugReport::AddAll (Context *context* = Context_Exception)

Adds all available information to the report.

Currently this includes a text (XML) file describing the process context and, under Win32, a minidump file.

virtual bool wxDebugReport::AddContext (Context *ctx*) [virtual]

Add an XML file containing the current or exception context and the stack trace.

bool wxDebugReport::AddCurrentContext ( )

The same as calling AddContext(Context_Current).

bool wxDebugReport::AddCurrentDump ( )

The same as calling AddDump(Context_Current).

virtual bool wxDebugReport::AddDump (Context *ctx*) [virtual]

Adds the minidump file to the debug report.

Minidumps are only available under recent Win32 versions (`dbghlp32.dll` can be installed under older systems to make minidumps available).

bool wxDebugReport::AddExceptionContext ( )

The same as calling AddContext(Context_Exception).

bool wxDebugReport::AddExceptionDump ( )

The same as calling AddDump(Context_Exception).

virtual void wxDebugReport::AddFile (const wxString & *filename,* const wxString & *description*) `[virtual]`

Add another file to the report.

If *filename* is an absolute path, it is copied to a file in the debug report directory with the same name. Otherwise the file will be searched in the temporary directory returned by GetDirectory().

The argument *description* only exists to be displayed to the user in the report summary shown by wxDebugReportPreview.

See also

GetDirectory(), AddText()

bool wxDebugReport::AddText (const wxString & *filename,* const wxString & *text,* const wxString & *description*)

This is a convenient wrapper around AddFile().

It creates the file with the given *name* and writes *text* to it, then adds the file to the report. The *filename* shouldn't contain the path.

Returns

true if file could be added successfully, false if an IO error occurred.

virtual void wxDebugReport::DoAddCustomContext (wxXmlNode * *nodeRoot*) `[protected], [virtual]`

This function may be overridden to add arbitrary custom context to the XML context file created by AddContext().

By default, it does nothing.

virtual bool wxDebugReport::DoAddExceptionInfo (wxXmlNode * *nodeContext*) `[protected], [virtual]`

This function may be overridden to modify the contents of the exception tag in the XML context file.

virtual bool wxDebugReport::DoAddLoadedModules (wxXmlNode * *nodeModules*) `[protected], [virtual]`

This function may be overridden to modify the contents of the modules tag in the XML context file.

virtual bool wxDebugReport::DoAddSystemInfo (wxXmlNode * *nodeSystemInfo*) `[protected], [virtual]`

This function may be overridden to modify the contents of the system tag in the XML context file.

const wxString& wxDebugReport::GetDirectory ( ) const

This method should be used to construct the full name of the files which you wish to add to the report using AddFile().

Returns

The name of the temporary directory used for the files in this report.

bool wxDebugReport::GetFile (size_t *n*, wxString * *name*, wxString * *desc*) const

Retrieves the name (relative to GetDirectory()) and the description of the file with the given index.

If *n* is greater than or equal to the number of files, then false is returned.

size_t wxDebugReport::GetFilesCount ( ) const

Gets the current number files in this report.

virtual wxString wxDebugReport::GetReportName ( ) const `[virtual]`

Gets the name used as a base name for various files, by default wxApp::GetAppName() is used.

bool wxDebugReport::IsOk ( ) const

Returns true if the object was successfully initialized.

If this method returns false the report can't be used.

bool wxDebugReport::Process ( )

Processes this report: the base class simply notifies the user that the report has been generated.

This is usually not enough – instead you should override this method to do something more useful to you.

void wxDebugReport::RemoveFile (const wxString & *name*)

Removes the file from report: this is used by wxDebugReportPreview to allow the user to remove files potentially containing private information from the report.

void wxDebugReport::Reset ( )

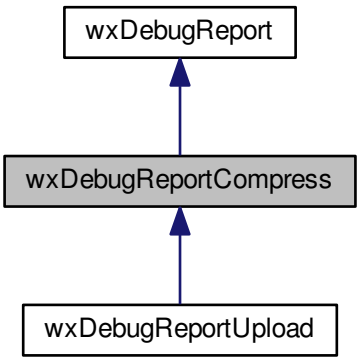
Resets the directory name we use.

The object can't be used any more after this as it becomes uninitialized and invalid.

21.185 wxDebugReportCompress Class Reference

`#include <wx/debugrpt.h>`

Inheritance diagram for wxDebugReportCompress:



21.185.1 Detailed Description

wxDebugReportCompress is a wxDebugReport which compresses all the files in this debug report into a single ZIP file in its wxDebugReport::Process() function.

Library: wxQA

Category: Debugging

Public Member Functions

- wxDebugReportCompress ()

  Default constructor does nothing special.
- void SetCompressedFileDirectory (const wxString &dir)

  Set the directory where the debug report should be generated.
- void SetCompressedFileBaseName (const wxString &name)

  Set the base name of the generated debug report file.
- const wxString & GetCompressedFileName () const

  Returns the full path of the compressed file (empty if creation failed).

Additional Inherited Members

21.185.2 Constructor & Destructor Documentation

wxDebugReportCompress::wxDebugReportCompress ( )

Default constructor does nothing special.

21.185.3 Member Function Documentation

const wxString& wxDebugReportCompress::GetCompressedFileName ( ) const

Returns the full path of the compressed file (empty if creation failed).

void wxDebugReportCompress::SetCompressedFileBaseName (const wxString & *name*)

Set the base name of the generated debug report file.

This function is similar to SetCompressedFileDirectory() but allows to change the base name of the file. Notice that the file extension will always be .zip.

By default, a unique name constructed from wxApp::GetAppName(), the current process id and the current date and time is used.

Parameters

name	The base name (i.e. without extension) of the file.

Since

2.9.1

void wxDebugReportCompress::SetCompressedFileDirectory (const wxString & *dir*)

Set the directory where the debug report should be generated.

By default, the debug report is generated under user temporary files directory. This is usually fine if it is meant to be processed in some way (e.g. automatically uploaded to a remote server) but if the user is asked to manually upload or send the report, it may be more convenient to generate it in e.g. the users home directory and this function allows to do this.

Notice that it should be called before wxDebugReport::Process() or it has no effect.

Parameters

dir	The full path to an existing directory where the debug report file should be generated.

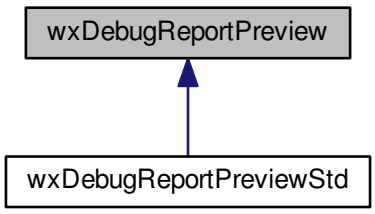
Since

2.9.1

21.186 wxDebugReportPreview Class Reference

`#include <wx/debugrpt.h>`

Inheritance diagram for wxDebugReportPreview:



21.186.1 Detailed Description

This class presents the debug report to the user and allows him to veto report entirely or remove some parts of it.

Although not mandatory, using this class is strongly recommended as data included in the debug report might contain sensitive private information and the user should be notified about it as well as having a possibility to examine the data which had been gathered to check whether this is effectively the case and discard the debug report if it is.

wxDebugReportPreview is an abstract base class, currently the only concrete class deriving from it is wxDebugReportPreviewStd.

Library: wxQA

Category: Debugging

Public Member Functions

- wxDebugReportPreview ()

  Default constructor.

- virtual ~wxDebugReportPreview ()

  Destructor is trivial as well but should be virtual for a base class.

- virtual bool Show (wxDebugReport &dbgrpt) const =0

  Present the report to the user and allow him to modify it by removing some or all of the files and, potentially, adding some notes.

21.186.2 Constructor & Destructor Documentation

wxDebugReportPreview::wxDebugReportPreview ( )

Default constructor.

virtual wxDebugReportPreview::~wxDebugReportPreview ( ) [virtual]

Destructor is trivial as well but should be virtual for a base class.

21.186.3 Member Function Documentation

virtual bool wxDebugReportPreview::Show (wxDebugReport & *dbgrpt*) const [pure virtual]

Present the report to the user and allow him to modify it by removing some or all of the files and, potentially, adding some notes.

Returns

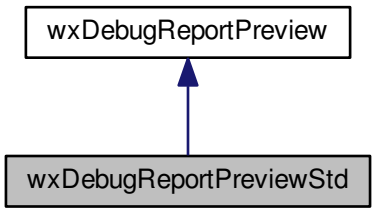
true if the report should be processed or false if the user chose to cancel report generation or removed all files from it.

Implemented in wxDebugReportPreviewStd.

21.187 wxDebugReportPreviewStd Class Reference

`#include <wx/debugrpt.h>`

Inheritance diagram for wxDebugReportPreviewStd:



21.187.1 Detailed Description

wxDebugReportPreviewStd is a standard debug report preview window.

It displays a dialog allowing the user to examine the contents of a debug report, remove files from and add notes to it.

Library: wxQA

Category: Debugging

Public Member Functions

- wxDebugReportPreviewStd ()

  Trivial default constructor.
- bool Show (wxDebugReport &dbgrpt) const

  Shows the dialog.

21.187.2 Constructor & Destructor Documentation

wxDebugReportPreviewStd::wxDebugReportPreviewStd ( )

Trivial default constructor.

21.187.3 Member Function Documentation

bool wxDebugReportPreviewStd::Show (wxDebugReport & *dbgrpt*) const `[virtual]`

Shows the dialog.

See also

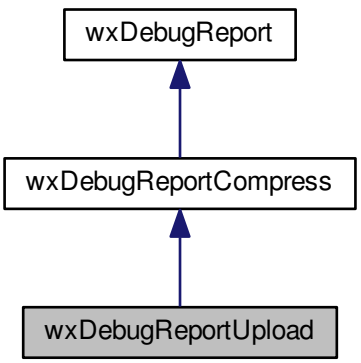
wxDebugReportPreview::Show()

Implements wxDebugReportPreview.

21.188 wxDebugReportUpload Class Reference

`#include <wx/debugrpt.h>`

Inheritance diagram for wxDebugReportUpload:



21.188.1 Detailed Description

This class is used to upload a compressed file using HTTP POST request.

As this class derives from wxDebugReportCompress, before upload the report is compressed in a single ZIP file.

Library: wxQA

Category: Debugging

Public Member Functions

- wxDebugReportUpload (const wxString &url, const wxString &input, const wxString &action, const wxString &curl="curl")

  This class will upload the compressed file created by its base class to an HTML multipart/form-data form at the specified address.

Protected Member Functions

- virtual bool OnServerReply (const wxArrayString &reply)

  This function may be overridden in a derived class to show the output from curl: this may be an HTML page or anything else that the server returned.

Additional Inherited Members

21.188.2 Constructor & Destructor Documentation

wxDebugReportUpload::wxDebugReportUpload (const wxString & *url,* const wxString & *input,* const wxString & *action,* const wxString & *curl* = `"curl"`)

This class will upload the compressed file created by its base class to an HTML multipart/form-data form at the specified address.

The *url* is the upload page address, *input* is the name of the `"type=file"` control on the form used for the file name and *action* is the value of the form action field. The report is uploaded using the *curl* program which should be available, the *curl* parameter may be used to specify the full path to it.

21.188.3 Member Function Documentation

virtual bool wxDebugReportUpload::OnServerReply (const wxArrayString & *reply*) `[protected], [virtual]`

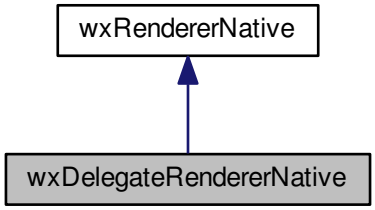
This function may be overridden in a derived class to show the output from curl: this may be an HTML page or anything else that the server returned.

Value returned by this function becomes the return value of wxDebugReport::Process().

21.189 wxDelegateRendererNative Class Reference

`#include <wx/renderer.h>`

Inheritance diagram for wxDelegateRendererNative:



21.189.1 Detailed Description

wxDelegateRendererNative allows reuse of renderers code by forwarding all the wxRendererNative methods to the given object and thus allowing you to only modify some of its methods – without having to reimplement all of them.

Note that the "normal", inheritance-based approach, doesn't work with the renderers as it is impossible to derive from a class unknown at compile-time and the renderer is only chosen at run-time. So suppose that you want to only add something to the drawing of the tree control buttons but leave all the other methods unchanged – the only way to do it, considering that the renderer class which you want to customize might not even be written yet when you write your code (it could be written later and loaded from a DLL during run-time), is by using this class.

Except for the constructor, it has exactly the same methods as wxRendererNative and their implementation is trivial: they are simply forwarded to the real renderer. Note that the "real" renderer may, in turn, be a wxDelegate↩ RendererNative as well and that there may be arbitrarily many levels like this – but at the end of the chain there must be a real renderer which does the drawing.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxRendererNative

Public Member Functions

- wxDelegateRendererNative ()

  The default constructor does the same thing as the other one except that it uses the generic renderer instead of the user-specified rendererNative.
- wxDelegateRendererNative (wxRendererNative &rendererNative)

  This constructor uses the user-specified rendererNative to set up the delegate renderer object to follow all calls to the specified real renderer.
- virtual int DrawHeaderButton (wxWindow ∗win, wxDC &dc, const wxRect &rect, int flags=0, wxHeaderSort↩ IconType sortArrow=wxHDR_SORT_ICON_NONE, wxHeaderButtonParams ∗params=NULL)

  Draw the header control button (used, for example, by wxListCtrl).
- virtual int DrawHeaderButtonContents (wxWindow ∗win, wxDC &dc, const wxRect &rect, int flags=0, wx↩ HeaderSortIconType sortArrow=wxHDR_SORT_ICON_NONE, wxHeaderButtonParams ∗params=NULL)

  Draw the contents of a header control button (label, sort arrows, etc.).

- virtual int GetHeaderButtonHeight (wxWindow *win)

  Returns the height of a header button, either a fixed platform height if available, or a generic height based on the win window's font.
- virtual int GetHeaderButtonMargin (wxWindow *win)

  Returns the horizontal margin on the left and right sides of header button's label.
- virtual void DrawTreeItemButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw the expanded/collapsed icon for a tree control item.
- virtual void DrawSplitterBorder (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw the border for sash window: this border must be such that the sash drawn by DrawSplitterSash() blends into it well.
- virtual void DrawSplitterSash (wxWindow *win, wxDC &dc, const wxSize &size, wxCoord position, wxOrientation orient, int flags=0)

  Draw a sash.
- virtual void DrawComboBoxDropButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw a button like the one used by wxComboBox to show a drop down window.
- virtual void DrawDropArrow (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw a drop down arrow that is suitable for use outside a combo box.
- virtual void DrawCheckBox (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw a check box.
- virtual wxSize GetCheckBoxSize (wxWindow *win)

  Returns the size of a check box.
- virtual void DrawPushButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw a blank push button that looks very similar to wxButton.
- virtual void DrawItemSelectionRect (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw a selection rectangle underneath the text as used e.g.
- virtual void DrawFocusRect (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)

  Draw a focus rectangle using the specified rectangle.
- virtual wxSplitterRenderParams GetSplitterParams (const wxWindow *win)

  Get the splitter parameters, see wxSplitterRenderParams.
- virtual wxRendererVersion GetVersion () const

  This function is used for version checking: Load() refuses to load any shared libraries implementing an older or incompatible version.

Additional Inherited Members

21.189.2 Constructor & Destructor Documentation

wxDelegateRendererNative::wxDelegateRendererNative ( )

The default constructor does the same thing as the other one except that it uses the generic renderer instead of the user-specified *rendererNative*.

In any case, this sets up the delegate renderer object to follow all calls to the specified real renderer.

wxDelegateRendererNative::wxDelegateRendererNative (**wxRendererNative** & *rendererNative*)

This constructor uses the user-specified *rendererNative* to set up the delegate renderer object to follow all calls to the specified real renderer.

Note

This object does not take ownership of (i.e. won't delete) *rendererNative*.

21.189.3 Member Function Documentation

virtual void wxDelegateRendererNative::DrawCheckBox (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [virtual]

Draw a check box.

flags may have the `wxCONTROL_CHECKED`, `wxCONTROL_CURRENT` or `wxCONTROL_UNDETERMINED` bit set, see wxCONTROL_FLAGS.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawComboBoxDropButton (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [virtual]

Draw a button like the one used by wxComboBox to show a drop down window.

The usual appearance is a downwards pointing arrow.

flags may have the `wxCONTROL_PRESSED` or `wxCONTROL_CURRENT` bit set, see wxCONTROL_FLAGS.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawDropArrow (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [virtual]

Draw a drop down arrow that is suitable for use outside a combo box.

Arrow will have transparent background.

rect is not entirely filled by the arrow. Instead, you should use bounding rectangle of a drop down button which arrow matches the size you need.

flags may have the `wxCONTROL_PRESSED` or `wxCONTROL_CURRENT` bit set, see wxCONTROL_FLAGS.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawFocusRect (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [virtual]

Draw a focus rectangle using the specified rectangle.

wxListCtrl.

The only supported flags is `wxCONTROL_SELECTED` for items which are selected. see wxCONTROL_FLAGS.

Implements wxRendererNative.

virtual int wxDelegateRendererNative::DrawHeaderButton (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0, wxHeaderSortIconType *sortArrow* = wxHDR_SORT_ICON_NONE, wxHeaderButtonParams * *params* = NULL) [virtual]

Draw the header control button (used, for example, by wxListCtrl).

Depending on platforms the *flags* parameter may support the `wxCONTROL_SELECTED` `wxCONTROL_DISABLED` and `wxCONTROL_CURRENT` bits, see wxCONTROL_FLAGS.

Returns

The optimal width to contain the unabbreviated label text or bitmap, the sort arrow if present, and internal margins.

Implements wxRendererNative.

virtual int wxDelegateRendererNative::DrawHeaderButtonContents (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = 0, wxHeaderSortIconType *sortArrow* = wxHDR_SORT_ICON_NONE, wxHeaderButtonParams * *params* = NULL) [virtual]

Draw the contents of a header control button (label, sort arrows, etc.).

This function is normally only called by DrawHeaderButton().

Depending on platforms the *flags* parameter may support the wxCONTROL_SELECTED wxCONTROL_DISABLED and wxCONTROL_CURRENT bits, see wxCONTROL_FLAGS.

Returns

The optimal width to contain the unabbreviated label text or bitmap, the sort arrow if present, and internal margins.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawItemSelectionRect (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = 0) [virtual]

Draw a selection rectangle underneath the text as used e.g.

in a wxListCtrl.

The supported *flags* are wxCONTROL_SELECTED for items which are selected (e.g. often a blue rectangle) and wxCONTROL_CURRENT for the item that has the focus (often a dotted line around the item's text). wxCONTROL_FOCUSED may be used to indicate if the control has the focus (otherwise the selection rectangle is e.g. often grey and not blue). This may be ignored by the renderer or deduced by the code directly from the *win*.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawPushButton (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = 0) [virtual]

Draw a blank push button that looks very similar to wxButton.

flags may have the wxCONTROL_PRESSED, wxCONTROL_CURRENT or wxCONTROL_ISDEFAULT bit set, see wxCONTROL_FLAGS.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawSplitterBorder (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = 0) [virtual]

Draw the border for sash window: this border must be such that the sash drawn by DrawSplitterSash() blends into it well.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawSplitterSash (wxWindow * *win,* wxDC & *dc,* const wxSize & *size,* wxCoord *position,* wxOrientation *orient,* int *flags* = 0) [virtual]

Draw a sash.

The *orient* parameter defines whether the sash should be vertical or horizontal and how the *position* should be interpreted.

Implements wxRendererNative.

virtual void wxDelegateRendererNative::DrawTreeItemButton (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [virtual]

Draw the expanded/collapsed icon for a tree control item.

To draw an expanded button the *flags* parameter must contain `wxCONTROL_EXPANDED` bit, see wxCONTROL_FLAGS.

Implements wxRendererNative.

virtual wxSize wxDelegateRendererNative::GetCheckBoxSize (wxWindow * *win*) [virtual]

Returns the size of a check box.

The *win* parameter is not used currently and can be NULL.

Implements wxRendererNative.

virtual int wxDelegateRendererNative::GetHeaderButtonHeight (wxWindow * *win*) [virtual]

Returns the height of a header button, either a fixed platform height if available, or a generic height based on the *win* window's font.

Implements wxRendererNative.

virtual int wxDelegateRendererNative::GetHeaderButtonMargin (wxWindow * *win*) [virtual]

Returns the horizontal margin on the left and right sides of header button's label.

Since

2.9.2

Implements wxRendererNative.

virtual wxSplitterRenderParams wxDelegateRendererNative::GetSplitterParams (const wxWindow * *win*) [virtual]

Get the splitter parameters, see wxSplitterRenderParams.

The *win* parameter should be a wxSplitterWindow.

Implements wxRendererNative.

virtual wxRendererVersion wxDelegateRendererNative::GetVersion ( ) const [virtual]

This function is used for version checking: Load() refuses to load any shared libraries implementing an older or incompatible version.

Remarks

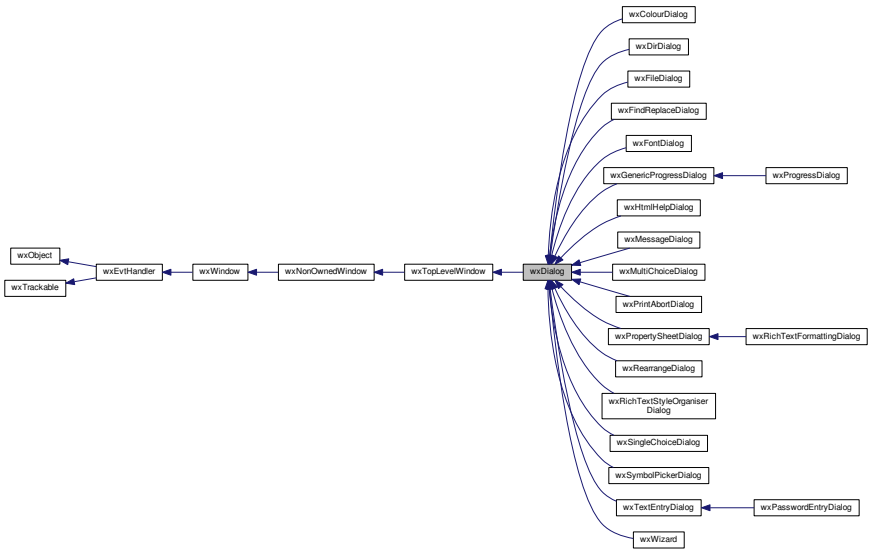
The implementation of this method is always the same in all renderers (simply construct wxRendererVersion using the `wxRendererVersion::Current_XXX` values), but it has to be in the derived, not base, class, to detect mismatches between the renderers versions and so you have to implement it anew in all renderers.

Implements wxRendererNative.

21.190 wxDialog Class Reference

`#include <wx/dialog.h>`

Inheritance diagram for wxDialog:



21.190.1 Detailed Description

A dialog box is a window with a title bar and sometimes a system menu, which can be moved around the screen.

It can contain controls and other windows and is often used to allow the user to make some choice or to answer a question.

Dialogs can be made scrollable, automatically, for computers with low resolution screens: please see Automatic Scrolled Dialogs for further details.

Dialogs usually contain either a single button allowing to close the dialog or two buttons, one accepting the changes and the other one discarding them (such button, if present, is automatically activated if the user presses the "Esc" key). By default, buttons with the standard wxID_OK and wxID_CANCEL identifiers behave as expected. Starting with wxWidgets 2.7 it is also possible to use a button with a different identifier instead, see SetAffirmativeId() and SetEscapeId().

Also notice that the CreateButtonSizer() should be used to create the buttons appropriate for the current platform and positioned correctly (including their order which is platform-dependent).

21.190.2 Modal and Modeless

There are two kinds of dialog, modal and modeless. A modal dialog blocks program flow and user input on other windows until it is dismissed, whereas a modeless dialog behaves more like a frame in that program flow continues, and input in other windows is still possible. To show a modal dialog you should use the ShowModal() method while to show a dialog modelessly you simply use Show(), just as with frames.

Note that the modal dialog is one of the very few examples of wxWindow-derived objects which may be created on the stack and not on the heap. In other words, while most windows would be created like this:

```
void AskUser()
{
    MyAskDialog *dlg = new MyAskDialog(...);
    if ( dlg->ShowModal() == wxID_OK )
```

```
        // ...
    //else: dialog was cancelled or some another button pressed

    dlg->Destroy();
}
```

You can achieve the same result with dialogs by using simpler code:

```
void AskUser()
{
    MyAskDialog dlg(...);
    if ( dlg.ShowModal() == wxID_OK )
        // ...

    // no need to call Destroy() here
}
```

An application can define a wxCloseEvent handler for the dialog to respond to system close events.

Styles

This class supports the following styles:

- wxCAPTION: Puts a caption on the dialog box.
- wxDEFAULT_DIALOG_STYLE: Equivalent to a combination of wxCAPTION, wxCLOSE_BOX and wxSYS↩TEM_MENU (the last one is not used under Unix).
- wxRESIZE_BORDER: Display a resizable frame around the window.
- wxSYSTEM_MENU: Display a system menu.
- wxCLOSE_BOX: Displays a close box on the frame.
- wxMAXIMIZE_BOX: Displays a maximize box on the dialog.
- wxMINIMIZE_BOX: Displays a minimize box on the dialog.
- wxTHICK_FRAME: Display a thick frame around the window.
- wxSTAY_ON_TOP: The dialog stays on top of all other windows.
- wxNO_3D: This style is obsolete and doesn't do anything any more, don't use it in any new code.
- wxDIALOG_NO_PARENT: By default, a dialog created with a NULL parent window will be given the application's top level window as parent. Use this style to prevent this from happening and create an orphan dialog. This is not recommended for modal dialogs.
- wxDIALOG_EX_CONTEXTHELP: Under Windows, puts a query button on the caption. When pressed, Windows will go into a context-sensitive help mode and wxWidgets will send a `wxEVT_HELP` event if the user clicked on an application window. Note that this is an extended style and must be set by calling SetExtraStyle() before Create is called (two-step construction).
- wxDIALOG_EX_METAL: On Mac OS X, frames with this style will be shown with a metallic look. This is an extra style.

Under Unix or Linux, MWM (the Motif Window Manager) or other window managers recognizing the MHM hints should be running for any of these styles to have an effect.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCloseEvent& event)

Event macros for events emitted by this class:

- EVT_CLOSE(func): The dialog is being closed by the user or programmatically (see wxWindow::Close). The user may generate this event clicking the close button (typically the 'X' on the top-right of the title bar) if it's present (see the `wxCLOSE_BOX` style) or by clicking a button with the `wxID_CANCEL` or `wxID_OK` ids.
- EVT_INIT_DIALOG(func): Process a `wxEVT_INIT_DIALOG` event. See wxInitDialogEvent.

Library: wxCore

Category: Common Dialogs

See also

wxDialog Overview, wxFrame, wxValidator Overview

Public Member Functions

- wxDialog ()

  Default constructor.
- wxDialog (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE, const wxString &name=wxDialogNameStr)

  Constructor.
- virtual ~wxDialog ()

  Destructor.
- void AddMainButtonId (wxWindowID id)

  Adds an identifier to be regarded as a main button for the non-scrolling area of a dialog.
- virtual bool CanDoLayoutAdaptation ()

  Returns true if this dialog can and should perform layout adaptation using DoLayoutAdaptation(), usually if the dialog is too large to fit on the display.
- void Centre (int direction=wxBOTH)

  Centres the dialog box on the display.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE, const wxString &name=wxDialogNameStr)

  Used for two-step dialog box construction.
- wxSizer * CreateButtonSizer (long flags)

  Creates a sizer with standard buttons.
- wxSizer * CreateSeparatedButtonSizer (long flags)

  Creates a sizer with standard buttons using CreateButtonSizer() separated from the rest of the dialog contents by a horizontal wxStaticLine.
- wxSizer * CreateSeparatedSizer (wxSizer *sizer)

  Returns the sizer containing the given one with a separating wxStaticLine if necessarily.
- wxStdDialogButtonSizer * CreateStdDialogButtonSizer (long flags)

  Creates a wxStdDialogButtonSizer with standard buttons.
- wxSizer * CreateTextSizer (const wxString &message)

  Splits text up at newlines and places the lines into wxStaticText objects in a vertical wxBoxSizer.

- virtual bool DoLayoutAdaptation ()

  Performs layout adaptation, usually if the dialog is too large to fit on the display.
- virtual bool DoOK ()

  This function is called when the titlebar OK button is pressed (PocketPC only).
- virtual void EndModal (int retCode)

  Ends a modal dialog, passing a value to be returned from the ShowModal() invocation.
- int GetAffirmativeId () const

  Gets the identifier of the button which works like standard OK button in this dialog.
- virtual wxWindow * GetContentWindow () const

  Override this to return a window containing the main content of the dialog.
- int GetEscapeId () const

  Gets the identifier of the button to map presses of `ESC` button to.
- bool GetLayoutAdaptationDone () const

  Returns true if the dialog has been adapted, usually by making it scrollable to work with a small display.
- int GetLayoutAdaptationLevel () const

  Gets a value representing the aggressiveness of search for buttons and sizers to be in the non-scrolling part of a layout-adapted dialog.
- wxDialogLayoutAdaptationMode GetLayoutAdaptationMode () const

  Gets the adaptation mode, overriding the global adaptation flag.
- wxArrayInt & GetMainButtonIds ()

  Returns an array of identifiers to be regarded as the main buttons for the non-scrolling area of a dialog.
- int GetReturnCode () const

  Gets the return code for this window.
- wxToolBar * GetToolBar () const

  On PocketPC, a dialog is automatically provided with an empty toolbar.
- virtual void Iconize (bool iconize=true)

  Iconizes or restores the dialog.
- virtual bool IsIconized () const

  Returns true if the dialog box is iconized.
- bool IsMainButtonId (wxWindowID id) const

  Returns true if id is in the array of identifiers to be regarded as the main buttons for the non-scrolling area of a dialog.
- virtual bool IsModal () const

  Returns true if the dialog box is modal, false otherwise.
- void SetAffirmativeId (int id)

  Sets the identifier to be used as OK button.
- void SetEscapeId (int id)

  Sets the identifier of the button which should work like the standard "Cancel" button in this dialog.
- void SetIcon (const wxIcon &icon)

  Sets the icon for this dialog.
- void SetIcons (const wxIconBundle &icons)

  Sets the icons for this dialog.
- void SetLayoutAdaptationDone (bool done)

  Marks the dialog as having been adapted, usually by making it scrollable to work with a small display.
- void SetLayoutAdaptationLevel (int level)

  Sets the aggressiveness of search for buttons and sizers to be in the non-scrolling part of a layout-adapted dialog.
- void SetLayoutAdaptationMode (wxDialogLayoutAdaptationMode mode)

  Sets the adaptation mode, overriding the global adaptation flag.
- void SetReturnCode (int retCode)

  Sets the return code for this window.
- virtual bool Show (bool show=1)

  Hides or shows the dialog.

- virtual int ShowModal ()

  Shows an application-modal dialog.
- void ShowWindowModal ()

  Shows a dialog modal to the parent top level window only.
- template<typename Functor >
  void ShowWindowModalThenDo (const Functor &onEndModal)

  Shows a dialog modal to the parent top level window only and call a functor after the dialog is closed.

Static Public Member Functions

- static void EnableLayoutAdaptation (bool enable)

  A static function enabling or disabling layout adaptation for all dialogs.
- static wxDialogLayoutAdapter * GetLayoutAdapter ()

  A static function getting the current layout adapter object.
- static bool IsLayoutAdaptationEnabled ()

  A static function returning true if layout adaptation is enabled for all dialogs.
- static wxDialogLayoutAdapter * SetLayoutAdapter (wxDialogLayoutAdapter *adapter)

  A static function for setting the current layout adapter object, returning the old adapter.

Additional Inherited Members

21.190.3 Constructor & Destructor Documentation

wxDialog::wxDialog ( )

Default constructor.

wxDialog::wxDialog (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE, const wxString & *name* = `wxDialogNameStr`)

Constructor.

Parameters

parent	Can be NULL, a frame or another dialog box.
id	An identifier for the dialog. A value of -1 is taken to mean a default.
title	The title of the dialog.
pos	The dialog position. The value wxDefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.
size	The dialog size. The value wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.
style	The window style.
name	Used to associate a name with the window, allowing the application user to set Motif resource values for individual dialog boxes.

See also

Create()

virtual wxDialog::~wxDialog ( ) [virtual]

Destructor.

Deletes any child windows before deleting the physical window.

See Window Deletion for more info.

21.190.4 Member Function Documentation

void wxDialog::AddMainButtonId (wxWindowID *id*)

Adds an identifier to be regarded as a main button for the non-scrolling area of a dialog.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

virtual bool wxDialog::CanDoLayoutAdaptation ( ) [virtual]

Returns true if this dialog can and should perform layout adaptation using DoLayoutAdaptation(), usually if the dialog is too large to fit on the display.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

void wxDialog::Centre (int *direction* = wxBOTH)

Centres the dialog box on the display.

Parameters

direction	May be wxHORIZONTAL, wxVERTICAL or wxBOTH.

bool wxDialog::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE, const wxString & *name* = wxDialogNameStr)

Used for two-step dialog box construction.

See also

wxDialog()

wxSizer* wxDialog::CreateButtonSizer (long *flags*)

Creates a sizer with standard buttons.

flags is a bit list of the following flags: wxOK, wxCANCEL, wxYES, wxNO, wxAPPLY, wxCLOSE, wxHELP, wxNO↩_DEFAULT.

The sizer lays out the buttons in a manner appropriate to the platform.

This function uses CreateStdDialogButtonSizer() internally for most platforms but doesn't create the sizer at all for the platforms with hardware buttons (such as smartphones) for which it sets up the hardware buttons appropriately and returns NULL, so don't forget to test that the return value is valid before using it.

wxSizer* wxDialog::CreateSeparatedButtonSizer (long *flags*)

Creates a sizer with standard buttons using CreateButtonSizer() separated from the rest of the dialog contents by a horizontal wxStaticLine.

Note

Just like CreateButtonSizer(), this function may return NULL if no buttons were created.

This is a combination of CreateButtonSizer() and CreateSeparatedSizer().

wxSizer* wxDialog::CreateSeparatedSizer (wxSizer * *sizer*)

Returns the sizer containing the given one with a separating wxStaticLine if necessarily.

This function is useful for creating the sizer containing footer-like contents in dialog boxes. It will add a separating static line only if it conforms to the current platform convention (currently it is not added under Mac where the use of static lines for grouping is discouraged and is added elsewhere).

Since

2.9.2

Parameters

sizer	The sizer to wrap, must be non-NULL.

Returns

The sizer wrapping the input one or possibly the input sizer itself if no wrapping is necessary.

wxStdDialogButtonSizer* wxDialog::CreateStdDialogButtonSizer (long *flags*)

Creates a wxStdDialogButtonSizer with standard buttons.

flags is a bit list of the following flags: wxOK, wxCANCEL, wxYES, wxNO, wxAPPLY, wxCLOSE, wxHELP, wxNO↩ _DEFAULT.

The sizer lays out the buttons in a manner appropriate to the platform.

wxSizer* wxDialog::CreateTextSizer (const wxString & *message*)

Splits text up at newlines and places the lines into wxStaticText objects in a vertical wxBoxSizer.

virtual bool wxDialog::DoLayoutAdaptation ( ) [virtual]

Performs layout adaptation, usually if the dialog is too large to fit on the display.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

virtual bool wxDialog::DoOK ( ) [virtual]

This function is called when the titlebar OK button is pressed (PocketPC only).

A command event for the identifier returned by GetAffirmativeId() is sent by default. You can override this function. If the function returns false, wxWidgets will call Close() for the dialog.

Availability: only available for the wxMSW port.

static void wxDialog::EnableLayoutAdaptation (bool *enable*) `[static]`

A static function enabling or disabling layout adaptation for all dialogs.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

virtual void wxDialog::EndModal (int *retCode*) `[virtual]`

Ends a modal dialog, passing a value to be returned from the ShowModal() invocation.

Parameters

retCode	The value that should be returned by ShowModal.

See also

ShowModal(), GetReturnCode(), SetReturnCode()

int wxDialog::GetAffirmativeId ( ) const

Gets the identifier of the button which works like standard OK button in this dialog.

See also

SetAffirmativeId()

virtual wxWindow* wxDialog::GetContentWindow ( ) const `[virtual]`

Override this to return a window containing the main content of the dialog.

This is particularly useful when the dialog implements pages, such as wxPropertySheetDialog, and allows the layout adaptation code to know that only the pages need to be made scrollable.

int wxDialog::GetEscapeId ( ) const

Gets the identifier of the button to map presses of `ESC` button to.

See also

SetEscapeId()

bool wxDialog::GetLayoutAdaptationDone ( ) const

Returns true if the dialog has been adapted, usually by making it scrollable to work with a small display.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

int wxDialog::GetLayoutAdaptationLevel ( ) const

Gets a value representing the aggressiveness of search for buttons and sizers to be in the non-scrolling part of a layout-adapted dialog.

Zero switches off adaptation, and 3 allows search for standard buttons anywhere in the dialog.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

wxDialogLayoutAdaptationMode wxDialog::GetLayoutAdaptationMode ( ) const

Gets the adaptation mode, overriding the global adaptation flag.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

static wxDialogLayoutAdapter* wxDialog::GetLayoutAdapter ( ) [static]

A static function getting the current layout adapter object.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

wxArrayInt& wxDialog::GetMainButtonIds ( )

Returns an array of identifiers to be regarded as the main buttons for the non-scrolling area of a dialog.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

int wxDialog::GetReturnCode ( ) const

Gets the return code for this window.

Remarks

A return code is normally associated with a modal dialog, where ShowModal() returns a code to the application.

See also

SetReturnCode(), ShowModal(), EndModal()

wxToolBar* wxDialog::GetToolBar ( ) const

On PocketPC, a dialog is automatically provided with an empty toolbar.

This function allows you to access the toolbar and add tools to it. Removing tools and adding arbitrary controls are not currently supported.

This function is not available on any other platform.

Availability: only available for the wxMSW port.

virtual void wxDialog::Iconize (bool *iconize* = `true`) `[virtual]`

Iconizes or restores the dialog.

Windows only.

Parameters

iconize	If true, iconizes the dialog box; if false, shows and restores it.

Remarks

Note that in Windows, iconization has no effect since dialog boxes cannot be iconized. However, applications may need to explicitly restore dialog boxes under Motif which have user-iconizable frames, and under Windows calling Iconize(false) will bring the window to the front, as does Show(true).

Reimplemented from wxTopLevelWindow.

virtual bool wxDialog::IsIconized ( ) const `[virtual]`

Returns true if the dialog box is iconized.

Windows only.

Remarks

Always returns false under Windows since dialogs cannot be iconized.

Reimplemented from wxTopLevelWindow.

static bool wxDialog::IsLayoutAdaptationEnabled ( ) `[static]`

A static function returning true if layout adaptation is enabled for all dialogs.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

bool wxDialog::IsMainButtonId (wxWindowID *id*) const

Returns true if *id* is in the array of identifiers to be regarded as the main buttons for the non-scrolling area of a dialog.

Availability: only available for the wxMSW port.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

virtual bool wxDialog::IsModal ( ) const `[virtual]`

Returns true if the dialog box is modal, false otherwise.

void wxDialog::SetAffirmativeId (int *id*)

Sets the identifier to be used as OK button.

When the button with this identifier is pressed, the dialog calls wxWindow::Validate() and wxWindow::TransferDataFromWindow() and, if they both return true, closes the dialog with the affirmative id return code.

Also, when the user presses a hardware OK button on the devices having one or the special OK button in the PocketPC title bar, an event with this id is generated.

By default, the affirmative id is wxID_OK.

See also

GetAffirmativeId(), SetEscapeId()

void wxDialog::SetEscapeId (int *id*)

Sets the identifier of the button which should work like the standard "Cancel" button in this dialog.

When the button with this id is clicked, the dialog is closed. Also, when the user presses ESC key in the dialog or closes the dialog using the close button in the title bar, this is mapped to the click of the button with the specified id.

By default, the escape id is the special value wxID_ANY meaning that wxID_CANCEL button is used if it's present in the dialog and otherwise the button with GetAffirmativeId() is used. Another special value for *id* is wxID_NONE meaning that ESC presses should be ignored. If any other value is given, it is interpreted as the id of the button to map the escape key to.

Note

This method should be used for custom modal dialog implemented in wxWidgets itself, native dialogs such as wxMessageDialog or wxFileDialog, handle ESC presses in their own way which cannot be customized.

void wxDialog::SetIcon (const wxIcon & *icon*)

Sets the icon for this dialog.

Parameters

icon	The icon to associate with this dialog.

See also

wxIcon

void wxDialog::SetIcons (const wxIconBundle & *icons*) [virtual]

Sets the icons for this dialog.

Parameters

icons	The icons to associate with this dialog.

See also

wxIconBundle

Reimplemented from wxTopLevelWindow.

void wxDialog::SetLayoutAdaptationDone (bool *done*)

Marks the dialog as having been adapted, usually by making it scrollable to work with a small display.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

void wxDialog::SetLayoutAdaptationLevel (int *level*)

Sets the aggressiveness of search for buttons and sizers to be in the non-scrolling part of a layout-adapted dialog.

Zero switches off adaptation, and 3 allows search for standard buttons anywhere in the dialog.

See also

Automatic Scrolled Dialogs (for more on layout adaptation)

void wxDialog::SetLayoutAdaptationMode (wxDialogLayoutAdaptationMode *mode*)

Sets the adaptation mode, overriding the global adaptation flag.

See also

wxDialogLayoutAdaptationMode, Automatic Scrolled Dialogs (for more on layout adaptation)

static wxDialogLayoutAdapter* wxDialog::SetLayoutAdapter (wxDialogLayoutAdapter * *adapter*) `[static]`

A static function for setting the current layout adapter object, returning the old adapter.

If you call this, you should delete the old adapter object.

See also

wxDialogLayoutAdapter, Automatic Scrolled Dialogs

void wxDialog::SetReturnCode (int *retCode*)

Sets the return code for this window.

A return code is normally associated with a modal dialog, where ShowModal() returns a code to the application. The function EndModal() calls SetReturnCode().

Parameters

retCode	The integer return code, usually a control identifier.

See also

GetReturnCode(), ShowModal(), EndModal()

virtual bool wxDialog::Show (bool *show* = `1`) `[virtual]`

Hides or shows the dialog.

The preferred way of dismissing a modal dialog is to use EndModal().

Parameters

show	If true, the dialog box is shown and brought to the front, otherwise the box is hidden. If false and the dialog is modal, control is returned to the calling program.

Reimplemented from wxWindow.

virtual int wxDialog::ShowModal ( ) [virtual]

Shows an application-modal dialog.

Program flow does not return until the dialog has been dismissed with EndModal().

Notice that it is possible to call ShowModal() for a dialog which had been previously shown with Show(), this allows to make an existing modeless dialog modal. However ShowModal() can't be called twice without intervening EndModal() calls.

Note that this function creates a temporary event loop which takes precedence over the application's main event loop (see wxEventLoopBase) and which is destroyed when the dialog is dismissed. This also results in a call to wxApp::ProcessPendingEvents().

Returns

The value set with SetReturnCode().

See also

ShowWindowModal(), ShowWindowModalThenDo(), EndModal(), GetReturnCode(), SetReturnCode()

Reimplemented in wxFileDialog, wxMessageDialog, wxSingleChoiceDialog, wxTextEntryDialog, wxMultiChoiceDialog, wxDirDialog, wxFontDialog, and wxColourDialog.

void wxDialog::ShowWindowModal ( )

Shows a dialog modal to the parent top level window only.

Unlike ShowModal(), dialogs shown with this function only prevent the user from interacting with their parent frame only but not with the rest of the application. They also don't block the program execution but instead return immediately, as Show(), and generate a wxEVT_WINDOW_MODAL_DIALOG_CLOSED event (wxWindowModalDialogEvent) later when the dialog is closed.

Currently this function is only fully implemented in wxOSX ports, under the other platforms it behaves like ShowModal() (but also sends the above mentioned event).

See also

wxWindowModalDialogEvent, ShowWindowModalThenDo()

Since

2.9.0

template<typename Functor > void wxDialog::ShowWindowModalThenDo (const Functor & *onEndModal*)

Shows a dialog modal to the parent top level window only and call a functor after the dialog is closed.

Same as the other ShowWindowModal() overload, but calls the functor passed as the argument upon completion, instead of generating the wxEVT_WINDOW_MODAL_DIALOG_CLOSED event.

This form is particularly useful in combination with C++11 lambdas, when it allows writing window-modal very similarly to how ShowModal() is used (with the notable exception of not being able to create the dialog on stack):

```
wxWindowPtr<wxDialog> dlg(new wxMessageDialog(this, "Hello!"));

dlg->ShowWindowModalThenDo([this,dlg](int retcode){
    if ( retcode == wxID_OK )
        DoSomething();
    // dlg is implicitly destroyed here, because the pointer was
    // explicitly captured by the lambda
});
```

Parameters

onEndModal	Function object to call when the dialog is closed. The functor is called with a single integer argument, dialog's return code.

Note

The dialog instance must not be destroyed until *onEndModal* is called. The best way to ensure that is to use wxWindowPtr to hold a pointer and include it in the lambda's capture, by value (not reference!), as shown in the example above.

Since

3.0

See also

wxWindowPtr<T>

21.191 wxDialogLayoutAdapter Class Reference

`#include <wx/dialog.h>`

21.191.1 Detailed Description

This abstract class is the base for classes that help wxWidgets perform run-time layout adaptation of dialogs.

Principally, this is to cater for small displays by making part of the dialog scroll, but the application developer may find other uses for layout adaption.

By default, there is one instance of wxStandardDialogLayoutAdapter which can perform adaptation for most custom dialogs and dialogs with book controls such as wxPropertySheetDialog.

Library: wxCore

Category: Window Layout

See also

Automatic Scrolled Dialogs

Public Member Functions

- wxDialogLayoutAdapter ()

  Default constructor.
- virtual bool CanDoLayoutAdaptation (wxDialog *dialog)=0

  Override this to returns true if adaptation can and should be done.

- virtual bool DoLayoutAdaptation (wxDialog *dialog)=0

  Override this to perform layout adaptation, such as making parts of the dialog scroll and resizing the dialog to fit the display.

21.191.2 Constructor & Destructor Documentation

wxDialogLayoutAdapter::wxDialogLayoutAdapter ( )

Default constructor.

21.191.3 Member Function Documentation

virtual bool wxDialogLayoutAdapter::CanDoLayoutAdaptation (wxDialog * *dialog*) `[pure virtual]`

Override this to returns true if adaptation can and should be done.

virtual bool wxDialogLayoutAdapter::DoLayoutAdaptation (wxDialog * *dialog*) `[pure virtual]`

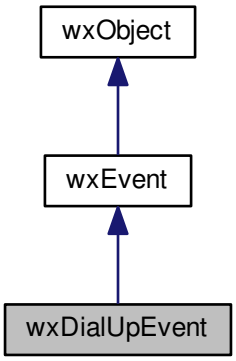
Override this to perform layout adaptation, such as making parts of the dialog scroll and resizing the dialog to fit the display.

Normally this function will be called just before the dialog is shown.

21.192 wxDialUpEvent Class Reference

`#include <wx/dialup.h>`

Inheritance diagram for wxDialUpEvent:



21.192.1 Detailed Description

This is the event class for the dialup events sent by wxDialUpManager.

Library: wxCore

Category: Events

Public Member Functions

- wxDialUpEvent (bool isConnected, bool isOwnEvent)

  Constructor is only used by wxDialUpManager.
- bool IsConnectedEvent () const

  Is this a `CONNECTED` *or* `DISCONNECTED` *event? In other words, does it notify about transition from offline to online state or vice versa?*
- bool IsOwnEvent () const

  Does this event come from wxDialUpManager::Dial() or from some external process (i.e.

Additional Inherited Members

21.192.2 Constructor & Destructor Documentation

wxDialUpEvent::wxDialUpEvent (bool *isConnected,* bool *isOwnEvent*)

Constructor is only used by wxDialUpManager.

21.192.3 Member Function Documentation

bool wxDialUpEvent::IsConnectedEvent ( ) const

Is this a `CONNECTED` or `DISCONNECTED` event? In other words, does it notify about transition from offline to online state or vice versa?

bool wxDialUpEvent::IsOwnEvent ( ) const

Does this event come from wxDialUpManager::Dial() or from some external process (i.e.

does it result from our own attempt to establish the connection)?

21.193 wxDialUpManager Class Reference

```
#include <wx/dialup.h>
```

21.193.1 Detailed Description

This class encapsulates functions dealing with verifying the connection status of the workstation (connected to the Internet via a direct connection, connected through a modem or not connected at all) and to establish this connection if possible/required (i.e.

in the case of the modem).

The program may also wish to be notified about the change in the connection status (for example, to perform some action when the user connects to the network the next time or, on the contrary, to stop receiving data from the net when the user hangs up the modem). For this, you need to use one of the event macros described below.

This class is different from other wxWidgets classes in that there is at most one instance of this class in the program accessed via Create() and you can't create the objects of this class directly.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxDialUpEvent& event)

Event macros for events emitted by this class:

- EVT_DIALUP_CONNECTED(func): A connection with the network was established.
- EVT_DIALUP_DISCONNECTED(func): The connection with the network was lost.

Library: wxCore

Category: Networking

See also

Dialup Sample, wxDialUpEvent

Public Member Functions

- virtual ~wxDialUpManager ()

  Destructor.
- virtual bool CancelDialing ()=0

  Cancel dialing the number initiated with Dial() with async parameter equal to true.
- virtual bool Dial (const wxString &nameOfISP=wxEmptyString, const wxString &username=wxEmptyString, const wxString &password=wxEmptyString, bool async=true)=0

  Dial the given ISP, use username and password to authenticate.
- virtual void DisableAutoCheckOnlineStatus ()=0

  Disable automatic check for connection status change - notice that the `wxEVT_DIALUP_XXX` events won't be sent any more neither.
- virtual bool EnableAutoCheckOnlineStatus (size_t nSeconds=60)=0

  Enable automatic checks for the connection status and sending of `wxEVT_DIALUP_CONNECTED/wxEVT_DIALUP_DISCONNECTED` events.
- virtual size_t GetISPNames (wxArrayString &names) const =0

  This function is only implemented under Windows.
- virtual bool HangUp ()=0

  Hang up the currently active dial up connection.
- virtual bool IsAlwaysOnline () const =0

  Returns true if the computer has a permanent network connection (i.e.
- virtual bool IsDialing () const =0

  Returns true if (async) dialing is in progress.
- virtual bool IsOk () const =0

  Returns true if the dialup manager was initialized correctly.
- virtual bool IsOnline () const =0

  Returns true if the computer is connected to the network: under Windows, this just means that a RAS connection exists, under Unix we check that the "well-known host" (as specified by SetWellKnownHost()) is reachable.
- virtual void SetConnectCommand (const wxString &commandDial="/usr/bin/pon", const wxString &commandHangup="/usr/bin/poff")=0

This method is for Unix only.

- virtual void SetOnlineStatus (bool isOnline=true)=0

  Sometimes the built-in logic for determining the online status may fail, so, in general, the user should be allowed to override it.

- virtual void SetWellKnownHost (const wxString &hostname, int portno=80)=0

  This method is for Unix only.

Static Public Member Functions

- static wxDialUpManager * Create ()

  This function should create and return the object of the platform-specific class derived from wxDialUpManager.

21.193.2 Constructor & Destructor Documentation

virtual wxDialUpManager::~wxDialUpManager ( ) [virtual]

Destructor.

21.193.3 Member Function Documentation

virtual bool wxDialUpManager::CancelDialing ( ) [pure virtual]

Cancel dialing the number initiated with Dial() with async parameter equal to true.

Note

This won't result in a DISCONNECTED event being sent.

See also

IsDialing()

static wxDialUpManager* wxDialUpManager::Create ( ) [static]

This function should create and return the object of the platform-specific class derived from wxDialUpManager.

You should delete the pointer when you are done with it.

virtual bool wxDialUpManager::Dial (const wxString & *nameOfISP* = wxEmptyString, const wxString & *username* = wxEmptyString, const wxString & *password* = wxEmptyString, bool *async* = true) [pure virtual]

Dial the given ISP, use *username* and *password* to authenticate.

The parameters are only used under Windows currently, for Unix you should use SetConnectCommand() to customize this functions behaviour.

If no *nameOfISP* is given, the function will select the default one (proposing the user to choose among all connections defined on this machine) and if no username and/or password are given, the function will try to do without them, but will ask the user if really needed.

If *async* parameter is false, the function waits until the end of dialing and returns true upon successful completion.

If *async* is true, the function only initiates the connection and returns immediately - the result is reported via events (an event is sent anyhow, but if dialing failed it will be a DISCONNECTED one).

virtual void wxDialUpManager::DisableAutoCheckOnlineStatus ( ) [pure virtual]

Disable automatic check for connection status change - notice that the wxEVT_DIALUP_XXX events won't be sent any more neither.

virtual bool wxDialUpManager::EnableAutoCheckOnlineStatus (size_t *nSeconds* = 60) [pure virtual]

Enable automatic checks for the connection status and sending of wxEVT_DIALUP_CONNECTED/wxEVT_DIALUP_DISCONNECTED events.

The interval parameter is only for Unix where we do the check manually and specifies how often should we repeat the check (each minute by default). Under Windows, the notification about the change of connection status is sent by the system and so we don't do any polling and this parameter is ignored.

Returns

false if couldn't set up automatic check for online status.

virtual size_t wxDialUpManager::GetISPNames (wxArrayString & *names*) const [pure virtual]

This function is only implemented under Windows.

Fills the array with the names of all possible values for the first parameter to Dial() on this machine and returns their number (may be 0).

virtual bool wxDialUpManager::HangUp ( ) [pure virtual]

Hang up the currently active dial up connection.

virtual bool wxDialUpManager::IsAlwaysOnline ( ) const [pure virtual]

Returns true if the computer has a permanent network connection (i.e.

\ is on a LAN) and so there is no need to use Dial() function to go online.

Note

This function tries to guess the result and it is not always guaranteed to be correct, so it is better to ask user for confirmation or give him a possibility to override it.

virtual bool wxDialUpManager::IsDialing ( ) const [pure virtual]

Returns true if (async) dialing is in progress.

See also

Dial()

virtual bool wxDialUpManager::IsOk ( ) const [pure virtual]

Returns true if the dialup manager was initialized correctly.

If this function returns false, no other functions will work neither, so it is a good idea to call this function and check its result before calling any other wxDialUpManager methods.

virtual bool wxDialUpManager::IsOnline ( ) const `[pure virtual]`

Returns true if the computer is connected to the network: under Windows, this just means that a RAS connection exists, under Unix we check that the "well-known host" (as specified by SetWellKnownHost()) is reachable.

virtual void wxDialUpManager::SetConnectCommand (const wxString & *commandDial* = "/usr/bin/pon", const wxString & *commandHangup* = "/usr/bin/poff") `[pure virtual]`

This method is for Unix only.

Sets the commands to start up the network and to hang up again.

virtual void wxDialUpManager::SetOnlineStatus (bool *isOnline* = true) `[pure virtual]`

Sometimes the built-in logic for determining the online status may fail, so, in general, the user should be allowed to override it.

This function allows to forcefully set the online status - whatever our internal algorithm may think about it.

See also

IsOnline()

virtual void wxDialUpManager::SetWellKnownHost (const wxString & *hostname*, int *portno* = 80) `[pure virtual]`

This method is for Unix only.

Under Unix, the value of well-known host is used to check whether we're connected to the internet. It is unused under Windows, but this function is always safe to call. The default value is "www.yahoo.com:80".

21.194 wxDir Class Reference

`#include <wx/dir.h>`

21.194.1 Detailed Description

wxDir is a portable equivalent of Unix open/read/closedir functions which allow enumerating of the files in a directory.

wxDir allows to enumerate files as well as directories.

wxDir also provides a flexible way to enumerate files recursively using Traverse() or a simpler GetAllFiles() function.

Example of use:

```
wxDir dir(wxGetCwd());

if ( !dir.IsOpened() )
{
    // deal with the error here - wxDir would already log an error message
    // explaining the exact reason of the failure
    return;
}

puts("Enumerating object files in current directory:");

wxString filename;

bool cont = dir.GetFirst(&filename, filespec, flags);
while ( cont )
{
    printf("%s\n", filename.c_str());

    cont = dir.GetNext(&filename);
}
```

Library: wxBase

Category: File Handling

Public Member Functions

- wxDir ()

  Default constructor, use Open() afterwards.
- wxDir (const wxString &dir)

  Opens the directory for enumeration, use IsOpened() to test for errors.
- ~wxDir ()

  Destructor cleans up the associated resources by calling Close().
- void Close ()

  Close the directory.
- bool GetFirst (wxString *filename, const wxString &filespec=wxEmptyString, int flags=wxDIR_DEFAULT) const

  Start enumerating all files matching filespec (or all files if it is empty) and flags, return true on success.
- wxString GetName () const

  Returns the name of the directory itself.
- wxString GetNameWithSep () const

  Returns the name of the directory with the path separator appended.
- bool GetNext (wxString *filename) const

  Continue enumerating files which satisfy the criteria specified by the last call to GetFirst().
- bool HasFiles (const wxString &filespec=wxEmptyString) const

  Returns true if the directory contains any files matching the given filespec.
- bool HasSubDirs (const wxString &dirspec=wxEmptyString) const

  Returns true if the directory contains any subdirectories (if a non empty filespec is given, only check for directories matching it).
- bool IsOpened () const

  Returns true if the directory was successfully opened by a previous call to Open().
- bool Open (const wxString &dir)

  Open the directory for enumerating, returns true on success or false if an error occurred.
- size_t Traverse (wxDirTraverser &sink, const wxString &filespec=wxEmptyString, int flags=wxDIR_DEFAULT) const

  Enumerate all files and directories under the given directory.

Static Public Member Functions

- static bool Exists (const wxString &dir)

  Test for existence of a directory with the given name.
- static wxString FindFirst (const wxString &dirname, const wxString &filespec, int flags=wxDIR_DEFAULT)

  The function returns the path of the first file matching the given filespec or an empty string if there are no files matching it.
- static size_t GetAllFiles (const wxString &dirname, wxArrayString *files, const wxString &filespec=wxEmptyString, int flags=wxDIR_DEFAULT)

  The function appends the names of all the files under directory dirname to the array files (note that its old content is preserved).
- static wxULongLong GetTotalSize (const wxString &dir, wxArrayString *filesSkipped=NULL)

  Returns the size (in bytes) of all files recursively found in `dir` or `wxInvalidSize` in case of error.
- static bool Make (const wxString &dir, int perm=wxS_DIR_DEFAULT, int flags=0)

Creates a directory.

- static bool Remove (const wxString &dir, int flags=0)

  Removes a directory.

21.194.2 Constructor & Destructor Documentation

wxDir::wxDir ( )

Default constructor, use Open() afterwards.

wxDir::wxDir (const wxString & *dir*)

Opens the directory for enumeration, use IsOpened() to test for errors.

wxDir::~wxDir ( )

Destructor cleans up the associated resources by calling Close().

It is not virtual and so this class is not meant to be used polymorphically.

21.194.3 Member Function Documentation

void wxDir::Close ( )

Close the directory.

The object can't be used after closing it, but Open() may be called again to reopen it later.

Since

2.9.5

static bool wxDir::Exists (const wxString & *dir*) `[static]`

Test for existence of a directory with the given name.

static wxString wxDir::FindFirst (const wxString & *dirname*, const wxString & *filespec*, int *flags* = wxDIR_DEFAULT) `[static]`

The function returns the path of the first file matching the given *filespec* or an empty string if there are no files matching it.

The *flags* parameter may or may not include wxDIR_FILES, the function always behaves as if it were specified. By default, *flags* includes wxDIR_DIRS and so the function recurses into the subdirectories but if this flag is not specified, the function restricts the search only to the directory *dirname* itself. See wxDirFlags for the list of the possible flags.

See also

Traverse()

static size_t wxDir::GetAllFiles (const wxString & *dirname*, wxArrayString * *files*, const wxString & *filespec* = wxEmptyString, int *flags* = wxDIR_DEFAULT) [static]

The function appends the names of all the files under directory *dirname* to the array *files* (note that its old content is preserved).

Only files matching the *filespec* are taken, with empty spec matching all the files.

The *flags* parameter should always include wxDIR_FILES or the array would be unchanged and should include wxDIR_DIRS flag to recurse into subdirectories (both flags are included in the value by default). See wxDirFlags for the list of the possible flags.

Returns

Returns the total number of files found while traversing the directory *dirname* (i.e. the number of entries appended to the *files* array).

See also

Traverse()

bool wxDir::GetFirst (wxString * *filename*, const wxString & *filespec* = wxEmptyString, int *flags* = wxDIR_DEFAULT) const

Start enumerating all files matching *filespec* (or all files if it is empty) and *flags*, return true on success.

See wxDirFlags for the list of the possible flags.

wxString wxDir::GetName ( ) const

Returns the name of the directory itself.

The returned string does not have the trailing path separator (slash or backslash).

Notice that in spite of this the last character of the returned string can still be the path separator if this directory is the root one. Because of this, don't append `wxFILE_SEP_PATH` to the returned value if you do need a slash-terminated directory name but use GetNameWithSep() instead to avoid having duplicate consecutive slashes.

wxString wxDir::GetNameWithSep ( ) const

Returns the name of the directory with the path separator appended.

The last character of the returned string is always `wxFILE_SEP_PATH` unless the string is empty, indicating that this directory is invalid.

See also

GetName()

Since

2.9.4

bool wxDir::GetNext (wxString * *filename*) const

Continue enumerating files which satisfy the criteria specified by the last call to GetFirst().

static wxULongLong wxDir::GetTotalSize (const wxString & *dir,* wxArrayString * *filesSkipped* = NULL) [static]

Returns the size (in bytes) of all files recursively found in `dir` or `wxInvalidSize` in case of error.

In case it happens that while traversing folders a file's size cannot be read, that file is added to the *filesSkipped* array, if not NULL, and then skipped. This usually happens with some special folders which are locked by the operating system or by another process. Remember that when the size of *filesSkipped* is not zero, then the returned value is not 100% accurate and, if the skipped files were big, it could be far from real size of the directory.

See also

wxFileName::GetHumanReadableSize(), wxGetDiskSpace()

bool wxDir::HasFiles (const wxString & *filespec* = wxEmptyString) const

Returns true if the directory contains any files matching the given *filespec*.

If *filespec* is empty, look for any files at all. In any case, even hidden files are taken into account.

bool wxDir::HasSubDirs (const wxString & *dirspec* = wxEmptyString) const

Returns true if the directory contains any subdirectories (if a non empty *filespec* is given, only check for directories matching it).

The hidden subdirectories are taken into account as well.

bool wxDir::IsOpened ( ) const

Returns true if the directory was successfully opened by a previous call to Open().

static bool wxDir::Make (const wxString & *dir,* int *perm* = wxS_DIR_DEFAULT, int *flags* = 0) [static]

Creates a directory.

This is just an alias for wxFileName::Mkdir(); refer to that function for more info.

bool wxDir::Open (const wxString & *dir*)

Open the directory for enumerating, returns true on success or false if an error occurred.

static bool wxDir::Remove (const wxString & *dir,* int *flags* = 0) [static]

Removes a directory.

This is just an alias for wxFileName::Rmdir(); refer to that function for more info.

size_t wxDir::Traverse (wxDirTraverser & *sink,* const wxString & *filespec* = wxEmptyString, int *flags* = wxDIR_DEFAULT) const

Enumerate all files and directories under the given directory.

If *flags* contains wxDIR_DIRS this enumeration is recursive, i.e. all the subdirectories of the given one and the files inside them will be traversed. Otherwise only the files in this directory itself are.

If *flags* doesn't contain wxDIR_FILES then only subdirectories are examined but not normal files. It doesn't make sense to not specify either wxDIR_DIRS or wxDIR_FILES and usually both of them should be specified, as is the case by default.

For each directory found, sink.OnDir() is called and sink.OnFile() is called for every file. Depending on the return value, the enumeration may continue or stop. If entering a subdirectory fails, sink.OnOpenError() is called.

The function returns the total number of files found or "`(size_t)-1`" on error.

See wxDirFlags for the full list of the possible flags.

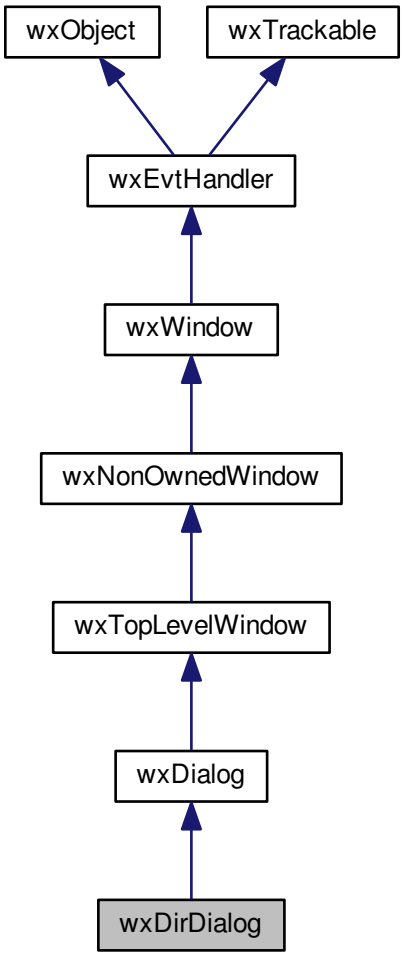
See also

GetAllFiles()

21.195 wxDirDialog Class Reference

`#include <wx/dirdlg.h>`

Inheritance diagram for wxDirDialog:



21.195.1 Detailed Description

This class represents the directory chooser dialog.

Styles

This class supports the following styles:

- wxDD_DEFAULT_STYLE: Equivalent to a combination of wxDEFAULT_DIALOG_STYLE and wxRESIZE↩ _BORDER (the last one is not used under wxWinCE).
- wxDD_DIR_MUST_EXIST: The dialog will allow the user to choose only an existing folder. When this style is not given, a "Create new directory" button is added to the dialog (on Windows) or some other way is provided to the user to type the name of a new folder.
- wxDD_CHANGE_DIR: Change the current working directory to the directory chosen by the user.

Notice that `wxRESIZE_BORDER` has special side effect under recent (i.e. later than Win9x) Windows where two different directory selection dialogs are available and this style also implicitly selects the new version as the old one always has fixed size. As the new version is almost always preferable, it is recommended that `wxRESIZE_BO↩ RDER` style be always used. This is the case if the dialog is created with the default style value but if you need to use any additional styles you should still specify `wxDD_DEFAULT_STYLE` unless you explicitly need to use the old dialog version under Windows. E.g. do

```
wxDirDialog dlg(NULL, "Choose input directory", "",
                wxDD_DEFAULT_STYLE | wxDD_DIR_MUST_EXIST);
```

instead of just using `wxDD_DIR_MUST_EXIST` style alone.

Library: wxCore

Category: Common Dialogs

See also

wxDirDialog Overview, wxFileDialog

Public Member Functions

- wxDirDialog (wxWindow *parent, const wxString &message=wxDirSelectorPromptStr, const wxString &defaultPath=wxEmptyString, long style=wxDD_DEFAULT_STYLE, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, const wxString &name=wxDirDialogNameStr)

  Constructor.
- virtual ~wxDirDialog ()

  Destructor.
- virtual wxString GetMessage () const

  Returns the message that will be displayed on the dialog.
- virtual wxString GetPath () const

  Returns the default or user-selected path.
- virtual void SetMessage (const wxString &message)

  Sets the message that will be displayed on the dialog.
- virtual void SetPath (const wxString &path)

  Sets the default path.
- int ShowModal ()

  Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

Additional Inherited Members

21.195.2 Constructor & Destructor Documentation

wxDirDialog::wxDirDialog (wxWindow * *parent*, const wxString & *message* = wxDirSelectorPromptStr, const wxString & *defaultPath* = wxEmptyString, long *style* = wxDD_DEFAULT_STYLE, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, const wxString & *name* = wxDirDialogNameStr)

Constructor.

Use ShowModal() to show the dialog.

Parameters

parent	Parent window.
message	Message to show on the dialog.
defaultPath	The default path, or the empty string.
style	The dialog style. See wxDirDialog
pos	Dialog position. Ignored under Windows.
size	Dialog size. Ignored under Windows.
name	The dialog name, not used.

virtual wxDirDialog::~wxDirDialog ( ) [virtual]

Destructor.

21.195.3 Member Function Documentation

virtual wxString wxDirDialog::GetMessage ( ) const [virtual]

Returns the message that will be displayed on the dialog.

virtual wxString wxDirDialog::GetPath ( ) const [virtual]

Returns the default or user-selected path.

virtual void wxDirDialog::SetMessage (const wxString & *message*) [virtual]

Sets the message that will be displayed on the dialog.

virtual void wxDirDialog::SetPath (const wxString & *path*) [virtual]

Sets the default path.

int wxDirDialog::ShowModal ( ) [virtual]

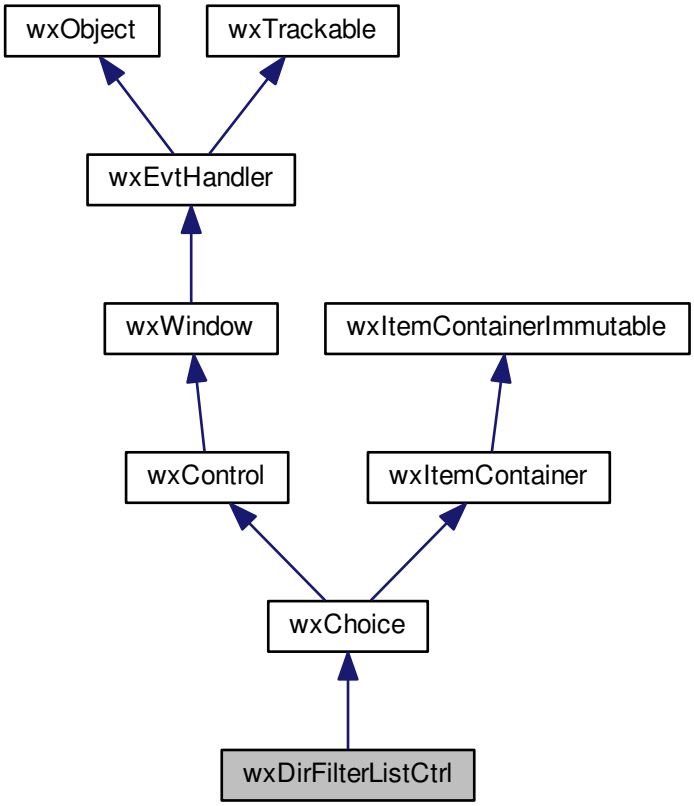
Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

Reimplemented from wxDialog.

21.196 wxDirFilterListCtrl Class Reference

```
#include <wx/dirctrl.h>
```

Inheritance diagram for wxDirFilterListCtrl:



Public Member Functions

- wxDirFilterListCtrl ()
- wxDirFilterListCtrl (wxGenericDirCtrl *parent, const wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)
- bool Create (wxGenericDirCtrl *parent, const wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)
- virtual ~wxDirFilterListCtrl ()
- void Init ()
- void FillFilterList (const wxString &filter, int defaultFilter)

Additional Inherited Members

21.196.1 Constructor & Destructor Documentation

wxDirFilterListCtrl::wxDirFilterListCtrl ( )

wxDirFilterListCtrl::wxDirFilterListCtrl (wxGenericDirCtrl * *parent*, const wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

virtual wxDirFilterListCtrl::~wxDirFilterListCtrl ( ) [inline],[virtual]

21.196.2 Member Function Documentation

bool wxDirFilterListCtrl::Create (wxGenericDirCtrl * *parent*, const wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

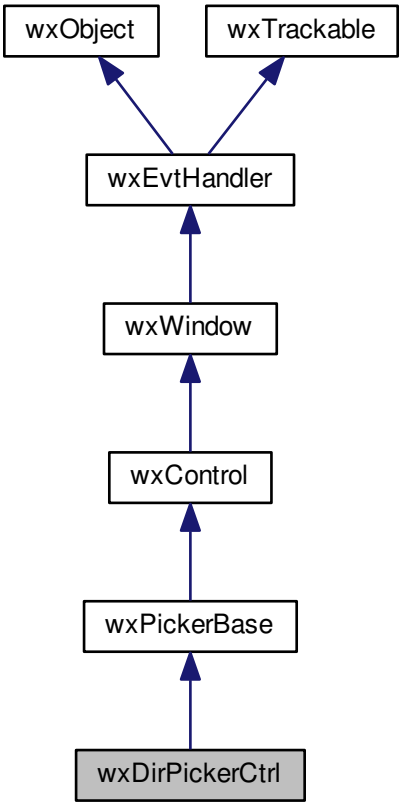
void wxDirFilterListCtrl::FillFilterList (const wxString & *filter*, int *defaultFilter*)

void wxDirFilterListCtrl::Init ( )

21.197 wxDirPickerCtrl Class Reference

#include <wx/filepicker.h>

Inheritance diagram for wxDirPickerCtrl:



21.197.1 Detailed Description

This control allows the user to select a directory.

The generic implementation is a button which brings up a wxDirDialog when clicked. Native implementation may differ but this is usually a (small) widget which give access to the dir-chooser dialog. It is only available if wxUSE↩

_DIRPICKERCTRL is set to 1 (the default).

Styles

This class supports the following styles:

- wxDIRP_DEFAULT_STYLE: The default style: includes wxDIRP_DIR_MUST_EXIST and, under wxMSW only, wxDIRP_USE_TEXTCTRL.
- wxDIRP_USE_TEXTCTRL: Creates a text control to the left of the picker button which is completely managed by the wxDirPickerCtrl and which can be used by the user to specify a path (see SetPath). The text control is automatically synchronized with button's value. Use functions defined in wxPickerBase to modify the text control.
- wxDIRP_DIR_MUST_EXIST: Creates a picker which allows to select only existing directories. wxGTK control always adds this flag internally as it does not support its absence.
- wxDIRP_CHANGE_DIR: Change current working directory on each user directory selection change.
- wxDIRP_SMALL: Use smaller version of the control with a small "..." button instead of the normal "Browse" one. This flag is new since wxWidgets 2.9.3.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFileDirPickerEvent& event)

Event macros for events emitted by this class:

- EVT_DIRPICKER_CHANGED(id, func): The user changed the directory selected in the control either using the button or using text control (see wxDIRP_USE_TEXTCTRL; note that in this case the event is fired only if the user's input is valid, e.g. an existing directory path).

Library: wxCore

Category: Picker Controls

See also

wxDirDialog, wxFileDirPickerEvent

Public Member Functions

- wxDirPickerCtrl ()
- wxDirPickerCtrl (wxWindow *parent, wxWindowID id, const wxString &path=wxEmptyString, const wxString &message=wxDirSelectorPromptStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDIRP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxDirPickerCtrlNameStr)

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &path=wxEmptyString, const wxString &message=wxDirSelectorPromptStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDIRP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxDirPickerCtrlNameStr)

  Creates the widgets with the given parameters.
- wxFileName GetDirName () const

Returns the absolute path of the currently selected directory as a wxFileName object.

- wxString GetPath () const

  Returns the absolute path of the currently selected directory.

- void SetDirName (const wxFileName &dirname)

  Just like SetPath() but this function takes a wxFileName object.

- void SetInitialDirectory (const wxString &dir)

  Set the directory to show when starting to browse for directories.

- void SetPath (const wxString &dirname)

  Sets the absolute path of the currently selected directory (the default converter uses current locale's charset).

Additional Inherited Members

21.197.2 Constructor & Destructor Documentation

wxDirPickerCtrl::wxDirPickerCtrl ( )

wxDirPickerCtrl::wxDirPickerCtrl (wxWindow * *parent*, wxWindowID *id*, const wxString & *path* = wxEmptyString, const wxString & *message* = wxDirSelectorPromptStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDIRP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxDirPickerCtrlNameStr`)

Initializes the object and calls Create() with all the parameters.

21.197.3 Member Function Documentation

bool wxDirPickerCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *path* = wxEmptyString, const wxString & *message* = wxDirSelectorPromptStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDIRP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxDirPickerCtrlNameStr`)

Creates the widgets with the given parameters.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
path	The initial directory shown in the control. Must be a valid path to a directory or the empty string.
message	The message shown to the user in the wxDirDialog shown by the control.
pos	Initial position.
size	Initial size.
style	The window style, see wxDIRP_* flags.
validator	Validator which can be used for additional date checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

wxFileName wxDirPickerCtrl::GetDirName ( ) const

Returns the absolute path of the currently selected directory as a wxFileName object.

This function is equivalent to GetPath().

wxString wxDirPickerCtrl::GetPath ( ) const

Returns the absolute path of the currently selected directory.

void wxDirPickerCtrl::SetDirName (const wxFileName & *dirname*)

Just like SetPath() but this function takes a wxFileName object.

void wxDirPickerCtrl::SetInitialDirectory (const wxString & *dir*)

Set the directory to show when starting to browse for directories.

This function is mostly useful for the directory picker controls which have no selection initially to configure the directory that should be shown if the user starts browsing for directories as otherwise the initially selected directory is used, which is usually the desired behaviour and so the directory specified by this function is ignored in this case.

Since

2.9.4

void wxDirPickerCtrl::SetPath (const wxString & *dirname*)

Sets the absolute path of the currently selected directory (the default converter uses current locale's charset).

This must be a valid directory if `wxDIRP_DIR_MUST_EXIST` style was given.

21.198 wxDirTraverser Class Reference

`#include <wx/dir.h>`

21.198.1 Detailed Description

wxDirTraverser is an abstract interface which must be implemented by objects passed to wxDir::Traverse() function.

Example of use (this works almost like wxDir::GetAllFiles()):

```
class wxDirTraverserSimple : public wxDirTraverser
{
public:
    wxDirTraverserSimple(wxArrayString& files) : m_files(files) { }

    virtual wxDirTraverseResult OnFile(const wxString& filename)
    {
        m_files.Add(filename);
        return wxDIR_CONTINUE;
    }

    virtual wxDirTraverseResult OnDir(const wxString& WXUNUSED(dirname))
    {
        return wxDIR_CONTINUE;
    }

private:
    wxArrayString& m_files;
};

// get the names of all files in the array
wxArrayString files;
wxDirTraverserSimple traverser(files);

wxDir dir(dirname);
dir.Traverse(traverser);
```

Library: wxBase

Category: File Handling

Public Member Functions

- virtual wxDirTraverseResult OnDir (const wxString &dirname)=0

  This function is called for each directory.
- virtual wxDirTraverseResult OnFile (const wxString &filename)=0

  This function is called for each file.
- virtual wxDirTraverseResult OnOpenError (const wxString &openerrorname)

  This function is called for each directory which we failed to open for enumerating.

21.198.2 Member Function Documentation

virtual wxDirTraverseResult wxDirTraverser::OnDir (const wxString & *dirname*) `[pure virtual]`

This function is called for each directory.

It may return wxDIR_STOP to abort traversing completely, wxDIR_IGNORE to skip this directory but continue with others or wxDIR_CONTINUE to enumerate all files and subdirectories in this directory.

This is a pure virtual function and must be implemented in the derived class.

virtual wxDirTraverseResult wxDirTraverser::OnFile (const wxString & *filename*) `[pure virtual]`

This function is called for each file.

It may return wxDIR_STOP to abort traversing (for example, if the file being searched is found) or wxDIR_CONTINUE to proceed.

This is a pure virtual function and must be implemented in the derived class.

virtual wxDirTraverseResult wxDirTraverser::OnOpenError (const wxString & *openerrorname*) `[virtual]`

This function is called for each directory which we failed to open for enumerating.

It may return wxDIR_STOP to abort traversing completely, wxDIR_IGNORE to skip this directory but continue with others or wxDIR_CONTINUE to retry opening this directory once again.

The base class version always returns wxDIR_IGNORE.

21.199 wxDisplay Class Reference

```
#include <wx/display.h>
```

21.199.1 Detailed Description

Determines the sizes and locations of displays connected to the system.

Library: wxCore

Category: Application and System configuration

See also

wxClientDisplayRect(), wxDisplaySize(), wxDisplaySizeMM()

Public Member Functions

- wxDisplay (unsigned int index=0)

  Constructor, setting up a wxDisplay instance with the specified display.
- ~wxDisplay ()

  Destructor.
- bool ChangeMode (const wxVideoMode &mode=wxDefaultVideoMode)

  Changes the video mode of this display to the mode specified in the mode parameter.
- wxRect GetClientArea () const

  Returns the client area of the display.
- wxVideoMode GetCurrentMode () const

  Returns the current video mode that this display is in.
- wxRect GetGeometry () const

  Returns the bounding rectangle of the display whose index was passed to the constructor.
- wxArrayVideoModes GetModes (const wxVideoMode &mode=wxDefaultVideoMode) const

  Fills and returns an array with all the video modes that are supported by this display, or video modes that are supported by this display and match the mode parameter (if mode is not wxDefaultVideoMode).
- wxString GetName () const

  Returns the display's name.
- bool IsPrimary () const

  Returns true if the display is the primary display.

Static Public Member Functions

- static unsigned int GetCount ()

  Returns the number of connected displays.
- static int GetFromPoint (const wxPoint &pt)

  Returns the index of the display on which the given point lies, or `wxNOT_FOUND` if the point is not on any connected display.
- static int GetFromWindow (const wxWindow *win)

  Returns the index of the display on which the given window lies.

21.199.2 Constructor & Destructor Documentation

wxDisplay::wxDisplay (unsigned int *index* = `0`)

Constructor, setting up a wxDisplay instance with the specified display.

Parameters

index	The index of the display to use. This must be non-negative and lower than the value returned by GetCount().

wxDisplay::~wxDisplay ( )

Destructor.

21.199.3 Member Function Documentation

bool wxDisplay::ChangeMode (const wxVideoMode & *mode* = wxDefaultVideoMode)

Changes the video mode of this display to the mode specified in the mode parameter.

If wxDefaultVideoMode is passed in as the mode parameter, the defined behaviour is that wxDisplay will reset the video mode to the default mode used by the display. On Windows, the behaviour is normal. However, there are differences on other platforms. On Unix variations using X11 extensions it should behave as defined, but some irregularities may occur.

wxRect wxDisplay::GetClientArea ( ) const

Returns the client area of the display.

The client area is the part of the display available for the normal (non full screen) windows, usually it is the same as GetGeometry() but it could be less if there is a taskbar (or equivalent) on this display.

static unsigned int wxDisplay::GetCount ( ) [static]

Returns the number of connected displays.

wxVideoMode wxDisplay::GetCurrentMode ( ) const

Returns the current video mode that this display is in.

static int wxDisplay::GetFromPoint (const wxPoint & *pt*) [static]

Returns the index of the display on which the given point lies, or `wxNOT_FOUND` if the point is not on any connected display.

Parameters

pt	The point to locate.

static int wxDisplay::GetFromWindow (const wxWindow * *win*) [static]

Returns the index of the display on which the given window lies.

If the window is on more than one display it gets the display that overlaps the window the most.

Returns `wxNOT_FOUND` if the window is not on any connected display.

Parameters

win	The window to locate.

wxRect wxDisplay::GetGeometry ( ) const

Returns the bounding rectangle of the display whose index was passed to the constructor.

See also

GetClientArea(), wxDisplaySize()

wxArrayVideoModes wxDisplay::GetModes (const wxVideoMode & *mode* = wxDefaultVideoMode) const

Fills and returns an array with all the video modes that are supported by this display, or video modes that are supported by this display and match the mode parameter (if mode is not wxDefaultVideoMode).

wxString wxDisplay::GetName ( ) const

Returns the display's name.

A name is not available on all platforms.

bool wxDisplay::IsPrimary ( ) const

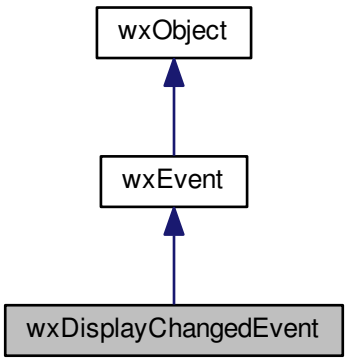
Returns true if the display is the primary display.

The primary display is the one whose index is 0.

21.200 wxDisplayChangedEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxDisplayChangedEvent:



Public Member Functions

- wxDisplayChangedEvent ()

Additional Inherited Members

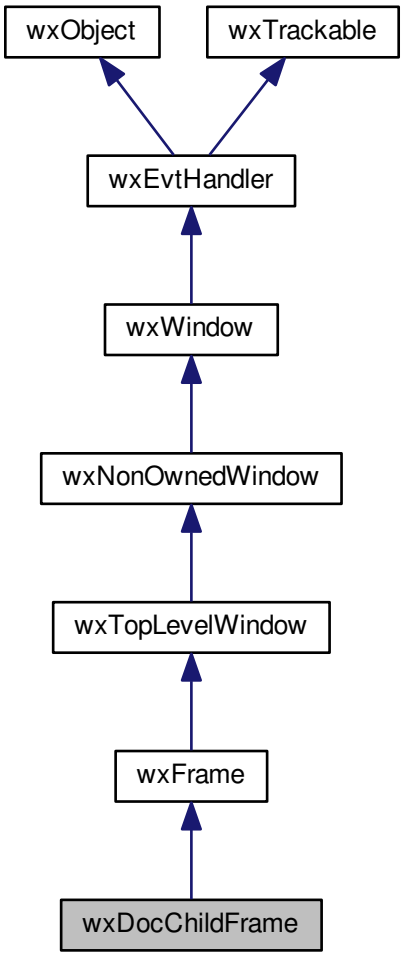
21.200.1 Constructor & Destructor Documentation

wxDisplayChangedEvent::wxDisplayChangedEvent ( )

21.201 wxDocChildFrame Class Reference

`#include <wx/docview.h>`

Inheritance diagram for wxDocChildFrame:



21.201.1 Detailed Description

The wxDocChildFrame class provides a default frame for displaying documents on separate windows.

This class can only be used for SDI (not MDI) child frames.

The class is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wx↩Document, wxDocManager and wxDocTemplate classes.

Notice that this class handles wxEVT_ACTIVATE event and activates the child view on receiving it. Don't intercept this event unless you want to prevent from this happening.

The same remark applies to wxEVT_CLOSE_WINDOW, as wxDocParentFrame the frame handles this event by trying to close the associated view.

Library: wxCore

Category: Document/View Framework

See also

Document/View Framework, Document/View Sample, wxFrame

Public Member Functions

- wxDocChildFrame (wxDocument *doc, wxView *view, wxFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT↩_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor.
- virtual ~wxDocChildFrame ()

  Destructor.
- wxDocument * GetDocument () const

  Returns the document associated with this frame.
- wxView * GetView () const

  Returns the view associated with this frame.
- void SetDocument (wxDocument *doc)

  Sets the document for this frame.
- void SetView (wxView *view)

  Sets the view for this frame.

Public Attributes

- wxDocument * m_childDocument

  The document associated with the frame.
- wxView * m_childView

  The view associated with the frame.

Additional Inherited Members

21.201.2 Constructor & Destructor Documentation

wxDocChildFrame::wxDocChildFrame (wxDocument * *doc,* wxView * *view,* wxFrame * *parent,* wxWindowID *id,* const wxString & *title,* const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor.

virtual wxDocChildFrame::~wxDocChildFrame ( ) `[virtual]`

Destructor.

21.201.3 Member Function Documentation

wxDocument* wxDocChildFrame::GetDocument ( ) const

Returns the document associated with this frame.

wxView* wxDocChildFrame::GetView ( ) const

Returns the view associated with this frame.

void wxDocChildFrame::SetDocument (wxDocument * *doc*)

Sets the document for this frame.

void wxDocChildFrame::SetView (wxView * *view*)

Sets the view for this frame.

21.201.4 Member Data Documentation

wxDocument* wxDocChildFrame::m_childDocument

The document associated with the frame.

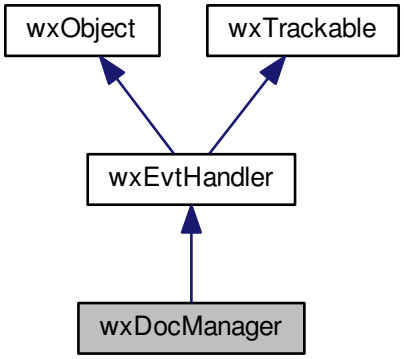
wxView* wxDocChildFrame::m_childView

The view associated with the frame.

21.202 wxDocManager Class Reference

`#include <wx/docview.h>`

Inheritance diagram for wxDocManager:



21.202.1 Detailed Description

The wxDocManager class is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocument and wxDocTemplate classes.

Library: wxCore

Category: Document/View Framework

See also

wxDocManager Overview, wxDocument, wxView, wxDocTemplate, wxFileHistory

Public Member Functions

- wxDocManager (long flags=0, bool initialize=true)

  Constructor.
- virtual ~wxDocManager ()

  Destructor.
- virtual void ActivateView (wxView *doc, bool activate=true)

  Sets the current view.
- void AddDocument (wxDocument *doc)

  Adds the document to the list of documents.
- virtual void AddFileToHistory (const wxString &filename)

  Adds a file to the file history list, if we have a pointer to an appropriate file menu.
- void AssociateTemplate (wxDocTemplate *temp)

  Adds the template to the document manager's template list.
- wxDocTemplate * FindTemplate (const wxClassInfo *classinfo)

  Search for a particular document template.
- wxDocument * FindDocumentByPath (const wxString &path) const

  Search for the document corresponding to the given file.

- bool CloseDocument (wxDocument *doc, bool force=false)

  Closes the specified document.
- bool CloseDocuments (bool force=true)

  Closes all currently opened documents.
- virtual wxDocument * CreateDocument (const wxString &path, long flags=0)

  Creates a new document.
- wxDocument * CreateNewDocument ()

  Creates an empty new document.
- virtual wxView * CreateView (wxDocument *doc, long flags=0)

  Creates a new view for the given document.
- void DisassociateTemplate (wxDocTemplate *temp)

  Removes the template from the list of templates.
- virtual void FileHistoryAddFilesToMenu ()

  Appends the files in the history list to all menus managed by the file history object.
- virtual void FileHistoryAddFilesToMenu (wxMenu *menu)

  Appends the files in the history list to the given menu only.
- virtual void FileHistoryLoad (const wxConfigBase &config)

  Loads the file history from a config object.
- virtual void FileHistoryRemoveMenu (wxMenu *menu)

  Removes the given menu from the list of menus managed by the file history object.
- virtual void FileHistorySave (wxConfigBase &resourceFile)

  Saves the file history into a config object.
- virtual void FileHistoryUseMenu (wxMenu *menu)

  Use this menu for appending recently-visited document filenames, for convenient access.
- virtual wxDocTemplate * FindTemplateForPath (const wxString &path)

  Given a path, try to find template that matches the extension.
- wxView * GetAnyUsableView () const

  Returns the view to apply a user command to.
- wxDocument * GetCurrentDocument () const

  Returns the document associated with the currently active view (if any).
- virtual wxView * GetCurrentView () const

  Returns the currently active view.
- wxDocVector GetDocumentsVector () const

  Returns a vector of wxDocument pointers.
- wxDocTemplateVector GetTemplatesVector () const

  Returns a vector of wxDocTemplate pointers.
- wxList & GetDocuments ()

  Returns a reference to the list of documents.
- virtual wxFileHistory * GetFileHistory () const

  Returns a pointer to file history.
- virtual size_t GetHistoryFilesCount () const

  Returns the number of files currently stored in the file history.
- wxString GetLastDirectory () const

  Returns the directory last selected by the user when opening a file.
- int GetMaxDocsOpen () const

  Returns the number of documents that can be open simultaneously.
- wxList & GetTemplates ()

  Returns a reference to the list of associated templates.
- virtual bool Initialize ()

  Initializes data; currently just calls OnCreateFileHistory().
- virtual wxString MakeNewDocumentName ()

Return a string containing a suitable default name for a new document.

- virtual wxFileHistory * OnCreateFileHistory ()

  A hook to allow a derived class to create a different type of file history.
- void OnFileClose (wxCommandEvent &event)

  Closes and deletes the currently active document.
- void OnFileCloseAll (wxCommandEvent &event)

  Closes and deletes all the currently opened documents.
- void OnFileNew (wxCommandEvent &event)

  Creates a document from a list of templates (if more than one template).
- void OnFileOpen (wxCommandEvent &event)

  Creates a new document and reads in the selected file.
- void OnFileRevert (wxCommandEvent &event)

  Reverts the current document by calling wxDocument::Revert() for the current document.
- void OnFileSave (wxCommandEvent &event)

  Saves the current document by calling wxDocument::Save() for the current document.
- void OnFileSaveAs (wxCommandEvent &event)

  Calls wxDocument::SaveAs() for the current document.
- void RemoveDocument (wxDocument *doc)

  Removes the document from the list of documents.
- virtual wxDocTemplate * SelectDocumentPath (wxDocTemplate **templates, int noTemplates, wxString &path, long flags, bool save=false)

  Under Windows, pops up a file selector with a list of filters corresponding to document templates.
- virtual wxDocTemplate * SelectDocumentType (wxDocTemplate **templates, int noTemplates, bool sort=false)

  Returns a document template by asking the user (if there is more than one template).
- virtual wxDocTemplate * SelectViewType (wxDocTemplate **templates, int noTemplates, bool sort=false)

  Returns a document template by asking the user (if there is more than one template), displaying a list of valid views.
- void SetLastDirectory (const wxString &dir)

  Sets the directory to be displayed to the user when opening a file.
- void SetMaxDocsOpen (int n)

  Sets the maximum number of documents that can be open at a time.

Protected Member Functions

- virtual void OnMRUFileNotExist (unsigned n, const wxString &filename)

  Called when a file selected from the MRU list doesn't exist any more.
- virtual wxPreviewFrame * CreatePreviewFrame (wxPrintPreviewBase *preview, wxWindow *parent, const wxString &title)

  Create the frame used for print preview.

Protected Attributes

- wxView * m_currentView

  The currently active view.
- int m_defaultDocumentNameCounter

  Stores the integer to be used for the next default document name.
- wxList m_docs

  A list of all documents.
- wxFileHistory * m_fileHistory

  A pointer to an instance of wxFileHistory, which manages the history of recently-visited files on the File menu.
- wxString m_lastDirectory

The directory last selected by the user when opening a file.

- int m_maxDocsOpen

  Stores the maximum number of documents that can be opened before existing documents are closed.

Additional Inherited Members

21.202.2 Constructor & Destructor Documentation

wxDocManager::wxDocManager (long *flags* = `0`, bool *initialize* = `true`)

Constructor.

Create a document manager instance dynamically near the start of your application before doing any document or view operations.

If *initialize* is true, the Initialize() function will be called to create a default history list object. If you derive from wxDocManager, you may wish to call the base constructor with false, and then call Initialize() in your own constructor, to allow your own Initialize() or OnCreateFileHistory functions to be called.

Parameters

flags	Currently unused.
initialize	Indicates whether Initialize() should be called by this ctor.

virtual wxDocManager::~wxDocManager ( ) `[virtual]`

Destructor.

21.202.3 Member Function Documentation

virtual void wxDocManager::ActivateView (wxView * *doc*, bool *activate* = `true`) `[virtual]`

Sets the current view.

void wxDocManager::AddDocument (wxDocument * *doc*)

Adds the document to the list of documents.

virtual void wxDocManager::AddFileToHistory (const wxString & *filename*) `[virtual]`

Adds a file to the file history list, if we have a pointer to an appropriate file menu.

void wxDocManager::AssociateTemplate (wxDocTemplate * *temp*)

Adds the template to the document manager's template list.

bool wxDocManager::CloseDocument (wxDocument * *doc*, bool *force* = `false`)

Closes the specified document.

If *force* is true, the document is closed even if it has unsaved changes.

Parameters

doc	The document to close, must be non-NULL.
force	If true, close the document even if wxDocument::Close() returns false.

Returns

true if the document was closed or false if closing it was cancelled by user (only in *force* = false case).

bool wxDocManager::CloseDocuments (bool *force* = `true`)

Closes all currently opened documents.

See also

CloseDocument()

virtual wxDocument* wxDocManager::CreateDocument (const wxString & *path*, long *flags* = `0`) `[virtual]`

Creates a new document.

This function can either create a document corresponding to a new file or to an already existing one depending on whether `wxDOC_NEW` is specified in the *flags*.

By default, this function asks the user for the type of document to open and the path to its file if it's not specified, i.e. if *path* is empty. Specifying `wxDOC_SILENT` flag suppresses any prompts and means that the *path* must be non-empty and there must be a registered document template handling the extension of this file, otherwise a warning message is logged and the function returns NULL. Notice that `wxDOC_SILENT` can be combined with `wxDOC_NEW`, however in this case the *path* must still be specified, even if the file with this path typically won't exist.

Finally notice that if this document manager was configured to allow only a limited number of simultaneously opened documents using SetMaxDocsOpen(), this function will try to close the oldest existing document if this number was reached before creating a new document. And if closing the old document fails (e.g. because it was vetoed by user), this function fails as well.

Parameters

path	Path to a file or an empty string. If the path is empty, the user will be asked to select it (thus, this is incompatible with the use of `wxDOC_SILENT`). The file should exist unless *flags* includes `wxDOC_NEW`.
flags	By default, none. May include `wxDOC_NEW` to indicate that the new document corresponds to a new file and not an existing one and `wxDOC_SILENT` to suppress any dialogs asking the user about the file path and type.

Returns

a new document object or NULL on failure.

wxDocument* wxDocManager::CreateNewDocument ( )

Creates an empty new document.

This is equivalent to calling CreateDocument() with `wxDOC_NEW` flags and without the file name.

virtual wxPreviewFrame* wxDocManager::CreatePreviewFrame (wxPrintPreviewBase * *preview*, wxWindow * *parent*, const wxString & *title*) `[protected]`, `[virtual]`

Create the frame used for print preview.

This method can be overridden if you need to change the behaviour or appearance of the preview window. By default, a standard wxPreviewFrame is created.

Since

2.9.1

Parameters

preview	The associated preview object.
parent	The parent window for the frame.
title	The suggested title for the print preview frame.

Returns

A new print preview frame, must not return NULL.

virtual wxView* wxDocManager::CreateView (wxDocument * *doc*, long *flags* = 0) [virtual]

Creates a new view for the given document.

If more than one view is allowed for the document (by virtue of multiple templates mentioning the same document type), a choice of view is presented to the user.

void wxDocManager::DisassociateTemplate (wxDocTemplate * *temp*)

Removes the template from the list of templates.

virtual void wxDocManager::FileHistoryAddFilesToMenu ( ) [virtual]

Appends the files in the history list to all menus managed by the file history object.

virtual void wxDocManager::FileHistoryAddFilesToMenu (wxMenu * *menu*) [virtual]

Appends the files in the history list to the given *menu* only.

virtual void wxDocManager::FileHistoryLoad (const wxConfigBase & *config*) [virtual]

Loads the file history from a config object.

See also

wxConfigBase

virtual void wxDocManager::FileHistoryRemoveMenu (wxMenu * *menu*) [virtual]

Removes the given menu from the list of menus managed by the file history object.

virtual void wxDocManager::FileHistorySave (wxConfigBase & *resourceFile*) [virtual]

Saves the file history into a config object.

This must be called explicitly by the application.

See also

wxConfigBase

virtual void wxDocManager::FileHistoryUseMenu (wxMenu * *menu*) [virtual]

Use this menu for appending recently-visited document filenames, for convenient access.

Calling this function with a valid menu pointer enables the history list functionality.

Note

You can add multiple menus using this function, to be managed by the file history object.

wxDocument* wxDocManager::FindDocumentByPath (const wxString & *path*) const

Search for the document corresponding to the given file.

Parameters

path	Document file path.

Returns

Pointer to a wxDocument, or NULL if none found.

Since

2.9.5

wxDocTemplate* wxDocManager::FindTemplate (const wxClassInfo * *classinfo*)

Search for a particular document template.

Example:

```
// creating a document instance of the specified document type:
m_doc = (MyDoc*)docManager->FindTemplate(CLASSINFO(MyDoc))->
            CreateDocument(wxEmptyString, wxDOC_SILENT);
```

Parameters

classinfo	Class info of a document class for which a wxDocTemplate had been previously created.

Returns

Pointer to a wxDocTemplate, or NULL if none found.

Since

2.9.2

virtual wxDocTemplate* wxDocManager::FindTemplateForPath (const wxString & *path*) [virtual]

Given a path, try to find template that matches the extension.

This is only an approximate method of finding a template for creating a document.

wxView* wxDocManager::GetAnyUsableView ( ) const

Returns the view to apply a user command to.

This method tries to find the view that the user wants to interact with. It returns the same view as GetCurrentDocument() if there is any currently active view but falls back to the first view of the first document if there is no active view.

Since

2.9.5

wxDocument* wxDocManager::GetCurrentDocument ( ) const

Returns the document associated with the currently active view (if any).

virtual wxView* wxDocManager::GetCurrentView ( ) const `[virtual]`

Returns the currently active view.

This method can return NULL if no view is currently active.

See also

GetAnyUsableView()

wxList& wxDocManager::GetDocuments ( )

Returns a reference to the list of documents.

wxDocVector wxDocManager::GetDocumentsVector ( ) const

Returns a vector of wxDocument pointers.

Since

2.9.5

virtual wxFileHistory* wxDocManager::GetFileHistory ( ) const `[virtual]`

Returns a pointer to file history.

virtual size_t wxDocManager::GetHistoryFilesCount ( ) const `[virtual]`

Returns the number of files currently stored in the file history.

wxString wxDocManager::GetLastDirectory ( ) const

Returns the directory last selected by the user when opening a file.

Initially empty.

int wxDocManager::GetMaxDocsOpen ( ) const

Returns the number of documents that can be open simultaneously.

wxList& wxDocManager::GetTemplates ( )

Returns a reference to the list of associated templates.

wxDocTemplateVector wxDocManager::GetTemplatesVector ( ) const

Returns a vector of wxDocTemplate pointers.

Since

2.9.5

virtual bool wxDocManager::Initialize ( ) [virtual]

Initializes data; currently just calls OnCreateFileHistory().

Some data cannot always be initialized in the constructor because the programmer must be given the opportunity to override functionality. If OnCreateFileHistory() was called from the constructor, an overridden virtual OnCreate↩ FileHistory() would not be called due to C++'s 'interesting' constructor semantics. In fact Initialize() *is* called from the wxDocManager constructor, but this can be vetoed by passing false to the second argument, allowing the derived class's constructor to call Initialize(), possibly calling a different OnCreateFileHistory() from the default.

The bottom line: if you're not deriving from Initialize(), forget it and construct wxDocManager with no arguments.

virtual wxString wxDocManager::MakeNewDocumentName ( ) [virtual]

Return a string containing a suitable default name for a new document.

By default this is implemented by appending an integer counter to the string **unnamed** but can be overridden in the derived classes to do something more appropriate.

virtual wxFileHistory* wxDocManager::OnCreateFileHistory ( ) [virtual]

A hook to allow a derived class to create a different type of file history.

Called from Initialize().

void wxDocManager::OnFileClose (wxCommandEvent & *event*)

Closes and deletes the currently active document.

void wxDocManager::OnFileCloseAll (wxCommandEvent & *event*)

Closes and deletes all the currently opened documents.

void wxDocManager::OnFileNew (wxCommandEvent & *event*)

Creates a document from a list of templates (if more than one template).

void wxDocManager::OnFileOpen (wxCommandEvent & *event*)

Creates a new document and reads in the selected file.

void wxDocManager::OnFileRevert (wxCommandEvent & *event*)

Reverts the current document by calling wxDocument::Revert() for the current document.

void wxDocManager::OnFileSave (wxCommandEvent & *event*)

Saves the current document by calling wxDocument::Save() for the current document.

void wxDocManager::OnFileSaveAs (wxCommandEvent & *event*)

Calls wxDocument::SaveAs() for the current document.

virtual void wxDocManager::OnMRUFileNotExist (unsigned *n,* const wxString & *filename*) `[protected], [virtual]`

Called when a file selected from the MRU list doesn't exist any more.

The default behaviour is to remove the file from the MRU (most recently used) files list and the corresponding menu and notify the user about it but this method can be overridden to customize it.

For example, an application may want to just give an error about the missing file *filename* but not remove it from the file history. Or it could ask the user whether the file should be kept or removed.

Notice that this method is called only if the file selected by user from the MRU files in the menu doesn't exist, but not if opening it failed for any other reason because in the latter case the default behaviour of removing the file from the MRU list is inappropriate. If you still want to do it, you would need to do it by calling RemoveFileFromHistory() explicitly in the part of the file opening code that may fail.

Since

2.9.3

Parameters

n	The index of the file in the MRU list, it can be passed to RemoveFileFromHistory() to remove this file from the list.
filename	The full name of the file.

void wxDocManager::RemoveDocument (wxDocument * *doc*)

Removes the document from the list of documents.

virtual wxDocTemplate* wxDocManager::SelectDocumentPath (wxDocTemplate ** *templates,* int *noTemplates,* wxString & *path,* long *flags,* bool *save* = `false`) `[virtual]`

Under Windows, pops up a file selector with a list of filters corresponding to document templates.

The wxDocTemplate corresponding to the selected file's extension is returned.

On other platforms, if there is more than one document template a choice list is popped up, followed by a file selector.

This function is used in CreateDocument().

wxPerl Note: In wxPerl *templates* is a reference to a list of templates. If you override this method in your document manager it must return two values, eg:

```
(doctemplate, path) = My::DocManager->SelectDocumentPath(...);
```

virtual wxDocTemplate* wxDocManager::SelectDocumentType (wxDocTemplate ** *templates*, int *noTemplates*, bool *sort* = false) [virtual]

Returns a document template by asking the user (if there is more than one template).

This function is used in CreateDocument().

Parameters

templates	Pointer to an array of templates from which to choose a desired template.
noTemplates	Number of templates being pointed to by the templates pointer.
sort	If more than one template is passed into templates, then this parameter indicates whether the list of templates that the user will have to choose from is sorted or not when shown the choice box dialog. Default is false.

wxPerl Note: In wxPerl *templates* is a reference to a list of templates.

virtual wxDocTemplate* wxDocManager::SelectViewType (wxDocTemplate ** *templates*, int *noTemplates*, bool *sort* = false) [virtual]

Returns a document template by asking the user (if there is more than one template), displaying a list of valid views.

This function is used in CreateView(). The dialog normally will not appear because the array of templates only contains those relevant to the document in question, and often there will only be one such.

Parameters

templates	Pointer to an array of templates from which to choose a desired template.
noTemplates	Number of templates being pointed to by the templates pointer.
sort	If more than one template is passed into templates, then this parameter indicates whether the list of templates that the user will have to choose from is sorted or not when shown the choice box dialog. Default is false.

wxPerl Note: In wxPerl *templates* is a reference to a list of templates.

void wxDocManager::SetLastDirectory (const wxString & *dir*)

Sets the directory to be displayed to the user when opening a file.

Initially this is empty.

void wxDocManager::SetMaxDocsOpen (int *n*)

Sets the maximum number of documents that can be open at a time.

By default, this is INT_MAX, i.e. the number of documents is unlimited. If you set it to 1, existing documents will be saved and deleted when the user tries to open or create a new one (similar to the behaviour of Windows Write, for example). Allowing multiple documents gives behaviour more akin to MS Word and other Multiple Document Interface applications.

21.202.4 Member Data Documentation

wxView* wxDocManager::m_currentView [protected]

The currently active view.

int wxDocManager::m_defaultDocumentNameCounter [protected]

Stores the integer to be used for the next default document name.

wxList wxDocManager::m_docs [protected]

A list of all documents.

wxFileHistory* **wxDocManager::m_fileHistory** [protected]

A pointer to an instance of wxFileHistory, which manages the history of recently-visited files on the File menu.

wxString wxDocManager::m_lastDirectory [protected]

The directory last selected by the user when opening a file.

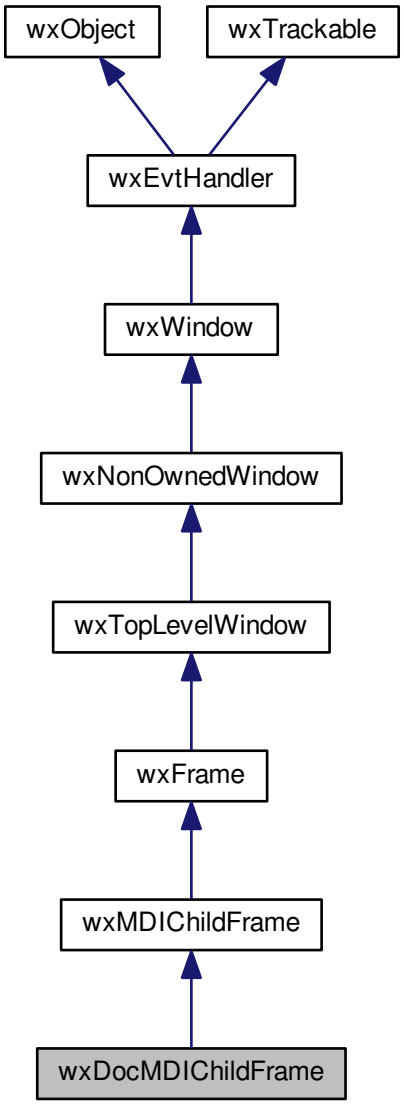
int wxDocManager::m_maxDocsOpen [protected]

Stores the maximum number of documents that can be opened before existing documents are closed.
By default, this is `INT_MAX` i.e. practically unlimited.

21.203 wxDocMDIChildFrame Class Reference

```
#include <wx/docmdi.h>
```

Inheritance diagram for wxDocMDIChildFrame:



21.203.1 Detailed Description

The wxDocMDIChildFrame class provides a default frame for displaying documents on separate windows.

This class can only be used for MDI child frames.

The class is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wx↩ Document, wxDocManager and wxDocTemplate classes.

Library: wxCore

Category: Document/View Framework

See also

Document/View Framework, Document/View Sample, wxMDIChildFrame

Public Member Functions

- wxDocMDIChildFrame (wxDocument *doc, wxView *view, wxMDIParentFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor.
- virtual ~wxDocMDIChildFrame ()

  Destructor.
- wxDocument * GetDocument () const

  Returns the document associated with this frame.
- wxView * GetView () const

  Returns the view associated with this frame.
- void SetDocument (wxDocument *doc)

  Sets the document for this frame.
- void SetView (wxView *view)

  Sets the view for this frame.

Additional Inherited Members

21.203.2 Constructor & Destructor Documentation

wxDocMDIChildFrame::wxDocMDIChildFrame (wxDocument * *doc,* wxView * *view,* wxMDIParentFrame * *parent,* wxWindowID *id,* const wxString & *title,* const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor.

virtual wxDocMDIChildFrame::~wxDocMDIChildFrame ( ) `[virtual]`

Destructor.

21.203.3 Member Function Documentation

wxDocument* wxDocMDIChildFrame::GetDocument ( ) const

Returns the document associated with this frame.

wxView* wxDocMDIChildFrame::GetView ( ) const

Returns the view associated with this frame.

void wxDocMDIChildFrame::SetDocument (wxDocument * *doc*)

Sets the document for this frame.

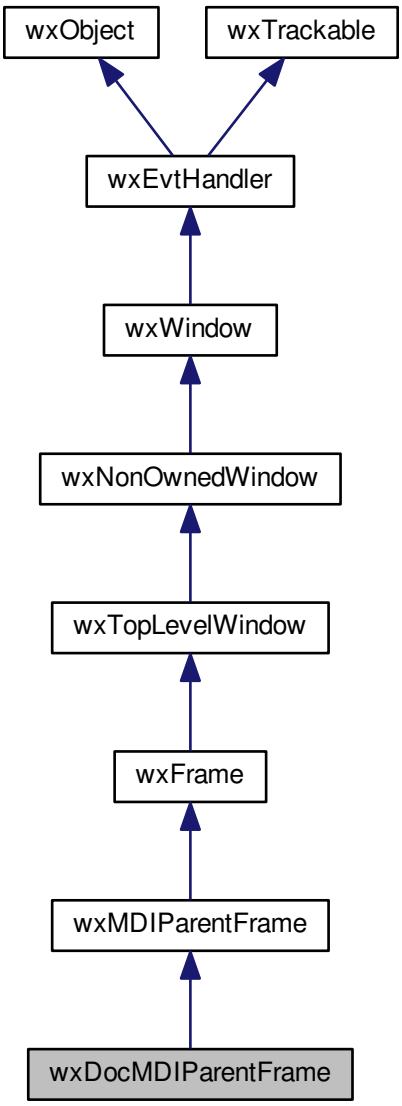
void wxDocMDIChildFrame::SetView (wxView * *view*)

Sets the view for this frame.

21.204 wxDocMDIParentFrame Class Reference

`#include <wx/docmdi.h>`

Inheritance diagram for wxDocMDIParentFrame:



21.204.1 Detailed Description

The wxDocMDIParentFrame class provides a default top-level frame for applications using the document/view framework.

This class can only be used for MDI parent frames.

It cooperates with the wxView, wxDocument, wxDocManager and wxDocTemplate classes.

Library: wxCore

Category: Document/View Framework

See also

Document/View Framework, Document/View Sample, wxMDIParentFrame

Public Member Functions

- virtual ~wxDocMDIParentFrame ()

  Destructor.
- bool Create (wxDocManager *manager, wxFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Creates the window.

- wxDocMDIParentFrame ()

  Constructor.
- wxDocMDIParentFrame (wxDocManager *manager, wxFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor.

Additional Inherited Members

21.204.2 Constructor & Destructor Documentation

wxDocMDIParentFrame::wxDocMDIParentFrame ( )

Constructor.

wxDocMDIParentFrame::wxDocMDIParentFrame (wxDocManager * *manager*, wxFrame * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor.

virtual wxDocMDIParentFrame::~wxDocMDIParentFrame ( ) `[virtual]`

Destructor.

21.204.3 Member Function Documentation

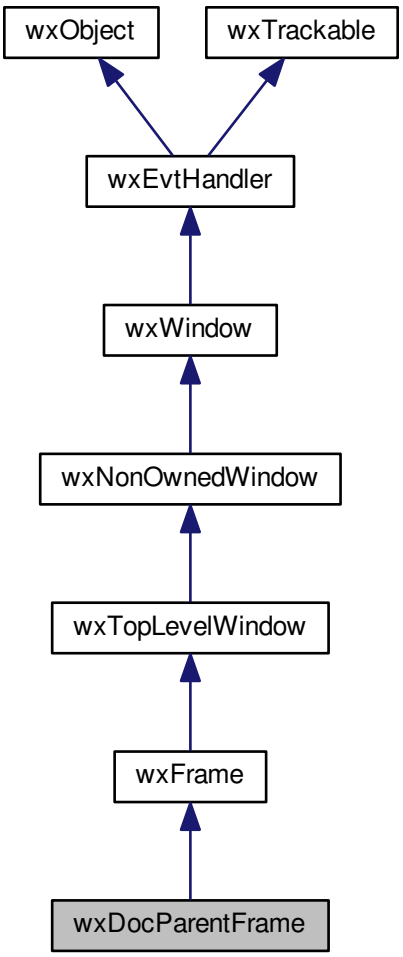
bool wxDocMDIParentFrame::Create (wxDocManager * *manager*, wxFrame * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Creates the window.

21.205 wxDocParentFrame Class Reference

`#include <wx/docview.h>`

Inheritance diagram for wxDocParentFrame:



21.205.1 Detailed Description

The wxDocParentFrame class provides a default top-level frame for applications using the document/view framework.

This class can only be used for SDI (not MDI) parent frames.

It cooperates with the wxView, wxDocument, wxDocManager and wxDocTemplate classes.

Notice that this class processes wxEVT_CLOSE_WINDOW event and tries to close all open views from its handler. If all the views can be closed, i.e. if none of them contains unsaved changes or the user decides to not save them, the window is destroyed. Don't intercept this event in your code unless you want to replace this logic.

Library: wxCore

Category: Document/View Framework

See also

Document/View Framework, Document/View Sample, wxFrame

Public Member Functions

- wxDocParentFrame ()

  Default constructor.
- wxDocParentFrame (wxDocManager *manager, wxFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor.
- virtual ~wxDocParentFrame ()

  Destructor.
- bool Create (wxDocManager *manager, wxFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=541072960, const wxString &name=wxFrameNameStr)

  Used in two-step construction.
- wxDocManager * GetDocumentManager () const

  Returns the associated document manager object.

Additional Inherited Members

21.205.2 Constructor & Destructor Documentation

wxDocParentFrame::wxDocParentFrame ( )

Default constructor.

wxDocParentFrame::wxDocParentFrame (wxDocManager * *manager*, wxFrame * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor.

virtual wxDocParentFrame::~wxDocParentFrame ( ) `[virtual]`

Destructor.

21.205.3 Member Function Documentation

bool wxDocParentFrame::Create (wxDocManager * *manager*, wxFrame * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 541072960, const wxString & *name* = wxFrameNameStr)

Used in two-step construction.

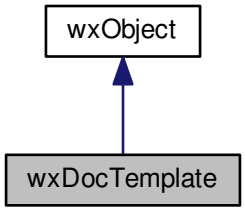
wxDocManager* wxDocParentFrame::GetDocumentManager ( ) const

Returns the associated document manager object.

21.206 wxDocTemplate Class Reference

`#include <wx/docview.h>`

Inheritance diagram for wxDocTemplate:



21.206.1 Detailed Description

The wxDocTemplate class is used to model the relationship between a document class and a view class.

Library: wxCore

Category: Document/View Framework

See also

wxDocTemplate Overview, wxDocument, wxView

Public Member Functions

- wxDocTemplate (wxDocManager *manager, const wxString &descr, const wxString &filter, const wxString &dir, const wxString &ext, const wxString &docTypeName, const wxString &viewTypeName, wxClassInfo *docClassInfo=0, wxClassInfo *viewClassInfo=0, long flags=wxTEMPLATE_VISIBLE)

  Constructor.
- virtual ~wxDocTemplate ()

Destructor.

- virtual wxDocument * CreateDocument (const wxString &path, long flags=0)

  Creates a new instance of the associated document class.
- virtual wxView * CreateView (wxDocument *doc, long flags=0)

  Creates a new instance of the associated view class.
- virtual bool FileMatchesTemplate (const wxString &path)

  This function implements the default (very primitive) format detection which checks if the extension is that of the template.
- wxString GetDefaultExtension () const

  Returns the default file extension for the document data, as passed to the document template constructor.
- wxString GetDescription () const

  Returns the text description of this template, as passed to the document template constructor.
- wxString GetDirectory () const

  Returns the default directory, as passed to the document template constructor.
- wxClassInfo * GetDocClassInfo () const

  Returns the run-time class information that allows document instances to be constructed dynamically, as passed to the document template constructor.
- wxDocManager * GetDocumentManager () const

  Returns a pointer to the document manager instance for which this template was created.
- virtual wxString GetDocumentName () const

  Returns the document type name, as passed to the document template constructor.
- wxString GetFileFilter () const

  Returns the file filter, as passed to the document template constructor.
- long GetFlags () const

  Returns the flags, as passed to the document template constructor.
- wxClassInfo * GetViewClassInfo () const

  Returns the run-time class information that allows view instances to be constructed dynamically, as passed to the document template constructor.
- virtual wxString GetViewName () const

  Returns the view type name, as passed to the document template constructor.
- virtual bool InitDocument (wxDocument *doc, const wxString &path, long flags=0)

  Initialises the document, calling wxDocument::OnCreate().
- bool IsVisible () const

  Returns true if the document template can be shown in user dialogs, false otherwise.
- void SetDefaultExtension (const wxString &ext)

  Sets the default file extension.
- void SetDescription (const wxString &descr)

  Sets the template description.
- void SetDirectory (const wxString &dir)

  Sets the default directory.
- void SetDocumentManager (wxDocManager *manager)

  Sets the pointer to the document manager instance for which this template was created.
- void SetFileFilter (const wxString &filter)

  Sets the file filter.
- void SetFlags (long flags)

  Sets the internal document template flags (see the constructor description for more details).

- wxPageSetupDialogData & GetPageSetupDialogData ()

  Returns a reference to the wxPageSetupDialogData associated with the printing operations of this document manager.
- const wxPageSetupDialogData & GetPageSetupDialogData () const

  Returns a reference to the wxPageSetupDialogData associated with the printing operations of this document manager.

Public Attributes

- wxString m_defaultExt

  The default extension for files of this type.
- wxString m_description

  A short description of this template.
- wxString m_directory

  The default directory for files of this type.
- wxClassInfo * m_docClassInfo

  Run-time class information that allows document instances to be constructed dynamically.
- wxString m_docTypeName

  The named type of the document associated with this template.
- wxDocTemplate * m_documentManager

  A pointer to the document manager for which this template was created.
- wxString m_fileFilter

  The file filter (such as ".txt") to be used in file selector dialogs.*
- long m_flags

  The flags passed to the constructor.
- wxClassInfo * m_viewClassInfo

  Run-time class information that allows view instances to be constructed dynamically.
- wxString m_viewTypeName

  The named type of the view associated with this template.

Additional Inherited Members

21.206.2 Constructor & Destructor Documentation

wxDocTemplate::wxDocTemplate (wxDocManager * *manager,* const wxString & *descr,* const wxString & *filter,* const wxString & *dir,* const wxString & *ext,* const wxString & *docTypeName,* const wxString & *viewTypeName,* wxClassInfo * *docClassInfo* = `0`, wxClassInfo * *viewClassInfo* = `0`, long *flags* = `wxTEMPLATE_VISIBLE`)

Constructor.

Create instances dynamically near the start of your application after creating a wxDocManager instance, and before doing any document or view operations.

Parameters

manager	The document manager object which manages this template.
descr	A short description of what the template is for. This string will be displayed in the file filter list of Windows file selectors.
filter	An appropriate file filter such as "*.txt".
dir	The default directory to use for file selectors.
ext	The default file extension (such as "txt").
docTypeName	A name that should be unique for a given type of document, used for gathering a list of views relevant to a particular document.
viewTypeName	A name that should be unique for a given view.
docClassInfo	A pointer to the run-time document class information as returned by the wxCLASSINFO() macro, e.g. wxCLASSINFO(MyDocumentClass). If this is not supplied, you will need to derive a new wxDocTemplate class and override the CreateDocument() member to return a new document instance on demand.

viewClassInfo	A pointer to the run-time view class information as returned by the wxCLASSINFO() macro, e.g. wxCLASSINFO(MyViewClass). If this is not supplied, you will need to derive a new wxDocTemplate class and override the CreateView() member to return a new view instance on demand.
flags	A bit list of the following: • wxTEMPLATE_VISIBLE - The template may be displayed to the user in dialogs. • wxTEMPLATE_INVISIBLE - The template may not be displayed to the user in dialogs. • wxDEFAULT_TEMPLATE_FLAGS - Defined as wxTEMPLATE_VISIBLE.

wxPerl Note:

In wxPerl *docClassInfo* and *viewClassInfo* can be either `Wx::ClassInfo` objects or strings containing the name of the perl packages which are to be used as `Wx::Document` and `Wx::View` classes (they must have a constructor named new); as an example:

- Wx::DocTemplate->new(docmgr, descr, filter, dir, ext, docTypeName, viewTypeName, docClassInfo, viewClassInfo, flags): will construct document and view objects from the class information.
- Wx::DocTemplate->new(docmgr, descr, filter, dir, ext, docTypeName, viewTypeName, docClassName, viewClassName, flags): will construct document and view objects from perl packages.
- Wx::DocTemplate->new(docmgr, descr, filter, dir, ext, docTypeName, viewTypeName): in this case `Wx::DocTemplate::CreateDocument()` and `Wx::DocTemplate::CreateView()` must be overridden

virtual wxDocTemplate::~wxDocTemplate ( ) [virtual]

Destructor.

21.206.3 Member Function Documentation

virtual wxDocument* wxDocTemplate::CreateDocument (const wxString & *path*, long *flags* = 0) [virtual]

Creates a new instance of the associated document class.

If you have not supplied a wxClassInfo parameter to the template constructor, you will need to override this function to return an appropriate document instance.

This function calls InitDocument() which in turns calls wxDocument::OnCreate().

virtual wxView* wxDocTemplate::CreateView (wxDocument * *doc*, long *flags* = 0) [virtual]

Creates a new instance of the associated view class.

If you have not supplied a wxClassInfo parameter to the template constructor, you will need to override this function to return an appropriate view instance.

If the new view initialization fails, it must call wxDocument::RemoveView() for consistency with the default behaviour of this function.

virtual bool wxDocTemplate::FileMatchesTemplate (const wxString & *path*) [virtual]

This function implements the default (very primitive) format detection which checks if the extension is that of the template.

Parameters

path	The path to be checked against the template.

wxString wxDocTemplate::GetDefaultExtension ( ) const

Returns the default file extension for the document data, as passed to the document template constructor.

wxString wxDocTemplate::GetDescription ( ) const

Returns the text description of this template, as passed to the document template constructor.

wxString wxDocTemplate::GetDirectory ( ) const

Returns the default directory, as passed to the document template constructor.

wxClassInfo* wxDocTemplate::GetDocClassInfo ( ) const

Returns the run-time class information that allows document instances to be constructed dynamically, as passed to the document template constructor.

wxDocManager* wxDocTemplate::GetDocumentManager ( ) const

Returns a pointer to the document manager instance for which this template was created.

virtual wxString wxDocTemplate::GetDocumentName ( ) const `[virtual]`

Returns the document type name, as passed to the document template constructor.

wxString wxDocTemplate::GetFileFilter ( ) const

Returns the file filter, as passed to the document template constructor.

long wxDocTemplate::GetFlags ( ) const

Returns the flags, as passed to the document template constructor.

wxPageSetupDialogData& wxDocTemplate::GetPageSetupDialogData ( )

Returns a reference to the wxPageSetupDialogData associated with the printing operations of this document manager.

const wxPageSetupDialogData& wxDocTemplate::GetPageSetupDialogData ( ) const

Returns a reference to the wxPageSetupDialogData associated with the printing operations of this document manager.

wxClassInfo* wxDocTemplate::GetViewClassInfo ( ) const

Returns the run-time class information that allows view instances to be constructed dynamically, as passed to the document template constructor.

virtual wxString wxDocTemplate::GetViewName ( ) const [virtual]

Returns the view type name, as passed to the document template constructor.

virtual bool wxDocTemplate::InitDocument (wxDocument * *doc,* const wxString & *path,* long *flags* = 0) [virtual]

Initialises the document, calling wxDocument::OnCreate().

This is called from CreateDocument().

If you override this method, notice that you must *delete* the *doc* if its initialization fails for consistency with the default behaviour.

Parameters

doc	The document to initialize.
path	The associated file path.
flags	Flags passed to CreateDocument().

Returns

true if the initialization was successful or false if it failed in which case *doc* should be deleted by this function.

bool wxDocTemplate::IsVisible ( ) const

Returns true if the document template can be shown in user dialogs, false otherwise.

void wxDocTemplate::SetDefaultExtension (const wxString & *ext*)

Sets the default file extension.

void wxDocTemplate::SetDescription (const wxString & *descr*)

Sets the template description.

void wxDocTemplate::SetDirectory (const wxString & *dir*)

Sets the default directory.

void wxDocTemplate::SetDocumentManager (wxDocManager * *manager*)

Sets the pointer to the document manager instance for which this template was created.

Should not be called by the application.

void wxDocTemplate::SetFileFilter (const wxString & *filter*)

Sets the file filter.

void wxDocTemplate::SetFlags (long *flags*)

Sets the internal document template flags (see the constructor description for more details).

21.206.4 Member Data Documentation

wxString wxDocTemplate::m_defaultExt

The default extension for files of this type.

wxString wxDocTemplate::m_description

A short description of this template.

wxString wxDocTemplate::m_directory

The default directory for files of this type.

wxClassInfo* wxDocTemplate::m_docClassInfo

Run-time class information that allows document instances to be constructed dynamically.

wxString wxDocTemplate::m_docTypeName

The named type of the document associated with this template.

wxDocTemplate* wxDocTemplate::m_documentManager

A pointer to the document manager for which this template was created.

wxString wxDocTemplate::m_fileFilter

The file filter (such as "*.txt") to be used in file selector dialogs.

long wxDocTemplate::m_flags

The flags passed to the constructor.

wxClassInfo* wxDocTemplate::m_viewClassInfo

Run-time class information that allows view instances to be constructed dynamically.

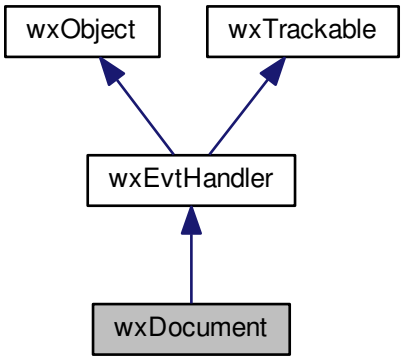
wxString wxDocTemplate::m_viewTypeName

The named type of the view associated with this template.

21.207 wxDocument Class Reference

`#include <wx/docview.h>`

Inheritance diagram for wxDocument:



21.207.1 Detailed Description

The document class can be used to model an application's file-based data.

It is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocTemplate and wxDocManager classes.

A normal document is the one created without parent document and is associated with a disk file. Since version 2.9.2 wxWidgets also supports a special kind of documents called *child documents* which are virtual in the sense that they do not correspond to a file but rather to a part of their parent document. Because of this, the child documents can't be created directly by user but can only be created by the parent document (usually when it's being created itself). They also can't be independently saved. A child document has its own view with the corresponding window. This view can be closed by user but, importantly, is also automatically closed when its parent document is closed. Thus, child documents may be convenient for creating additional windows which need to be closed when the main document is. The docview sample demonstrates this use of child documents by creating a child document containing the information about the parameters of the image opened in the main document.

Library: wxCore

Category: Document/View Framework

See also

Document/View Framework, wxView, wxDocTemplate, wxDocManager

Public Member Functions

- wxDocument (wxDocument *parent=NULL)

  Constructor.
- virtual ~wxDocument ()

Destructor.

- virtual bool AddView (wxView ∗view)

  If the view is not already in the list of views, adds the view and calls OnChangedViewList().
- bool AlreadySaved () const

  Returns true if the document hasn't been modified since the last time it had been saved.
- void Activate () const

  Activate the first view of the document if any.
- virtual bool Close ()

  Closes the document, by calling OnSaveModified() and then (if this returned true) OnCloseDocument().
- virtual bool DeleteAllViews ()

  Calls wxView::Close() and deletes each view.
- virtual bool DeleteContents ()

  Virtual method called from OnCloseDocument().
- virtual wxCommandProcessor ∗ GetCommandProcessor () const

  Returns a pointer to the command processor associated with this document.
- virtual wxDocManager ∗ GetDocumentManager () const

  Gets a pointer to the associated document manager.
- wxString GetDocumentName () const

  Gets the document type name for this document.
- bool GetDocumentSaved () const

  Return true if this document had been already saved.
- virtual wxDocTemplate ∗ GetDocumentTemplate () const

  Gets a pointer to the template that created the document.
- virtual wxWindow ∗ GetDocumentWindow () const

  Intended to return a suitable window for using as a parent for document-related dialog boxes.
- wxString GetFilename () const

  Gets the filename associated with this document, or "" if none is associated.
- wxView ∗ GetFirstView () const

  A convenience function to get the first view for a document, because in many cases a document will only have a single view.
- wxString GetTitle () const

  Gets the title for this document.
- virtual wxString GetUserReadableName () const

  Return the document name suitable to be shown to the user.
- wxViewVector GetViewsVector () const

  Returns a vector of wxView pointers.
- bool IsChildDocument () const

  Returns true if this document is a child document corresponding to a part of the parent document and not a disk file as usual.
- virtual bool IsModified () const

  Returns true if the document has been modified since the last save, false otherwise.
- virtual void Modify (bool modify)

  Call with true to mark the document as modified since the last save, false otherwise.
- virtual void OnChangedViewList ()

  Called when a view is added to or deleted from this document.
- virtual bool OnCloseDocument ()

  This virtual function is called when the document is being closed.
- virtual bool OnCreate (const wxString &path, long flags)

  Called just after the document object is created to give it a chance to initialize itself.
- virtual wxCommandProcessor ∗ OnCreateCommandProcessor ()

  Override this function if you want a different (or no) command processor to be created when the document is created.

- virtual bool OnNewDocument ()

  The default implementation calls OnSaveModified() and DeleteContents(), makes a default title for the document, and notifies the views that the filename (in fact, the title) has changed.
- virtual bool OnOpenDocument (const wxString &filename)

  Constructs an input file stream for the given filename (which must not be empty), and calls LoadObject().
- virtual bool OnSaveDocument (const wxString &filename)

  Constructs an output file stream for the given filename (which must not be empty), and calls SaveObject().
- virtual bool OnSaveModified ()

  If the document has been modified, prompts the user to ask if the changes should be saved.
- virtual bool RemoveView (wxView ∗view)

  Removes the view from the document's list of views, and calls OnChangedViewList().
- virtual bool Save ()

  Saves the document by calling OnSaveDocument() if there is an associated filename, or SaveAs() if there is no filename.
- virtual bool SaveAs ()

  Prompts the user for a file to save to, and then calls OnSaveDocument().
- virtual bool Revert ()

  Discard changes and load last saved version.
- virtual void SetCommandProcessor (wxCommandProcessor ∗processor)

  Sets the command processor to be used for this document.
- void SetDocumentName (const wxString &name)

  Sets the document type name for this document.
- virtual void SetDocumentTemplate (wxDocTemplate ∗templ)

  Sets the pointer to the template that created the document.
- void SetDocumentSaved (bool saved=true)

  Sets if this document has been already saved or not.
- void SetFilename (const wxString &filename, bool notifyViews=false)

  Sets the filename for this document.
- virtual void OnChangeFilename (bool notifyViews)

  If notifyViews is true, wxView::OnChangeFilename() is called for all views.
- void SetTitle (const wxString &title)

  Sets the title for this document.
- virtual void UpdateAllViews (wxView ∗sender=NULL, wxObject ∗hint=NULL)

  Updates all views.

- wxList & GetViews ()

  Returns the list whose elements are the views on the document.
- const wxList & GetViews () const

  Returns the list whose elements are the views on the document.

- virtual istream & LoadObject (istream &stream)

  Override this function and call it from your own LoadObject() before streaming your own data.
- virtual wxInputStream & LoadObject (wxInputStream &stream)

  Override this function and call it from your own LoadObject() before streaming your own data.

- virtual ostream & SaveObject (ostream &stream)

  Override this function and call it from your own SaveObject() before streaming your own data.
- virtual wxOutputStream & SaveObject (wxOutputStream &stream)

  Override this function and call it from your own SaveObject() before streaming your own data.

Protected Member Functions

- virtual bool DoSaveDocument (const wxString &file)

  This method is called by OnSaveDocument() to really save the document contents to the specified file.
- virtual bool DoOpenDocument (const wxString &file)

  This method is called by OnOpenDocument() to really load the document contents from the specified file.

Protected Attributes

- wxCommandProcessor * m_commandProcessor

  A pointer to the command processor associated with this document.
- wxString m_documentFile

  Filename associated with this document ("" if none).
- bool m_documentModified

  true if the document has been modified, false otherwise.
- wxDocTemplate * m_documentTemplate

  A pointer to the template from which this document was created.
- wxString m_documentTitle

  Document title.
- wxString m_documentTypeName

  The document type name given to the wxDocTemplate constructor, copied to this variable when the document is created.
- wxList m_documentViews

  List of wxView instances associated with this document.

Additional Inherited Members

21.207.2 Constructor & Destructor Documentation

wxDocument::wxDocument (wxDocument * *parent* = NULL)

Constructor.

Define your own default constructor to initialize application-specific data.

Parameters

parent	Specifying a non-NULL parent document here makes this document a special *child document*, see their description in the class documentation. Notice that this parameter exists but is ignored in wxWidgets versions prior to 2.9.1.

virtual wxDocument::~wxDocument ( ) [virtual]

Destructor.

Removes itself from the document manager.

21.207.3 Member Function Documentation

void wxDocument::Activate ( ) const

Activate the first view of the document if any.

This function simply calls the Raise() method of the frame of the first view. You may need to override the Raise() method to get the desired effect if you are not using a standard wxFrame for your view. For instance, if your document is inside its own notebook tab you could implement Raise() like this:

```
void MyNotebookPage::Raise()
{
    wxNotebook* notebook = wxStaticCast(GetParent(),
      wxNotebook);
    notebook->SetSelection(notebook->FindPage(this));
}
```

See also

GetFirstView()

Since

2.9.5

virtual bool wxDocument::AddView (wxView * *view*) [virtual]

If the view is not already in the list of views, adds the view and calls OnChangedViewList().

bool wxDocument::AlreadySaved ( ) const

Returns true if the document hasn't been modified since the last time it had been saved.

Notice that this function returns false if the document had been never saved at all, so it may be also used to test whether it makes sense to save the document: if it returns true, there is nothing to save but if false is returned, it can be saved, even if it might be not modified (this can be used to create an empty document file by the user).

See also

IsModified(), GetDocumentSaved()

Since

2.9.0

virtual bool wxDocument::Close ( ) [virtual]

Closes the document, by calling OnSaveModified() and then (if this returned true) OnCloseDocument().

This does not normally delete the document object, use DeleteAllViews() to do this implicitly.

virtual bool wxDocument::DeleteAllViews ( ) [virtual]

Calls wxView::Close() and deletes each view.

Deleting the final view will implicitly delete the document itself, because the wxView destructor calls RemoveView(). This in turns calls OnChangedViewList(), whose default implemention is to save and delete the document if no views exist.

virtual bool wxDocument::DeleteContents ( ) [virtual]

Virtual method called from OnCloseDocument().

This method may be overridden to perform any additional cleanup which might be needed when the document is closed.

The return value of this method is currently ignored.

The default version does nothing and simply returns true.

virtual bool wxDocument::DoOpenDocument (const wxString & *file*) [protected], [virtual]

This method is called by OnOpenDocument() to really load the document contents from the specified file.

Base class version creates a file-based stream and calls LoadObject(). Override this if you need to do something else or prefer not to use LoadObject() at all.

virtual bool wxDocument::DoSaveDocument (const wxString & *file*) [protected], [virtual]

This method is called by OnSaveDocument() to really save the document contents to the specified file.

Base class version creates a file-based stream and calls SaveObject(). Override this if you need to do something else or prefer not to use SaveObject() at all.

virtual wxCommandProcessor* wxDocument::GetCommandProcessor ( ) const [virtual]

Returns a pointer to the command processor associated with this document.

See also

wxCommandProcessor

virtual wxDocManager* wxDocument::GetDocumentManager ( ) const [virtual]

Gets a pointer to the associated document manager.

wxString wxDocument::GetDocumentName ( ) const

Gets the document type name for this document.

See the comment for m_documentTypeName.

bool wxDocument::GetDocumentSaved ( ) const

Return true if this document had been already saved.

See also

IsModified()

virtual wxDocTemplate* wxDocument::GetDocumentTemplate ( ) const [virtual]

Gets a pointer to the template that created the document.

virtual wxWindow* wxDocument::GetDocumentWindow ( ) const `[virtual]`

Intended to return a suitable window for using as a parent for document-related dialog boxes.

By default, uses the frame associated with the first view.

wxString wxDocument::GetFilename ( ) const

Gets the filename associated with this document, or "" if none is associated.

wxView* wxDocument::GetFirstView ( ) const

A convenience function to get the first view for a document, because in many cases a document will only have a single view.

See also

GetViews()

wxString wxDocument::GetTitle ( ) const

Gets the title for this document.

The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

virtual wxString wxDocument::GetUserReadableName ( ) const `[virtual]`

Return the document name suitable to be shown to the user.

The default implementation uses the document title, if any, of the name part of the document filename if it was set or, otherwise, the string **unnamed**.

wxList& wxDocument::GetViews ( )

Returns the list whose elements are the views on the document.

See also

GetFirstView()

const wxList& wxDocument::GetViews ( ) const

Returns the list whose elements are the views on the document.

See also

GetFirstView()

wxViewVector wxDocument::GetViewsVector ( ) const

Returns a vector of wxView pointers.

Since

2.9.5

bool wxDocument::IsChildDocument ( ) const

Returns true if this document is a child document corresponding to a part of the parent document and not a disk file as usual.

This method can be used to check whether file-related operations make sense for this document as they only apply to top-level documents and not child ones.

Since

2.9.2

virtual bool wxDocument::IsModified ( ) const [virtual]

Returns true if the document has been modified since the last save, false otherwise.

You may need to override this if your document view maintains its own record of being modified.

See also

Modify()

virtual istream& wxDocument::LoadObject (istream & *stream*) [virtual]

Override this function and call it from your own LoadObject() before streaming your own data.

LoadObject() is called by the framework automatically when the document contents need to be loaded.

Note

This version of LoadObject() may not exist depending on how wxWidgets was configured.

virtual wxInputStream& wxDocument::LoadObject (wxInputStream & *stream*) [virtual]

Override this function and call it from your own LoadObject() before streaming your own data.

LoadObject() is called by the framework automatically when the document contents need to be loaded.

Note

This version of LoadObject() may not exist depending on how wxWidgets was configured.

virtual void wxDocument::Modify (bool *modify*) [virtual]

Call with true to mark the document as modified since the last save, false otherwise.

You may need to override this if your document view maintains its own record of being modified.

See also

IsModified()

virtual void wxDocument::OnChangedViewList ( ) [virtual]

Called when a view is added to or deleted from this document.

The default implementation saves and deletes the document if no views exist (the last one has just been removed).

virtual void wxDocument::OnChangeFilename (bool *notifyViews*) `[virtual]`

If *notifyViews* is true, wxView::OnChangeFilename() is called for all views.

Since

2.9.0

virtual bool wxDocument::OnCloseDocument ( ) `[virtual]`

This virtual function is called when the document is being closed.

The default implementation calls DeleteContents() (which may be overridden to perform additional cleanup) and sets the modified flag to false. You can override it to supply additional behaviour when the document is closed with Close().

Notice that previous wxWidgets versions used to call this function also from OnNewDocument(), rather counter-intuitively. This is no longer the case since wxWidgets 2.9.0.

virtual bool wxDocument::OnCreate (const wxString & *path,* long *flags*) `[virtual]`

Called just after the document object is created to give it a chance to initialize itself.

The default implementation uses the template associated with the document to create an initial view.

For compatibility reasons, this method may either delete the document itself if its initialization fails or not do it in which case it is deleted by caller. It is recommended to delete the document explicitly in this function if it can't be initialized.

Parameters

path	The associated file path.
flags	Flags passed to CreateDocument().

Returns

true if the initialization was successful or false if it failed.

virtual wxCommandProcessor* wxDocument::OnCreateCommandProcessor ( ) `[virtual]`

Override this function if you want a different (or no) command processor to be created when the document is created.

By default, it returns an instance of wxCommandProcessor.

See also

wxCommandProcessor

virtual bool wxDocument::OnNewDocument ( ) `[virtual]`

The default implementation calls OnSaveModified() and DeleteContents(), makes a default title for the document, and notifies the views that the filename (in fact, the title) has changed.

virtual bool wxDocument::OnOpenDocument (const wxString & *filename*) `[virtual]`

Constructs an input file stream for the given filename (which must not be empty), and calls LoadObject().

If LoadObject() returns true, the document is set to unmodified; otherwise, an error message box is displayed. The document's views are notified that the filename has changed, to give windows an opportunity to update their titles. All of the document's views are then updated.

virtual bool wxDocument::OnSaveDocument (const wxString & *filename*) `[virtual]`

Constructs an output file stream for the given filename (which must not be empty), and calls SaveObject().

If SaveObject() returns true, the document is set to unmodified; otherwise, an error message box is displayed.

virtual bool wxDocument::OnSaveModified ( ) `[virtual]`

If the document has been modified, prompts the user to ask if the changes should be saved.

If the user replies Yes, the Save() function is called. If No, the document is marked as unmodified and the function succeeds. If Cancel, the function fails.

virtual bool wxDocument::RemoveView (wxView * *view*) `[virtual]`

Removes the view from the document's list of views, and calls OnChangedViewList().

virtual bool wxDocument::Revert ( ) `[virtual]`

Discard changes and load last saved version.

Prompts the user first, and then calls DoOpenDocument() to reload the current file.

virtual bool wxDocument::Save ( ) `[virtual]`

Saves the document by calling OnSaveDocument() if there is an associated filename, or SaveAs() if there is no filename.

virtual bool wxDocument::SaveAs ( ) `[virtual]`

Prompts the user for a file to save to, and then calls OnSaveDocument().

virtual ostream& wxDocument::SaveObject (ostream & *stream*) `[virtual]`

Override this function and call it from your own SaveObject() before streaming your own data.

SaveObject() is called by the framework automatically when the document contents need to be saved.

Note

This version of SaveObject() may not exist depending on how wxWidgets was configured.

virtual wxOutputStream& wxDocument::SaveObject (wxOutputStream & *stream*) `[virtual]`

Override this function and call it from your own SaveObject() before streaming your own data.

SaveObject() is called by the framework automatically when the document contents need to be saved.

Note

This version of SaveObject() may not exist depending on how wxWidgets was configured.

virtual void wxDocument::SetCommandProcessor (wxCommandProcessor * *processor*) [virtual]

Sets the command processor to be used for this document.

The document will then be responsible for its deletion. Normally you should not call this; override OnCreate↩ CommandProcessor() instead.

See also

wxCommandProcessor

void wxDocument::SetDocumentName (const wxString & *name*)

Sets the document type name for this document.

See the comment for m_documentTypeName.

void wxDocument::SetDocumentSaved (bool *saved* = true)

Sets if this document has been already saved or not.

Normally there is no need to call this function as the document-view framework does it itself as the documents are loaded from and saved to the files. However it may be useful in some particular cases, for example it may be called with false argument to prevent the user from saving the just opened document into the same file if this shouldn't be done for some reason (e.g. file format version changes and a new extension should be used for saving).

See also

GetDocumentSaved(), AlreadySaved()

virtual void wxDocument::SetDocumentTemplate (wxDocTemplate * *templ*) [virtual]

Sets the pointer to the template that created the document.

Should only be called by the framework.

void wxDocument::SetFilename (const wxString & *filename,* bool *notifyViews* = false)

Sets the filename for this document.

Usually called by the framework.

Calls OnChangeFilename() which in turn calls wxView::OnChangeFilename() for all views if *notifyViews* is true.

void wxDocument::SetTitle (const wxString & *title*)

Sets the title for this document.

The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

virtual void wxDocument::UpdateAllViews (wxView * *sender* = NULL, wxObject * *hint* = NULL) [virtual]

Updates all views.

If *sender* is non-NULL, does not update this view. *hint* represents optional information to allow a view to optimize its update.

21.207.4 Member Data Documentation

wxCommandProcessor* **wxDocument::m_commandProcessor** [protected]

A pointer to the command processor associated with this document.

wxString wxDocument::m_documentFile [protected]

Filename associated with this document ("" if none).

bool wxDocument::m_documentModified [protected]

true if the document has been modified, false otherwise.

wxDocTemplate* **wxDocument::m_documentTemplate** [protected]

A pointer to the template from which this document was created.

wxString wxDocument::m_documentTitle [protected]

Document title.

The document title is used for an associated frame (if any), and is usually constructed by the framework from the filename.

wxString wxDocument::m_documentTypeName [protected]

The document type name given to the wxDocTemplate constructor, copied to this variable when the document is created.

If several document templates are created that use the same document type, this variable is used in wxDocManager::CreateView() to collate a list of alternative view types that can be used on this kind of document. Do not change the value of this variable.

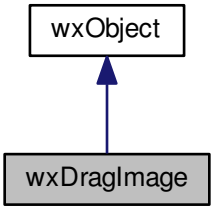
wxList wxDocument::m_documentViews [protected]

List of wxView instances associated with this document.

21.208 wxDragImage Class Reference

`#include <wx/dragimag.h>`

Inheritance diagram for wxDragImage:



21.208.1 Detailed Description

This class is used when you wish to drag an object on the screen, and a simple cursor is not enough.

On Windows, the Win32 API is used to achieve smooth dragging. On other platforms, wxGenericDragImage is used. Applications may also prefer to use wxGenericDragImage on Windows, too.

To use this class, when you wish to start dragging an image, create a wxDragImage object and store it somewhere you can access it as the drag progresses. Call BeginDrag() to start, and EndDrag() to stop the drag. To move the image, initially call Show() and then Move(). If you wish to update the screen contents during the drag (for example, highlight an item as in the dragimag sample), first call Hide(), update the screen, call Move(), and then call Show().

You can drag within one window, or you can use full-screen dragging either across the whole screen, or just restricted to one area of the screen to save resources. If you want the user to drag between two windows, then you will need to use full-screen dragging.

If you wish to draw the image yourself, use wxGenericDragImage and override DoDrawImage() and GetImage↩ Rect().

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag Image Sample

Public Member Functions

- wxDragImage ()

  Default constructor.
- wxDragImage (const wxBitmap &image, const wxCursor &cursor=wxNullCursor)

  Constructs a drag image from a bitmap and optional cursor.
- wxDragImage (const wxIcon &image, const wxCursor &cursor=wxNullCursor)

  Constructs a drag image from an icon and optional cursor.
- wxDragImage (const wxString &text, const wxCursor &cursor=wxNullCursor)

  Constructs a drag image from a text string and optional cursor.
- wxDragImage (const wxTreeCtrl &treeCtrl, wxTreeItemId &id)

Constructs a drag image from the text in the given tree control item, and optional cursor.

- wxDragImage (const wxListCtrl &listCtrl, long id)

  Constructs a drag image from the text in the given list control item, and optional cursor.
- bool BeginDrag (const wxPoint &hotspot, wxWindow *window, bool fullScreen=false, wxRect *rect=NULL)

  Start dragging the image, in a window or full screen.
- bool BeginDrag (const wxPoint &hotspot, wxWindow *window, wxWindow *boundingWindow)

  Start dragging the image, using the first window to capture the mouse and the second to specify the bounding area.
- virtual bool DoDrawImage (wxDC &dc, const wxPoint &pos) const

  Draws the image on the device context with top-left corner at the given position.
- bool EndDrag ()

  Call this when the drag has finished.
- virtual wxRect GetImageRect (const wxPoint &pos) const

  Returns the rectangle enclosing the image, assuming that the image is drawn with its top-left corner at the given point.
- bool Hide ()

  Hides the image.
- bool Move (const wxPoint &pt)

  Call this to move the image to a new position.
- bool Show ()

  Shows the image.
- virtual bool UpdateBackingFromWindow (wxDC &windowDC, wxMemoryDC &destDC, const wxRect &sourceRect, const wxRect &destRect) const

  Override this if you wish to draw the window contents to the backing bitmap yourself.

Additional Inherited Members

21.208.2 Constructor & Destructor Documentation

wxDragImage::wxDragImage ( )

Default constructor.

wxDragImage::wxDragImage (const wxBitmap & *image*, const wxCursor & *cursor* = wxNullCursor)

Constructs a drag image from a bitmap and optional cursor.

Parameters

image	Bitmap to be used as the drag image. The bitmap can have a mask.
cursor	Optional cursor to combine with the image.

wxDragImage::wxDragImage (const wxIcon & *image*, const wxCursor & *cursor* = wxNullCursor)

Constructs a drag image from an icon and optional cursor.

Parameters

image	Icon to be used as the drag image.
cursor	Optional cursor to combine with the image.

wxDragImage::wxDragImage (const wxString & *text*, const wxCursor & *cursor* = wxNullCursor)

Constructs a drag image from a text string and optional cursor.

Parameters

text	Text used to construct a drag image.
cursor	Optional cursor to combine with the image.

wxDragImage::wxDragImage (const wxTreeCtrl & *treeCtrl*, wxTreeItemId & *id*)

Constructs a drag image from the text in the given tree control item, and optional cursor.

Parameters

treeCtrl	Tree control for constructing a tree drag image.
id	Tree control item id.

wxDragImage::wxDragImage (const wxListCtrl & *listCtrl*, long *id*)

Constructs a drag image from the text in the given list control item, and optional cursor.

Parameters

listCtrl	List control for constructing a list drag image.
id	List control item id.

21.208.3 Member Function Documentation

bool wxDragImage::BeginDrag (const wxPoint & *hotspot*, wxWindow * *window*, bool *fullScreen* = `false`, wxRect * *rect* = `NULL`)

Start dragging the image, in a window or full screen.

You need to then call Show() and Move() to show the image on the screen. Call EndDrag() when the drag has finished.

Note that this call automatically calls CaptureMouse().

Parameters

hotspot	The location of the drag position relative to the upper-left corner of the image.
window	The window that captures the mouse, and within which the dragging is limited unless fullScreen is true.
fullScreen	If true, specifies that the drag will be visible over the full screen, or over as much of the screen as is specified by rect. Note that the mouse will still be captured in window.
rect	If non-NULL, specifies the rectangle (in screen coordinates) that bounds the dragging operation. Specifying this can make the operation more efficient by cutting down on the area under consideration, and it can also make a visual difference since the drag is clipped to this area.

bool wxDragImage::BeginDrag (const wxPoint & *hotspot*, wxWindow * *window*, wxWindow * *boundingWindow*)

Start dragging the image, using the first window to capture the mouse and the second to specify the bounding area.

This form is equivalent to using the first form, but more convenient than working out the bounding rectangle explicitly.

You need to then call Show() and Move() to show the image on the screen. Call EndDrag() when the drag has finished.

Note that this call automatically calls CaptureMouse().

Parameters

hotspot	The location of the drag position relative to the upper-left corner of the image.
window	The window that captures the mouse, and within which the dragging is limited.
boundingWindow	Specifies the area within which the drag occurs.

virtual bool wxDragImage::DoDrawImage (wxDC & *dc,* const wxPoint & *pos*) const [virtual]

Draws the image on the device context with top-left corner at the given position.

This function is only available with wxGenericDragImage, to allow applications to draw their own image instead of using an actual bitmap. If you override this function, you must also override GetImageRect().

bool wxDragImage::EndDrag ( )

Call this when the drag has finished.

Note

This function automatically releases mouse capture.

virtual wxRect wxDragImage::GetImageRect (const wxPoint & *pos*) const [virtual]

Returns the rectangle enclosing the image, assuming that the image is drawn with its top-left corner at the given point.

This function is available in wxGenericDragImage only, and may be overridden (together with DoDrawImage()) to provide a virtual drawing capability.

bool wxDragImage::Hide ( )

Hides the image.

You may wish to call this before updating the window contents (perhaps highlighting an item). Then call Move() and Show().

bool wxDragImage::Move (const wxPoint & *pt*)

Call this to move the image to a new position.

The image will only be shown if Show() has been called previously (for example at the start of the drag).

Parameters

pt	The position in client coordinates (relative to the window specified in BeginDrag()).

You can move the image either when the image is hidden or shown, but in general dragging will be smoother if you move the image when it is shown.

bool wxDragImage::Show ( )

Shows the image.

Call this at least once when dragging.

virtual bool wxDragImage::UpdateBackingFromWindow (wxDC & *windowDC,* wxMemoryDC & *destDC,* const wxRect & *sourceRect,* const wxRect & *destRect*) const `[virtual]`

Override this if you wish to draw the window contents to the backing bitmap yourself.

This can be desirable if you wish to avoid flicker by not having to redraw the updated window itself just before dragging, which can cause a flicker just as the drag starts. Instead, paint the drag image's backing bitmap to show the appropriate graphic *minus* the objects to be dragged, and leave the window itself to be updated by the drag image. This can provide eerily smooth, flicker-free drag behaviour.

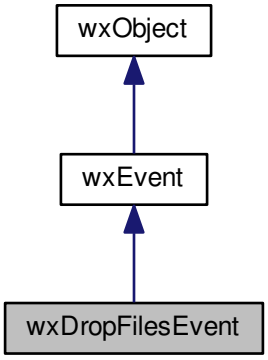
The default implementation copies the window contents to the backing bitmap. A new implementation will normally copy information from another source, such as from its own backing bitmap if it has one, or directly from internal data structures.

This function is available in wxGenericDragImage only.

21.209 wxDropFilesEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxDropFilesEvent:



21.209.1 Detailed Description

This class is used for drop files events, that is, when files have been dropped onto the window.

This functionality is currently only available under Windows.

The window must have previously been enabled for dropping by calling wxWindow::DragAcceptFiles().

Important note: this is a separate implementation to the more general drag and drop implementation documented in the Drag and Drop Overview. It uses the older, Windows message-based approach of dropping files.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxDropFilesEvent& event)

Event macros:

- EVT_DROP_FILES(func): Process a `wxEVT_DROP_FILES` event.

Availability: only available for the wxMSW port.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxDropFilesEvent (wxEventType id=0, int noFiles=0, wxString *files=NULL)

  Constructor.
- wxString * GetFiles () const

  Returns an array of filenames.
- int GetNumberOfFiles () const

  Returns the number of files dropped.
- wxPoint GetPosition () const

  Returns the position at which the files were dropped.

Additional Inherited Members

21.209.2 Constructor & Destructor Documentation

wxDropFilesEvent::wxDropFilesEvent (wxEventType *id* = `0`, int *noFiles* = `0`, wxString * *files* = `NULL`)

Constructor.

21.209.3 Member Function Documentation

wxString* wxDropFilesEvent::GetFiles ( ) const

Returns an array of filenames.

int wxDropFilesEvent::GetNumberOfFiles ( ) const

Returns the number of files dropped.

wxPoint wxDropFilesEvent::GetPosition ( ) const

Returns the position at which the files were dropped.

Returns an array of filenames.

21.210 wxDropSource Class Reference

```
#include <wx/dnd.h>
```

21.210.1 Detailed Description

This class represents a source for a drag and drop operation.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject Overview, wxDropTarget, wxTextDropTarget, wxFileDropTarget

Public Member Functions

- wxDropSource (wxWindow *win=NULL, const wxCursor &iconCopy=wxNullCursor, const wxCursor &iconMove=wxNullCursor, const wxCursor &iconNone=wxNullCursor)

  This constructor requires that you must call SetData() later.
- wxDropSource (wxDataObject &data, wxWindow *win=NULL, const wxCursor &iconCopy=wxNullCursor, const wxCursor &iconMove=wxNullCursor, const wxCursor &iconNone=wxNullCursor)

  The constructor taking a wxDataObject.
- wxDropSource (wxWindow *win=NULL, const wxIcon &iconCopy=wxNullIcon, const wxIcon &iconMove=wxNullIcon, const wxIcon &iconNone=wxNullIcon)

  This constructor requires that you must call SetData() later.
- wxDropSource (wxDataObject &data, wxWindow *win=NULL, const wxIcon &iconCopy=wxNullIcon, const wxIcon &iconMove=wxNullIcon, const wxIcon &iconNone=wxNullIcon)

  The constructor taking a wxDataObject.
- virtual wxDragResult DoDragDrop (int flags=wxDrag_CopyOnly)

  Starts the drag-and-drop operation which will terminate when the user releases the mouse.
- wxDataObject * GetDataObject ()

  Returns the wxDataObject object that has been assigned previously.
- virtual bool GiveFeedback (wxDragResult effect)

  You may give some custom UI feedback during the drag and drop operation by overriding this function.
- void SetCursor (wxDragResult res, const wxCursor &cursor)

  Set the icon to use for a certain drag result.
- void SetIcon (wxDragResult res, const wxIcon &icon)

  Set the icon to use for a certain drag result.
- void SetData (wxDataObject &data)

  Sets the data wxDataObject associated with the drop source.

21.210.2 Constructor & Destructor Documentation

wxDropSource::wxDropSource (wxWindow * *win* = NULL, const wxCursor & *iconCopy* = wxNullCursor, const wxCursor & *iconMove* = wxNullCursor, const wxCursor & *iconNone* = wxNullCursor)

This constructor requires that you must call SetData() later.

Note that the type of *iconCopy* and subsequent parameters differs between different ports: these are cursors under Windows and OS X but icons for GTK. You should use the macro wxDROP_ICON() in portable programs instead of directly using either of these types.

Availability: only available for the wxMSW, wxOSX ports.

Parameters

win	The window which initiates the drag and drop operation.
iconCopy	The icon or cursor used for feedback for copy operation.
iconMove	The icon or cursor used for feedback for move operation.
iconNone	The icon or cursor used for feedback when operation can't be done.

wxDropSource::wxDropSource (wxDataObject & *data*, wxWindow * *win* = NULL, const wxCursor & *iconCopy* = wxNullCursor, const wxCursor & *iconMove* = wxNullCursor, const wxCursor & *iconNone* = wxNullCursor)

The constructor taking a wxDataObject.

Note that the type of *iconCopy* and subsequent parameters differs between different ports: these are cursors under Windows and OS X but icons for GTK. You should use the macro wxDROP_ICON() in portable programs instead of directly using either of these types.

Availability: only available for the wxMSW, wxOSX ports.

Parameters

data	The data associated with the drop source.
win	The window which initiates the drag and drop operation.
iconCopy	The icon or cursor used for feedback for copy operation.
iconMove	The icon or cursor used for feedback for move operation.
iconNone	The icon or cursor used for feedback when operation can't be done.

wxDropSource::wxDropSource (wxWindow * *win* = NULL, const wxIcon & *iconCopy* = wxNullIcon, const wxIcon & *iconMove* = wxNullIcon, const wxIcon & *iconNone* = wxNullIcon)

This constructor requires that you must call SetData() later.

This is the wxGTK-specific version of the constructor taking wxIcon instead of wxCursor as the other ports.

Availability: only available for the wxGTK port.

Parameters

win	The window which initiates the drag and drop operation.
iconCopy	The icon or cursor used for feedback for copy operation.
iconMove	The icon or cursor used for feedback for move operation.
iconNone	The icon or cursor used for feedback when operation can't be done.

wxDropSource::wxDropSource (wxDataObject & *data*, wxWindow * *win* = NULL, const wxIcon & *iconCopy* = wxNullIcon, const wxIcon & *iconMove* = wxNullIcon, const wxIcon & *iconNone* = wxNullIcon)

The constructor taking a wxDataObject.

This is the wxGTK-specific version of the constructor taking wxIcon instead of wxCursor as the other ports.

Availability: only available for the wxGTK port.

Parameters

data	The data associated with the drop source.
win	The window which initiates the drag and drop operation.

iconCopy	The icon or cursor used for feedback for copy operation.
iconMove	The icon or cursor used for feedback for move operation.
iconNone	The icon or cursor used for feedback when operation can't be done.

21.210.3 Member Function Documentation

virtual wxDragResult wxDropSource::DoDragDrop (int *flags* = wxDrag_CopyOnly) [virtual]

Starts the drag-and-drop operation which will terminate when the user releases the mouse.

Call this in response to a mouse button press, for example.

Parameters

flags	If wxDrag_AllowMove is included in the flags, data may be moved and not only copied as is the case for the default wxDrag_CopyOnly. If wxDrag_DefaultMove is specified (which includes the previous flag), moving is not only possible but becomes the default operation.

Returns

The operation requested by the user, may be wxDragCopy, wxDragMove, wxDragLink, wxDragCancel or wxDragNone if an error occurred.

wxDataObject* wxDropSource::GetDataObject ( )

Returns the wxDataObject object that has been assigned previously.

virtual bool wxDropSource::GiveFeedback (wxDragResult *effect*) [virtual]

You may give some custom UI feedback during the drag and drop operation by overriding this function.

It is called on each mouse move, so your implementation must not be too slow.

Parameters

effect	The effect to implement. One of wxDragCopy, wxDragMove, wxDragLink and wxDragNone.

Returns

false if you want default feedback, or true if you implement your own feedback. The return value is ignored under GTK.

void wxDropSource::SetCursor (wxDragResult *res*, const wxCursor & *cursor*)

Set the icon to use for a certain drag result.

Parameters

res	The drag result to set the icon for.
cursor	The icon to show when this drag result occurs.

Availability: only available for the wxMSW, wxOSX ports.

void wxDropSource::SetData (wxDataObject & *data*)

Sets the data wxDataObject associated with the drop source.

This will not delete any previously associated data.

void wxDropSource::SetIcon (wxDragResult *res,* const wxIcon & *icon*)

Set the icon to use for a certain drag result.

Parameters

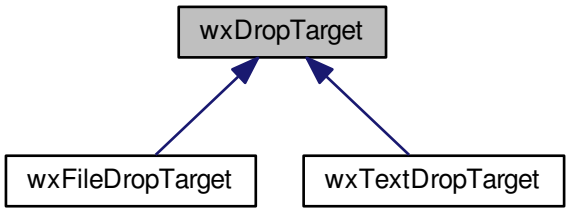
res	The drag result to set the icon for.
icon	The icon to show when this drag result occurs.

Availability: only available for the wxGTK port.

21.211 wxDropTarget Class Reference

`#include <wx/dnd.h>`

Inheritance diagram for wxDropTarget:



21.211.1 Detailed Description

This class represents a target for a drag and drop operation.

A wxDataObject can be associated with it and by default, this object will be filled with the data from the drag source, if the data formats supported by the data object match the drag source data format.

There are various virtual handler functions defined in this class which may be overridden to give visual feedback or react in a more fine-tuned way, e.g. by not accepting data on the whole window area, but only a small portion of it. The normal sequence of calls is OnEnter(), OnDragOver() possibly many times, OnDrop() and finally OnData().

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject Overview, wxDropSource, wxTextDropTarget, wxFileDropTarget, wxDataFormat, wxDataObject

Public Member Functions

- wxDropTarget (wxDataObject *data=NULL)

  Constructor.

- virtual ~wxDropTarget ()

  Destructor.
- virtual bool GetData ()

  This method may only be called from within OnData().
- virtual wxDragResult OnData (wxCoord x, wxCoord y, wxDragResult defResult)=0

  Called after OnDrop() returns true.
- virtual wxDragResult OnDragOver (wxCoord x, wxCoord y, wxDragResult defResult)

  Called when the mouse is being dragged over the drop target.
- virtual bool OnDrop (wxCoord x, wxCoord y)

  Called when the user drops a data object on the target.
- virtual wxDragResult OnEnter (wxCoord x, wxCoord y, wxDragResult defResult)

  Called when the mouse enters the drop target.
- virtual void OnLeave ()

  Called when the mouse leaves the drop target.
- wxDataObject * GetDataObject () const

  Returns the data wxDataObject associated with the drop target.
- void SetDataObject (wxDataObject *data)

  Sets the data wxDataObject associated with the drop target and deletes any previously associated data object.
- void SetDefaultAction (wxDragResult action)

  Sets the default action for drag and drop.
- wxDragResult GetDefaultAction ()

  Returns default action for drag and drop or wxDragNone if this not specified.

21.211.2 Constructor & Destructor Documentation

wxDropTarget::wxDropTarget (wxDataObject * *data* = NULL)

Constructor.

data is the data to be associated with the drop target.

virtual wxDropTarget::~wxDropTarget ( ) [virtual]

Destructor.

Deletes the associated data object, if any.

21.211.3 Member Function Documentation

virtual bool wxDropTarget::GetData ( ) [virtual]

This method may only be called from within OnData().

By default, this method copies the data from the drop source to the wxDataObject associated with this drop target, calling its wxDataObject::SetData() method.

wxDataObject* wxDropTarget::GetDataObject ( ) const

Returns the data wxDataObject associated with the drop target.

wxDragResult wxDropTarget::GetDefaultAction ( )

Returns default action for drag and drop or wxDragNone if this not specified.

virtual wxDragResult wxDropTarget::OnData (wxCoord *x*, wxCoord *y*, wxDragResult *defResult*) `[pure virtual]`

Called after OnDrop() returns true.

By default this will usually GetData() and will return the suggested default value *defResult*.

virtual wxDragResult wxDropTarget::OnDragOver (wxCoord *x*, wxCoord *y*, wxDragResult *defResult*) `[virtual]`

Called when the mouse is being dragged over the drop target.

By default, this calls functions return the suggested return value *defResult*.

Parameters

x	The x coordinate of the mouse.
y	The y coordinate of the mouse.
defResult	Suggested value for return value. Determined by SHIFT or CONTROL key states.

Returns

The desired operation or wxDragNone. This is used for optical feedback from the side of the drop source, typically in form of changing the icon.

virtual bool wxDropTarget::OnDrop (wxCoord *x*, wxCoord *y*) `[virtual]`

Called when the user drops a data object on the target.

Return false to veto the operation.

Parameters

x	The x coordinate of the mouse.
y	The y coordinate of the mouse.

Returns

true to accept the data, or false to veto the operation.

Reimplemented in wxFileDropTarget, and wxTextDropTarget.

virtual wxDragResult wxDropTarget::OnEnter (wxCoord *x*, wxCoord *y*, wxDragResult *defResult*) `[virtual]`

Called when the mouse enters the drop target.

By default, this calls OnDragOver().

Parameters

x	The x coordinate of the mouse.
y	The y coordinate of the mouse.
defResult	Suggested default for return value. Determined by SHIFT or CONTROL key states.

Returns

The desired operation or wxDragNone. This is used for optical feedback from the side of the drop source, typically in form of changing the icon.

virtual void wxDropTarget::OnLeave ( ) [virtual]

Called when the mouse leaves the drop target.

void wxDropTarget::SetDataObject (wxDataObject * *data*)

Sets the data wxDataObject associated with the drop target and deletes any previously associated data object.

void wxDropTarget::SetDefaultAction (wxDragResult *action*)

Sets the default action for drag and drop.

Use wxDragMove or wxDragCopy to set default action to move or copy and use wxDragNone (default) to set default action specified by initialization of draging (see wxDropSource::DoDragDrop())

21.212 wxDynamicLibrary Class Reference

```
#include <wx/dynlib.h>
```

21.212.1 Detailed Description

wxDynamicLibrary is a class representing dynamically loadable library (Windows DLL, shared library under Unix etc).

Just create an object of this class to load a library and don't worry about unloading it – it will be done in the objects destructor automatically.

The following flags can be used with wxDynamicLibrary() or Load():

Styles

This class supports the following styles:

- wxDL_LAZY: Equivalent of RTLD_LAZY under Unix, ignored elsewhere.
- wxDL_NOW: Equivalent of RTLD_NOW under Unix, ignored elsewhere.
- wxDL_GLOBAL: Equivalent of RTLD_GLOBAL under Unix, ignored elsewhere.
- wxDL_VERBATIM: Don't try to append the appropriate extension to the library name (this is done by default).
- wxDL_DEFAULT: Default flags, same as wxDL_NOW currently.
- wxDL_QUIET: Don't log an error message if the library couldn't be loaded.

Library: wxBase

Category: Application and Process Management

Public Member Functions

- wxDynamicLibrary ()

  Default constructor.
- wxDynamicLibrary (const wxString &name, int flags=wxDL_DEFAULT)

  Constructor.
- wxDllType Detach ()

  Detaches this object from its library handle, i.e. the object will not unload the library any longer in its destructor but it is now the callers responsibility to do this using Unload().
- void * GetSymbol (const wxString &name, bool *success=0) const

  Returns pointer to symbol name in the library or NULL if the library contains no such symbol.
- void * GetSymbolAorW (const wxString &name) const

  This function is available only under Windows as it is only useful when dynamically loading symbols from standard Windows DLLs.
- bool HasSymbol (const wxString &name) const

  Returns true if the symbol with the given name is present in the dynamic library, false otherwise.
- bool IsLoaded () const

  Returns true if the library was successfully loaded, false otherwise.
- bool Load (const wxString &name, int flags=wxDL_DEFAULT)

  Loads DLL with the given name into memory.
- void Unload ()

  Unloads the library from memory.

Static Public Member Functions

- static wxString CanonicalizeName (const wxString &name, wxDynamicLibraryCategory cat=wxDL_LIBRARY)

  Returns the platform-specific full name for the library called name.
- static wxString CanonicalizePluginName (const wxString &name, wxPluginCategory cat=wxDL_PLUGIN_GUI)

  This function does the same thing as CanonicalizeName() but for wxWidgets plugins.
- static wxDllType GetProgramHandle ()

  Return a valid handle for the main program itself or NULL if symbols from the main program can't be loaded on this platform.
- static wxDynamicLibraryDetailsArray ListLoaded ()

  This static method returns a wxArray containing the details of all modules loaded into the address space of the current project.
- static void * GetModuleFromAddress (const void *addr, wxString *path=NULL)

  Returns the load address of the module containing the specified address or NULL if not found.
- static void Unload (wxDllType handle)

  Unloads the library from memory.

21.212.2 Constructor & Destructor Documentation

wxDynamicLibrary::wxDynamicLibrary ( )

Default constructor.

wxDynamicLibrary::wxDynamicLibrary (const wxString & *name*, int *flags* = `wxDL_DEFAULT`)

Constructor.

Calls Load() with the given *name*.

21.212.3 Member Function Documentation

static wxString wxDynamicLibrary::CanonicalizeName (const wxString & *name,* wxDynamicLibraryCategory *cat* = wxDL_LIBRARY) [static]

Returns the platform-specific full name for the library called *name*.

E.g. it adds a "`.dll`" extension under Windows and "`lib`" prefix and "`.so`", "`.sl`" or "`.dylib`" extension under Unix.

See also

CanonicalizePluginName()

static wxString wxDynamicLibrary::CanonicalizePluginName (const wxString & *name,* wxPluginCategory *cat* = wxDL_PLUGIN_GUI) [static]

This function does the same thing as CanonicalizeName() but for wxWidgets plugins.

The only difference is that compiler and version information are added to the name to ensure that the plugin which is going to be loaded will be compatible with the main program.

wxDllType wxDynamicLibrary::Detach ( )

Detaches this object from its library handle, i.e. the object will not unload the library any longer in its destructor but it is now the callers responsibility to do this using Unload().

static void* wxDynamicLibrary::GetModuleFromAddress (const void * *addr,* wxString * *path* = `NULL`) [static]

Returns the load address of the module containing the specified address or NULL if not found.

If the second argument *path* is not NULL, it is filled with the full path to the file the module was loaded from upon a successful return.

This method is implemented under MSW and Unix platforms providing `dladdr()` call (which include Linux and various BSD systems) and always returns NULL elsewhere.

Since

3.1.0

static wxDllType wxDynamicLibrary::GetProgramHandle ( ) [static]

Return a valid handle for the main program itself or NULL if symbols from the main program can't be loaded on this platform.

void* wxDynamicLibrary::GetSymbol (const wxString & *name,* bool * *success* = `0`) const

Returns pointer to symbol *name* in the library or NULL if the library contains no such symbol.

See also

wxDYNLIB_FUNCTION()

void* wxDynamicLibrary::GetSymbolAorW (const wxString & *name*) const

This function is available only under Windows as it is only useful when dynamically loading symbols from standard Windows DLLs.

Such functions have either `'A'` (in ANSI build) or `'W'` (in Unicode, or wide character build) suffix if they take string parameters. Using this function, you can use just the base name of the function and the correct suffix is appended automatically depending on the current build. Otherwise, this method is identical to GetSymbol().

Availability: only available for the wxMSW port.

bool wxDynamicLibrary::HasSymbol (const wxString & *name*) const

Returns true if the symbol with the given *name* is present in the dynamic library, false otherwise.

Unlike GetSymbol(), this function doesn't log an error message if the symbol is not found.

Since

2.5.4

bool wxDynamicLibrary::IsLoaded ( ) const

Returns true if the library was successfully loaded, false otherwise.

static wxDynamicLibraryDetailsArray wxDynamicLibrary::ListLoaded ( ) [static]

This static method returns a wxArray containing the details of all modules loaded into the address space of the current project.

The array elements are objects of the type: wxDynamicLibraryDetails. The array will be empty if an error occurred.

This method is currently implemented only under Win32 and Linux and is useful mostly for diagnostics purposes.

bool wxDynamicLibrary::Load (const wxString & *name*, int *flags* = `wxDL_DEFAULT`)

Loads DLL with the given *name* into memory.

The *flags* argument can be a combination of the styles outlined in the class description.

Returns true if the library was successfully loaded, false otherwise.

void wxDynamicLibrary::Unload ( )

Unloads the library from memory.

wxDynamicLibrary object automatically calls this method from its destructor if it had been successfully loaded.

static void wxDynamicLibrary::Unload (wxDllType *handle*) [static]

Unloads the library from memory.

wxDynamicLibrary object automatically calls this method from its destructor if it had been successfully loaded.

This version of Unload() is only used if you need to keep the library in memory during a longer period of time than the scope of the wxDynamicLibrary object. In this case you may call Detach() and store the handle somewhere and call this static method later to unload it.

21.213 wxDynamicLibraryDetails Class Reference

`#include <wx/dynlib.h>`

21.213.1 Detailed Description

This class is used for the objects returned by the wxDynamicLibrary::ListLoaded() method and contains the information about a single module loaded into the address space of the current process.

A module in this context may be either a dynamic library or the main program itself.

Library: wxBase

Category: Application and Process Management

Public Member Functions

- bool GetAddress (void *addr, size_t *len) const

  Retrieves the load address and the size of this module.
- wxString GetName () const

  Returns the base name of this module, e.g. "kernel32.dll" or "libc-2.3.2.so".
- wxString GetPath () const

  Returns the full path of this module if available, e.g. "c:\windows\system32\kernel32.dll" or "/lib/libc-2.3.2.so".
- wxString GetVersion () const

  Returns the version of this module, e.g. "5.2.3790.0" or "2.3.2".

21.213.2 Member Function Documentation

bool wxDynamicLibraryDetails::GetAddress (void * *addr*, size_t * *len*) const

Retrieves the load address and the size of this module.

Parameters

addr	The pointer to the location to return load address in, may be NULL.
len	Pointer to the location to return the size of this module in memory in, may be NULL.

Returns

true if the load address and module size were retrieved, false if this information is not available.

wxString wxDynamicLibraryDetails::GetName ( ) const

Returns the base name of this module, e.g. "kernel32.dll" or "libc-2.3.2.so".

wxString wxDynamicLibraryDetails::GetPath ( ) const

Returns the full path of this module if available, e.g. "c:\windows\system32\kernel32.dll" or "/lib/libc-2.3.2.so".

wxString wxDynamicLibraryDetails::GetVersion ( ) const

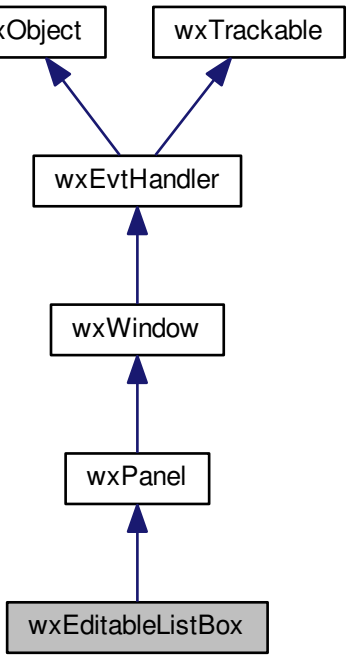
Returns the version of this module, e.g. `"5.2.3790.0"` or `"2.3.2"`.

The returned string is empty if the version information is not available.

21.214 wxEditableListBox Class Reference

`#include <wx/editlbox.h>`

Inheritance diagram for wxEditableListBox:



21.214.1 Detailed Description

An editable listbox is composite control that lets the user easily enter, delete and reorder a list of strings.

Styles

This class supports the following styles:

- wxEL_ALLOW_NEW: Allows the user to enter new strings.
- wxEL_ALLOW_EDIT: Allows the user to edit existing strings.
- wxEL_ALLOW_DELETE: Allows the user to delete existing strings.
- wxEL_NO_REORDER: Does not allow the user to reorder the strings.

- wxEL_DEFAULT_STYLE: Default style: wxEL_ALLOW_NEW|wxEL_ALLOW_EDIT|wxEL_ALLOW_DELETE.

The control uses a wxListCtrl internally and emit its events.

Library: wxAdvanced

Category: Controls

See also

wxListBox, wxListCtrl

Public Member Functions

- wxEditableListBox ()

  Default ctor.
- wxEditableListBox (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxEL_DEFAULT_STYLE, const wxString &name=wxEditableListBoxNameStr)

  Constructor, creating and showing a list box.
- virtual ~wxEditableListBox ()

  Destructor, destroying the list box.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxEL_DEFAULT_STYLE, const wxString &name=wxEditableListBoxNameStr)

  Creates the editable listbox for two-step construction.
- void SetStrings (const wxArrayString &strings)

  Replaces current contents with given strings.
- void GetStrings (wxArrayString &strings) const

  Returns in the given array the current contents of the control (the array will be erased before control's contents are appended).

Additional Inherited Members

21.214.2 Constructor & Destructor Documentation

wxEditableListBox::wxEditableListBox ( )

Default ctor.

wxEditableListBox::wxEditableListBox (wxWindow * *parent,* wxWindowID *id,* const wxString & *label,* const wxPoint & *pos =* wxDefaultPosition, const wxSize & *size =* wxDefaultSize, long *style =* wxEL_DEFAULT_STYLE, const wxString & *name =* `wxEditableListBoxNameStr`)

Constructor, creating and showing a list box.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
label	The text shown just before the list control.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxEditableListBox.
name	Window name.

See also

Create()

virtual wxEditableListBox::~wxEditableListBox ( ) [virtual]

Destructor, destroying the list box.

21.214.3 Member Function Documentation

bool wxEditableListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxEL_DEFAULT_STYLE, const wxString & *name* = wxEditableListBoxNameStr)

Creates the editable listbox for two-step construction.

See wxEditableListBox() for further details.

void wxEditableListBox::GetStrings (wxArrayString & *strings*) const

Returns in the given array the current contents of the control (the array will be erased before control's contents are appended).

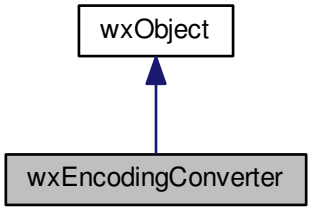
void wxEditableListBox::SetStrings (const wxArrayString & *strings*)

Replaces current contents with given strings.

21.215 wxEncodingConverter Class Reference

`#include <wx/encconv.h>`

Inheritance diagram for wxEncodingConverter:



21.215.1 Detailed Description

This class is capable of converting strings between two 8-bit encodings/charsets.

It can also convert from/to Unicode.

Only a limited subset of encodings is supported by wxEncodingConverter: `wxFONTENCODING_ISO8859_1`..15, `wxFONTENCODING_CP1250`..1257 and `wxFONTENCODING_KOI8`.

Note

Please use wxMBConv classes instead if possible. wxCSConv has much better support for various encodings than wxEncodingConverter. wxEncodingConverter is useful only if you rely on wxCONVERT_SUBSTITUTE mode of operation (see wxEncodingConverter::Init()).

Library: wxBase

Category: Text Conversion

See also

wxFontMapper, wxMBConv, Writing Non-English Applications

Public Member Functions

- wxEncodingConverter ()

  Constructor.
- bool Init (wxFontEncoding input_enc, wxFontEncoding output_enc, int method=wxCONVERT_STRICT)

  Initialize the conversion.

Conversion functions

- bool Convert (const char *input, char *output) const

  Convert input string according to settings passed to Init() and writes the result to output.
- bool Convert (const wchar_t *input, wchar_t *output) const

  Convert input string according to settings passed to Init() and writes the result to output.
- bool Convert (const char *input, wchar_t *output) const

  Convert input string according to settings passed to Init() and writes the result to output.
- bool Convert (const wchar_t *input, char *output) const

Convert input string according to settings passed to Init() and writes the result to output.

- bool Convert (char *str) const

  Convert input string according to settings passed to Init() in-place.

- bool Convert (wchar_t *str) const

  Convert input string according to settings passed to Init() in-place.

- wxString Convert (const wxString &input) const

  Convert a wxString and return a new wxString object.

Static Public Member Functions

- static bool CanConvert (wxFontEncoding encIn, wxFontEncoding encOut)

  Return true if (any text in) multibyte encoding encIn can be converted to another one (encOut) losslessly.

- static wxFontEncodingArray GetAllEquivalents (wxFontEncoding enc)

  Similar to GetPlatformEquivalents(), but this one will return ALL equivalent encodings, regardless of the platform, and including itself.

- static wxFontEncodingArray GetPlatformEquivalents (wxFontEncoding enc, int platform=wxPLATFORM_CURRENT)

  Return equivalents for given font that are used under given platform.

Additional Inherited Members

21.215.2 Constructor & Destructor Documentation

wxEncodingConverter::wxEncodingConverter ( )

Constructor.

21.215.3 Member Function Documentation

static bool wxEncodingConverter::CanConvert (wxFontEncoding *encIn*, wxFontEncoding *encOut*) [static]

Return true if (any text in) multibyte encoding *encIn* can be converted to another one (*encOut*) losslessly.

Do not call this method with `wxFONTENCODING_UNICODE` as either parameter, it doesn't make sense (always works in one sense and always depends on the text to convert in the other).

bool wxEncodingConverter::Convert (const char * *input*, char * *output*) const

Convert input string according to settings passed to Init() and writes the result to output.

All the Convert() function overloads return true if the conversion was lossless and false if at least one of the characters couldn't be converted was and replaced with '?' in the output.

Note that if `wxCONVERT_SUBSTITUTE` was passed to Init(), substitution is considered a lossless operation.

Note

You must call Init() before using this method!

bool wxEncodingConverter::Convert (const wchar_t * *input*, wchar_t * *output*) const

Convert input string according to settings passed to Init() and writes the result to output.

All the Convert() function overloads return true if the conversion was lossless and false if at least one of the characters couldn't be converted was and replaced with '?' in the output.

Note that if `wxCONVERT_SUBSTITUTE` was passed to Init(), substitution is considered a lossless operation.

Note

You must call Init() before using this method!

bool wxEncodingConverter::Convert (const char * *input*, wchar_t * *output*) const

Convert input string according to settings passed to Init() and writes the result to output.

All the Convert() function overloads return true if the conversion was lossless and false if at least one of the characters couldn't be converted was and replaced with '?' in the output.

Note that if `wxCONVERT_SUBSTITUTE` was passed to Init(), substitution is considered a lossless operation.

Note

You must call Init() before using this method!

bool wxEncodingConverter::Convert (const wchar_t * *input*, char * *output*) const

Convert input string according to settings passed to Init() and writes the result to output.

All the Convert() function overloads return true if the conversion was lossless and false if at least one of the characters couldn't be converted was and replaced with '?' in the output.

Note that if `wxCONVERT_SUBSTITUTE` was passed to Init(), substitution is considered a lossless operation.

Note

You must call Init() before using this method!

bool wxEncodingConverter::Convert (char * *str*) const

Convert input string according to settings passed to Init() in-place.

With this overload, the conversion result is written to the same memory area from which the input is read.

See the Convert(const char*,char*) const overload for more info.

bool wxEncodingConverter::Convert (wchar_t * *str*) const

Convert input string according to settings passed to Init() in-place.

With this overload, the conversion result is written to the same memory area from which the input is read.

See the Convert(const wchar_t*,wchar_t*) const overload for more info.

wxString wxEncodingConverter::Convert (const wxString & *input*) const

Convert a wxString and return a new wxString object.

See the Convert(const char*,char*) const overload for more info.

static wxFontEncodingArray wxEncodingConverter::GetAllEquivalents (wxFontEncoding *enc*) [static]

Similar to GetPlatformEquivalents(), but this one will return ALL equivalent encodings, regardless of the platform, and including itself.

This platform's encodings are before others in the array. And again, if *enc* is in the array, it is the very first item in it.

static wxFontEncodingArray wxEncodingConverter::GetPlatformEquivalents (wxFontEncoding *enc,* int *platform =* `wxPLATFORM_CURRENT`) [static]

Return equivalents for given font that are used under given platform.

Supported platforms:

- wxPLATFORM_UNIX
- wxPLATFORM_WINDOWS
- wxPLATFORM_MAC
- wxPLATFORM_CURRENT

wxPLATFORM_CURRENT means the platform this binary was compiled for.

Examples:

```
current platform   enc          returned value
-----------------------------------------------
unix            CP1250             {ISO8859_2}
unix         ISO8859_2             {ISO8859_2}
windows      ISO8859_2                {CP1250}
unix            CP1252  {ISO8859_1,ISO8859_15}
```

Equivalence is defined in terms of convertibility: two encodings are equivalent if you can convert text between then without losing information (it may - and will - happen that you lose special chars like quotation marks or em-dashes but you shouldn't lose any diacritics and language-specific characters when converting between equivalent encodings).

Remember that this function does **NOT** check for presence of fonts in system. It only tells you what are most suitable encodings. (It usually returns only one encoding.)

Note

Note that argument enc itself may be present in the returned array, so that you can, as a side-effect, detect whether the encoding is native for this platform or not.
Convert() is not limited to converting between equivalent encodings, it can convert between two arbitrary encodings.
If *enc* is present in the returned array, then it is always the first item of it.
Please note that the returned array may contain no items at all.

bool wxEncodingConverter::Init (wxFontEncoding *input_enc,* wxFontEncoding *output_enc,* int *method =* `wxCONVERT_STRICT`)

Initialize the conversion.

Both output or input encoding may be wxFONTENCODING_UNICODE, but only if wxUSE_ENCODING is set to 1.

All subsequent calls to Convert() will interpret its argument as a string in *input_enc* encoding and will output string in *output_enc* encoding.

You must call this method before calling Convert. You may call it more than once in order to switch to another conversion.

method affects behaviour of Convert() in case input character cannot be converted because it does not exist in output encoding:

- **wxCONVERT_STRICT:** follow behaviour of GNU Recode - just copy unconvertible characters to output and don't change them (its integer value will stay the same)

- **wxCONVERT_SUBSTITUTE:** try some (lossy) substitutions - e.g. replace unconvertible latin capitals with acute by ordinary capitals, replace en-dash or em-dash by '-' etc.

Both modes guarantee that output string will have same length as input string.

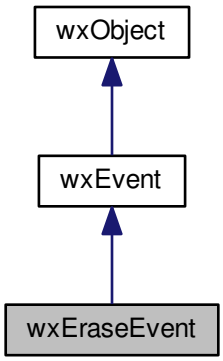
Returns

false if given conversion is impossible, true otherwise (conversion may be impossible either if you try to convert to Unicode with non-Unicode build of wxWidgets or if input or output encoding is not supported).

21.216 wxEraseEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxEraseEvent:



21.216.1 Detailed Description

An erase event is sent when a window's background needs to be repainted.

On some platforms, such as GTK+, this event is simulated (simply generated just before the paint event) and may cause flicker. It is therefore recommended that you set the text background colour explicitly in order to prevent flicker. The default background colour under GTK+ is grey.

To intercept this event, use the EVT_ERASE_BACKGROUND macro in an event table definition.

You must use the device context returned by GetDC() to draw on, don't create a wxPaintDC in the event handler.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxEraseEvent& event)

Event macros:

- EVT_ERASE_BACKGROUND(func): Process a `wxEVT_ERASE_BACKGROUND` event.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxEraseEvent (int id=0, wxDC *dc=NULL)

  Constructor.

- wxDC * GetDC () const

  Returns the device context associated with the erase event to draw on.

Additional Inherited Members

21.216.2 Constructor & Destructor Documentation

wxEraseEvent::wxEraseEvent (int *id* = `0`, wxDC * *dc* = `NULL`)

Constructor.

21.216.3 Member Function Documentation

wxDC* wxEraseEvent::GetDC ( ) const

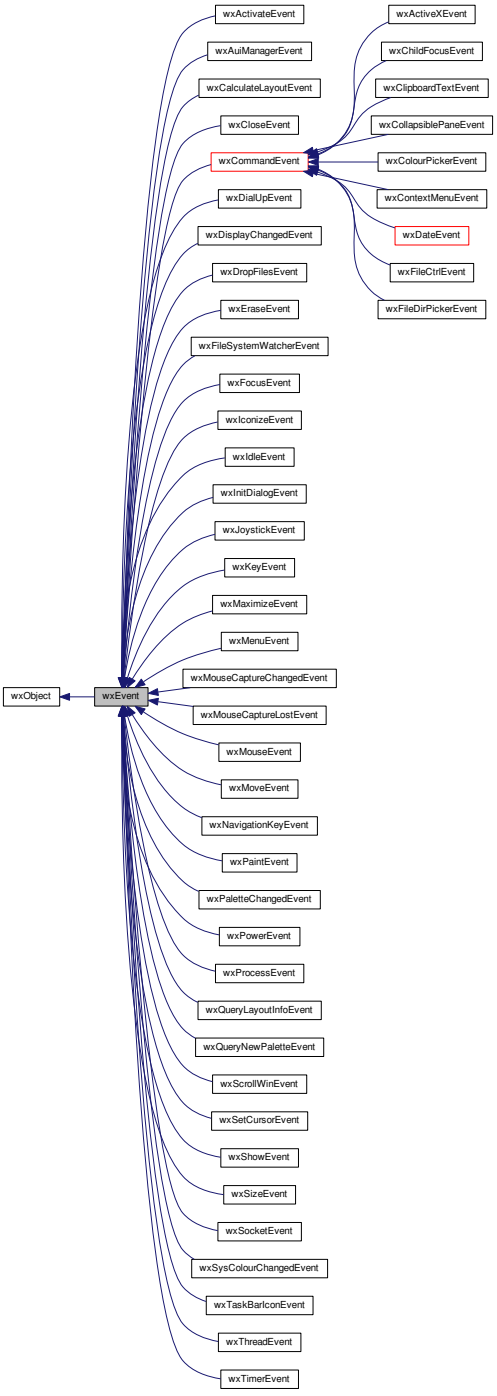
Returns the device context associated with the erase event to draw on.

The returned pointer is never NULL.

21.217 wxEvent Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxEvent:



21.217.1 Detailed Description

An event is a structure holding information about an event passed to a callback or member function.

wxEvent used to be a multipurpose event object, and is an abstract base class for other event classes (see below).

For more information about events, see the Events and Event Handling overview.

wxPerl Note: In wxPerl custom event classes should be derived from `Wx::PlEvent` and `Wx::PlCommand`↩

`Event`.

Library: wxBase

Category: Events

See also

wxCommandEvent, wxMouseEvent

Public Member Functions

- wxEvent (int id=0, wxEventType eventType=wxEVT_NULL)

  Constructor.
- virtual wxEvent * Clone () const =0

  Returns a copy of the event.
- wxObject * GetEventObject () const

  Returns the object (usually a window) associated with the event, if any.
- wxEventType GetEventType () const

  Returns the identifier of the given event type, such as `wxEVT_BUTTON`.
- virtual wxEventCategory GetEventCategory () const

  Returns a generic category for this event.
- int GetId () const

  Returns the identifier associated with this event, such as a button command id.
- wxObject * GetEventUserData () const

  Return the user data associated with a dynamically connected event handler.
- bool GetSkipped () const

  Returns true if the event handler should be skipped, false otherwise.
- long GetTimestamp () const

  Gets the timestamp for the event.
- bool IsCommandEvent () const

  Returns true if the event is or is derived from wxCommandEvent else it returns false.
- void ResumePropagation (int propagationLevel)

  Sets the propagation level to the given value (for example returned from an earlier call to wxEvent::StopPropagation).
- void SetEventObject (wxObject *object)

  Sets the originating object.
- void SetEventType (wxEventType type)

  Sets the event type.
- void SetId (int id)

  Sets the identifier associated with this event, such as a button command id.
- void SetTimestamp (long timeStamp=0)

  Sets the timestamp for the event.
- bool ShouldPropagate () const

  Test if this event should be propagated or not, i.e. if the propagation level is currently greater than 0.
- void Skip (bool skip=true)

  This method can be used inside an event handler to control whether further event handlers bound to this event will be called after the current one returns.
- int StopPropagation ()

  Stop the event from propagating to its parent window.

Protected Attributes

- int m_propagationLevel

  Indicates how many levels the event can propagate.

Additional Inherited Members

21.217.2 Constructor & Destructor Documentation

wxEvent::wxEvent (int *id* = 0, wxEventType *eventType* = wxEVT_NULL)

Constructor.

Notice that events are usually created by wxWidgets itself and creating e.g. a wxPaintEvent in your code and sending it to e.g. a wxTextCtrl will not usually affect it at all as native controls have no specific knowledge about wxWidgets events. However you may construct objects of specific types and pass them to wxEvtHandler::ProcessEvent() if you want to create your own custom control and want to process its events in the same manner as the standard ones.

Also please notice that the order of parameters in this constructor is different from almost all the derived classes which specify the event type as the first argument.

Parameters

id	The identifier of the object (window, timer, ...) which generated this event.
eventType	The unique type of event, e.g. `wxEVT_PAINT`, `wxEVT_SIZE` or `wxEVT_BUTTON`.

21.217.3 Member Function Documentation

virtual wxEvent* wxEvent::Clone ( ) const `[pure virtual]`

Returns a copy of the event.

Any event that is posted to the wxWidgets event system for later action (via wxEvtHandler::AddPendingEvent, wxEvtHandler::QueueEvent or wxPostEvent()) must implement this method.

All wxWidgets events fully implement this method, but any derived events implemented by the user should also implement this method just in case they (or some event derived from them) are ever posted.

All wxWidgets events implement a copy constructor, so the easiest way of implementing the Clone function is to implement a copy constructor for a new event (call it MyEvent) and then define the Clone function like this:

```
wxEvent *Clone() const { return new MyEvent(*this); }
```

Implemented in wxThreadEvent, wxRichTextEvent, wxTextUrlEvent, and wxWindowModalDialogEvent.

virtual wxEventCategory wxEvent::GetEventCategory ( ) const `[virtual]`

Returns a generic category for this event.

wxEvent implementation returns `wxEVT_CATEGORY_UI` by default.

This function is used to selectively process events in wxEventLoopBase::YieldFor.

Reimplemented in wxThreadEvent.

wxObject* wxEvent::GetEventObject ( ) const

Returns the object (usually a window) associated with the event, if any.

wxEventType wxEvent::GetEventType ( ) const

Returns the identifier of the given event type, such as `wxEVT_BUTTON`.

wxObject* wxEvent::GetEventUserData ( ) const

Return the user data associated with a dynamically connected event handler.

wxEvtHandler::Connect() and wxEvtHandler::Bind() allow associating optional `userData` pointer with the handler and this method returns the value of this pointer.

The returned pointer is owned by wxWidgets and must not be deleted.

Since

2.9.5

int wxEvent::GetId ( ) const

Returns the identifier associated with this event, such as a button command id.

bool wxEvent::GetSkipped ( ) const

Returns true if the event handler should be skipped, false otherwise.

long wxEvent::GetTimestamp ( ) const

Gets the timestamp for the event.

The timestamp is the time in milliseconds since some fixed moment (not necessarily the standard Unix Epoch, so only differences between the timestamps and not their absolute values usually make sense).

Warning

wxWidgets returns a non-NULL timestamp only for mouse and key events (see wxMouseEvent and wxKeyEvent).

bool wxEvent::IsCommandEvent ( ) const

Returns true if the event is or is derived from wxCommandEvent else it returns false.

Note

exists only for optimization purposes.

void wxEvent::ResumePropagation (int *propagationLevel*)

Sets the propagation level to the given value (for example returned from an earlier call to wxEvent::StopPropagation).

void wxEvent::SetEventObject (wxObject * *object*)

Sets the originating object.

void wxEvent::SetEventType (wxEventType *type*)

Sets the event type.

void wxEvent::SetId (int *id*)

Sets the identifier associated with this event, such as a button command id.

void wxEvent::SetTimestamp (long *timeStamp* = `0`)

Sets the timestamp for the event.

bool wxEvent::ShouldPropagate ( ) const

Test if this event should be propagated or not, i.e. if the propagation level is currently greater than 0.

void wxEvent::Skip (bool *skip* = `true`)

This method can be used inside an event handler to control whether further event handlers bound to this event will be called after the current one returns.

Without Skip() (or equivalently if Skip(false) is used), the event will not be processed any more. If Skip(true) is called, the event processing system continues searching for a further handler function for this event, even though it has been processed already in the current handler.

In general, it is recommended to skip all non-command events to allow the default handling to take place. The command events are, however, normally not skipped as usually a single command such as a button click or menu item selection must only be processed by one handler.

int wxEvent::StopPropagation ( )

Stop the event from propagating to its parent window.

Returns the old propagation level value which may be later passed to ResumePropagation() to allow propagating the event again.

21.217.4 Member Data Documentation

int wxEvent::m_propagationLevel `[protected]`

Indicates how many levels the event can propagate.

This member is protected and should typically only be set in the constructors of the derived classes. It may be temporarily changed by StopPropagation() and ResumePropagation() and tested with ShouldPropagate().

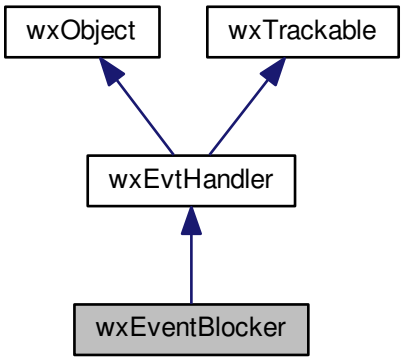
The initial value is set to either `wxEVENT_PROPAGATE_NONE` (by default) meaning that the event shouldn't be propagated at all or to `wxEVENT_PROPAGATE_MAX` (for command events) meaning that it should be propagated as much as necessary.

Any positive number means that the event should be propagated but no more than the given number of times. E.g. the propagation level may be set to 1 to propagate the event to its parent only, but not to its grandparent.

21.218 wxEventBlocker Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxEventBlocker:



21.218.1 Detailed Description

This class is a special event handler which allows to discard any event (or a set of event types) directed to a specific window.

Example:

```
void MyWindow::DoSomething()
{
    {
        // block all events directed to this window while
        // we do the 1000 FunctionWhichSendsEvents() calls
        wxEventBlocker blocker(this);

        for ( int i = 0; i  1000; i++ )
            FunctionWhichSendsEvents(i);

    } // ~wxEventBlocker called, old event handler is restored

    // the event generated by this call will be processed:
    FunctionWhichSendsEvents(0)
}
```

Library: wxCore

Category: Events

See also

How Events are Processed, wxEvtHandler

Public Member Functions

- wxEventBlocker (wxWindow *win, wxEventType type=-1)

  Constructs the blocker for the given window and for the given event type.
- virtual ~wxEventBlocker ()

  Destructor.
- void Block (wxEventType eventType)

  Adds to the list of event types which should be blocked the given eventType.

Additional Inherited Members

21.218.2 Constructor & Destructor Documentation

wxEventBlocker::wxEventBlocker (wxWindow * *win,* wxEventType *type* = -1)

Constructs the blocker for the given window and for the given event type.

If *type* is `wxEVT_ANY`, then all events for that window are blocked. You can call Block() after creation to add other event types to the list of events to block.

Note that the *win* window **must** remain alive until the wxEventBlocker object destruction.

virtual wxEventBlocker::~wxEventBlocker ( ) [virtual]

Destructor.

The blocker will remove itself from the chain of event handlers for the window provided in the constructor, thus restoring normal processing of events.

21.218.3 Member Function Documentation

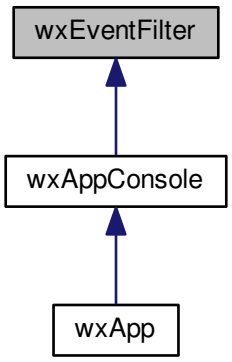
void wxEventBlocker::Block (wxEventType *eventType*)

Adds to the list of event types which should be blocked the given *eventType*.

21.219 wxEventFilter Class Reference

`#include <wx/eventfilter.h>`

Inheritance diagram for wxEventFilter:



21.219.1 Detailed Description

A global event filter for pre-processing all the events generated in the program.

This is a very simple class which just provides FilterEvent() virtual method to be called by wxEvtHandler before starting process of any event. Thus, inheriting from this class and overriding FilterEvent() allows to capture and possibly handle or ignore all the events happening in the program. Of course, having event filters adds additional overhead to every event processing and so should not be used lightly and your FilterEvent() code should try to return as quickly as possible, especially for the events it is not interested in.

An example of using this class:

```
// This class allows to determine the last time the user has worked with
// this application:
class LastActivityTimeDetector : public wxEventFilter
{
public:
    LastActivityTimeDetector()
    {
        wxEvtHandler::AddFilter(this);

        m_last = wxDateTime::Now();
    }

    virtual ~LastActivityTimeDetector()
    {
        wxEvtHandler::RemoveFilter(this);
    }

    virtual int FilterEvent(wxEvent& event)
    {
        // Update the last user activity
        const wxEventType t = event.GetEventType();
        if ( t == wxEVT_KEY_DOWN || t == wxEVT_MOTION ||
                t == wxEVT_LEFT_DOWN ||
                    t == wxEVT_RIGHT_DOWN ||
                        t == wxEVT_MIDDLE_DOWN )
        {
            m_last = wxDateTime::Now();
        }

        // Continue processing the event normally as well.
        return Event_Skip;
    }

    // This function could be called periodically from some timer to
    // do something (e.g. hide sensitive data or log out from remote
    // server) if the user has been inactive for some time period.
    bool IsInactiveFor(const wxTimeSpan& diff) const
    {
        return wxDateTime::Now() - diff > m_last;
    }

private:
    wxDateTime m_last;
};
```

Notice that wxApp derives from wxEventFilter and is registered as an event filter during its creation so you may also override FilterEvent() method in your wxApp-derived class and, in fact, this is often the most convenient way to do it. However creating a new class deriving directly from wxEventFilter allows to isolate the event filtering code in its own separate class and also to have several independent filters, if necessary.

Category: Events

Since

2.9.3

Public Types

- enum {
  Event_Skip = -1,
  Event_Ignore = 0,
  Event_Processed = 1 }

  Possible return values for FilterEvent().

Public Member Functions

- wxEventFilter ()

  Default constructor.

- virtual ~wxEventFilter ()

  Destructor.

- virtual int FilterEvent (wxEvent &event)=0

  Override this method to implement event pre-processing.

21.219.2 Constructor & Destructor Documentation

wxEventFilter::wxEventFilter ( )

Default constructor.

Constructor does not register this filter using wxEvtHandler::AddFilter(), it's your responsibility to do it when necessary.

Notice that the objects of this class can't be copied.

virtual wxEventFilter::~wxEventFilter ( ) `[virtual]`

Destructor.

You must call wxEvtHandler::RemoveFilter() before destroying this object (possibly from the derived class destructor), failure to do this is indicated by an assert unless assertions are disabled.

21.219.3 Member Function Documentation

virtual int wxEventFilter::FilterEvent (wxEvent & *event*) `[pure virtual]`

Override this method to implement event pre-processing.

This method allows to filter all the events processed by the program, so you should try to return quickly from it to avoid slowing down the program to a crawl.

Although the return type of this method is `int`, this is only due to backwards compatibility concerns and the actual return value must be one of the `Event_XXX` constants defined above:

- Event_Skip to continue processing the event normally (this should be used in most cases).
- Event_Ignore to not process this event at all (this can be used to suppress some events).
- Event_Processed to not process this event normally but indicate that it was already processed by the event filter and so no default processing should take place neither (this should only be used if the filter really did process the event).

Implemented in wxAppConsole.

21.220 wxEventLoopActivator Class Reference

```
#include <wx/evtloop.h>
```

21.220.1 Detailed Description

Makes an event loop temporarily active.

This class is used to make the event loop active during its life-time, e.g.:

```
class MyEventLoop : public wxEventLoopBase { ... };

void RunMyLoop()
{
    MyEventLoop loop;
    wxEventLoopActivator activate(&loop);

    ...
} // the previously active event loop restored here
```

Library: wxBase

Category: Application and Process Management

See also

wxEventLoopBase

Public Member Functions

- wxEventLoopActivator (wxEventLoopBase *loop)

  Makes the loop passed as the parameter currently active.

- ~wxEventLoopActivator ()

  Restores the previously active event loop stored by the constructor.

21.220.2 Constructor & Destructor Documentation

wxEventLoopActivator::wxEventLoopActivator (wxEventLoopBase * *loop*)

Makes the loop passed as the parameter currently active.

This saves the current return value of wxEventLoopBase::GetActive() and then calls wxEventLoopBase::SetActive() with the given *loop*.

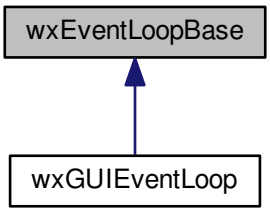
wxEventLoopActivator::~wxEventLoopActivator ( )

Restores the previously active event loop stored by the constructor.

21.221 wxEventLoopBase Class Reference

`#include <wx/evtloop.h>`

Inheritance diagram for wxEventLoopBase:



21.221.1 Detailed Description

Base class for all event loop implementations.

An event loop is a class which queries the queue of native events sent to the wxWidgets application and dispatches them to the appropriate wxEvtHandlers.

An object of this class is created by wxAppTraits::CreateEventLoop() and used by wxApp to run the main application event loop. Temporary event loops are usually created by wxDialog::ShowModal().

You can create your own event loop if you need, provided that you restore the main event loop once yours is destroyed (see wxEventLoopActivator).

Notice that there can be more than one event loop at any given moment, e.g. an event handler called from the main loop can show a modal dialog, which starts its own loop resulting in two nested loops, with the modal dialog being the active one (its IsRunning() returns true). And a handler for a button inside the modal dialog can, of course, create another modal dialog with its own event loop and so on. So in general event loops form a stack and only the event loop at the top of the stack is considered to be active. It is also the only loop that can be directly asked to terminate by calling Exit() (which is done by wxDialog::EndModal()), an outer event loop can't be stopped while an inner one is still running. It is however possible to ask an outer event loop to terminate as soon as all its nested loops exit and the control returns back to it by using ScheduleExit().

Library: wxBase

Category: Application and Process Management

See also

wxApp, wxEventLoopActivator

Public Member Functions

- bool IsMain () const

  Returns true if this is the main loop executed by wxApp::OnRun().

Dispatch and processing

- virtual int Run ()=0

  Start the event loop, return the exit code when it is finished.

- bool IsRunning () const

  Return true if this event loop is currently running.
- virtual bool IsOk () const

  Use this to check whether the event loop was successfully created before using it.
- virtual void Exit (int rc=0)

  Exit the currently running loop with the given exit code.
- virtual void ScheduleExit (int rc=0)=0

  Schedule an exit from the loop with the given exit code.
- virtual bool Pending () const =0

  Return true if any events are available.
- virtual bool Dispatch ()=0

  Dispatches the next event in the windowing system event queue.
- virtual int DispatchTimeout (unsigned long timeout)=0

  Dispatch an event but not wait longer than the specified timeout for it.
- virtual void WakeUp ()=0

  Called by wxWidgets to wake up the event loop even if it is currently blocked inside Dispatch().

Idle handling

- virtual void WakeUpIdle ()

  Makes sure that idle events are sent again.
- virtual bool ProcessIdle ()

  This virtual function is called when the application becomes idle and normally just sends wxIdleEvent to all interested parties.

Yield-related hooks

- virtual bool IsYielding () const

  Returns true if called from inside Yield() or from inside YieldFor().
- bool Yield (bool onlyIfNeeded=false)

  Yields control to pending messages in the windowing system.
- bool YieldFor (long eventsToProcess)

  Works like Yield() with onlyIfNeeded == true, except that it allows the caller to specify a mask of the wxEvent↩ Category values which indicates which events should be processed and which should instead be "delayed" (i.e.
- virtual bool IsEventAllowedInsideYield (wxEventCategory cat) const

  Returns true if the given event category is allowed inside a YieldFor() call (i.e.

Static Public Member Functions

- static wxEventLoopBase * GetActive ()

  Return the currently active (running) event loop.
- static void SetActive (wxEventLoopBase *loop)

  Set currently active (running) event loop.

Protected Member Functions

- virtual void OnExit ()

  This function is called before the event loop terminates, whether this happens normally (because of Exit() call) or abnormally (because of an exception thrown from inside the loop).

21.221.2 Member Function Documentation

virtual bool wxEventLoopBase::Dispatch ( ) [pure virtual]

Dispatches the next event in the windowing system event queue.

Blocks until an event appears if there are none currently (use Pending() if this is not wanted).

This can be used for programming event loops, e.g.

```
while (evtloop->Pending())
    evtloop->Dispatch();
```

Returns

false if the event loop should stop and true otherwise.

See also

Pending(), wxEventLoopBase

virtual int wxEventLoopBase::DispatchTimeout (unsigned long *timeout*) [pure virtual]

Dispatch an event but not wait longer than the specified timeout for it.

If an event is received before the specified *timeout* expires, it is processed and the function returns 1 normally or 0 if the event loop should quite. Otherwise, i.e. if the timeout expires, the functions returns -1 without processing any events.

Parameters

timeout	The maximal time to wait for the events in milliseconds.

Returns

1 if an event was processed, 0 if the event loop should quit or -1 if the timeout expired.

virtual void wxEventLoopBase::Exit (int *rc* = 0) [virtual]

Exit the currently running loop with the given exit code.

The loop will exit, i.e. its Run() method will return, during the next event loop iteration.

Notice that this method can only be used if this event loop is the currently running one, i.e. its IsRunning() returns true. If this is not the case, an assert failure is triggered and nothing is done as outer event loops can't be exited from immediately. Use ScheduleExit() if you'd like to exit this loop even if it doesn't run currently.

static wxEventLoopBase* wxEventLoopBase::GetActive ( ) [static]

Return the currently active (running) event loop.

May return NULL if there is no active event loop (e.g. during application startup or shutdown).

virtual bool wxEventLoopBase::IsEventAllowedInsideYield (wxEventCategory *cat*) const [virtual]

Returns true if the given event category is allowed inside a YieldFor() call (i.e.

compares the given category against the last mask passed to YieldFor()).

See also

wxEvent::GetEventCategory

bool wxEventLoopBase::IsMain ( ) const

Returns true if this is the main loop executed by wxApp::OnRun().

virtual bool wxEventLoopBase::IsOk ( ) const `[virtual]`

Use this to check whether the event loop was successfully created before using it.

bool wxEventLoopBase::IsRunning ( ) const

Return true if this event loop is currently running.

Notice that even if this event loop hasn't terminated yet but has just spawned a nested (e.g. modal) event loop, this method would return false.

virtual bool wxEventLoopBase::IsYielding ( ) const `[virtual]`

Returns true if called from inside Yield() or from inside YieldFor().

virtual void wxEventLoopBase::OnExit ( ) `[protected], [virtual]`

This function is called before the event loop terminates, whether this happens normally (because of Exit() call) or abnormally (because of an exception thrown from inside the loop).

The default implementation calls wxAppConsole::OnEventLoopExit.

virtual bool wxEventLoopBase::Pending ( ) const `[pure virtual]`

Return true if any events are available.

If this method returns true, calling Dispatch() will not block.

virtual bool wxEventLoopBase::ProcessIdle ( ) `[virtual]`

This virtual function is called when the application becomes idle and normally just sends wxIdleEvent to all interested parties.

It should return true if more idle events are needed, false if not.

virtual int wxEventLoopBase::Run ( ) `[pure virtual]`

Start the event loop, return the exit code when it is finished.

Logically, this method calls Dispatch() in a loop until it returns false and also takes care of generating idle events during each loop iteration. However not all implementations of this class really implement it like this (e.g. wxGTK does not) so you shouldn't rely on Dispatch() being called from inside this function.

Returns

The argument passed to Exit() which terminated this event loop.

virtual void wxEventLoopBase::ScheduleExit (int *rc* = 0) [pure virtual]

Schedule an exit from the loop with the given exit code.

This method is similar to Exit() but can be called even if this event loop is not the currently running one – and if it is the active loop, then it works in exactly the same way as Exit().

The loop will exit as soon as the control flow returns to it, i.e. after any nested loops terminate.

Since

2.9.5

static void wxEventLoopBase::SetActive (wxEventLoopBase * *loop*) [static]

Set currently active (running) event loop.

Called by wxEventLoopActivator, use an instance of this class instead of calling this method directly to ensure that the previously active event loop is restored.

Results in a call to wxAppConsole::OnEventLoopEnter.

virtual void wxEventLoopBase::WakeUp ( ) [pure virtual]

Called by wxWidgets to wake up the event loop even if it is currently blocked inside Dispatch().

virtual void wxEventLoopBase::WakeUpIdle ( ) [virtual]

Makes sure that idle events are sent again.

bool wxEventLoopBase::Yield (bool *onlyIfNeeded* = false)

Yields control to pending messages in the windowing system.

This can be useful, for example, when a time-consuming process writes to a text window. Without an occasional yield, the text window will not be updated properly, and on systems with cooperative multitasking, such as Windows 3.1 other processes will not respond.

Caution should be exercised, however, since yielding may allow the user to perform actions which are not compatible with the current task. Disabling menu items or whole menus during processing can avoid unwanted reentrance of code: see wxSafeYield for a better function.

Note that Yield() will not flush the message logs. This is intentional as calling Yield() is usually done to quickly update the screen and popping up a message box dialog may be undesirable. If you do wish to flush the log messages immediately (otherwise it will be done during the next idle loop iteration), call wxLog::FlushActive.

If *onlyIfNeeded* parameter is true and the flow control is already inside Yield(), i.e. IsYielding() returns true, the method just silently returns false and doesn't do anything.

bool wxEventLoopBase::YieldFor (long *eventsToProcess*)

Works like Yield() with *onlyIfNeeded* == true, except that it allows the caller to specify a mask of the wxEvent↩ Category values which indicates which events should be processed and which should instead be "delayed" (i.e.

processed by the main loop later).

Note that this is a safer alternative to Yield() since it ensures that only the events you're interested to will be processed; i.e. this method helps to avoid unwanted reentrancies.

Note that currently only wxMSW and wxGTK do support selective yield of native events coming from the underlying GUI toolkit. wxWidgets events posted using wxEvtHandler::AddPendingEvent or wxEvtHandler::QueueEvent are instead selectively processed by all ports.

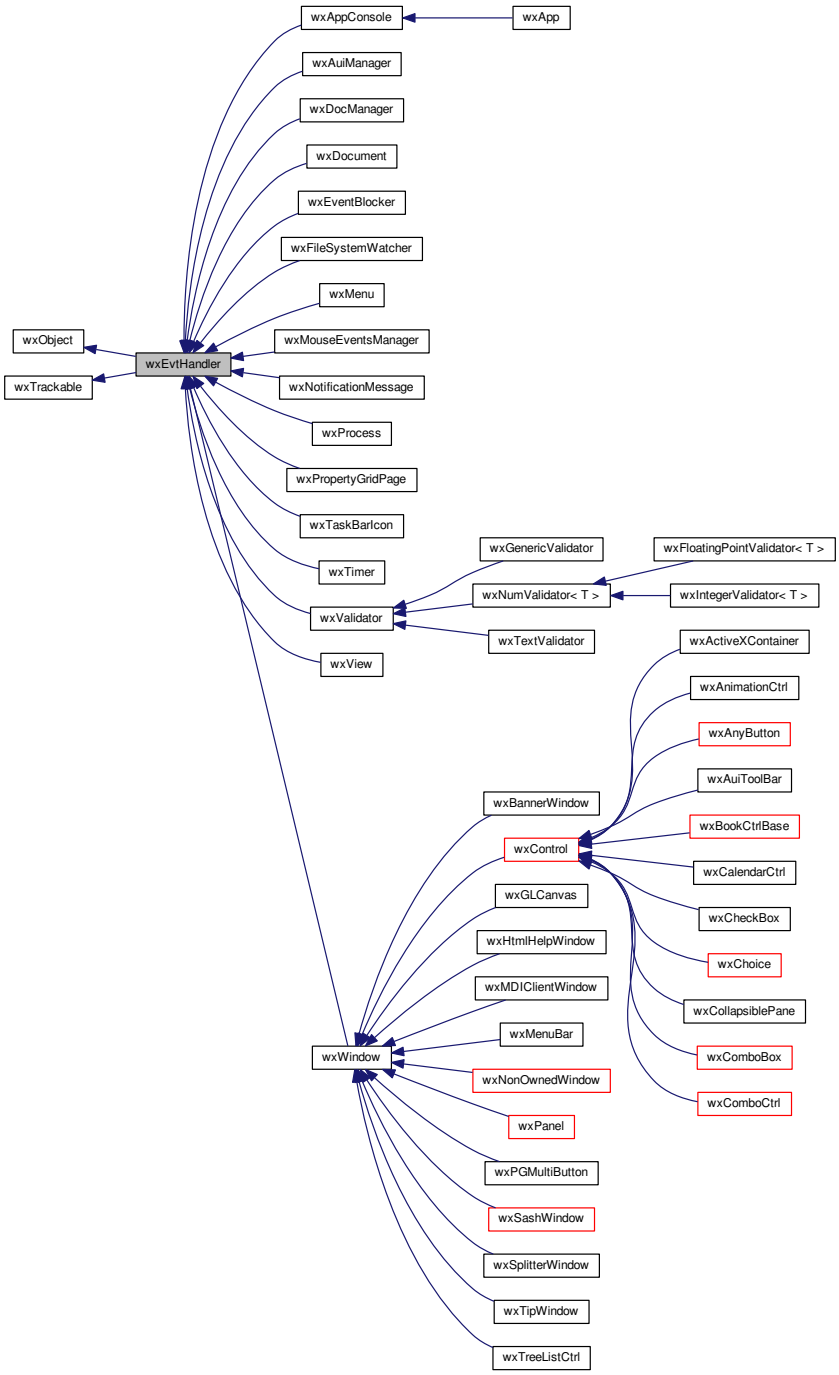
See also

wxEvent::GetEventCategory

21.222 wxEvtHandler Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxEvtHandler:



21.222.1 Detailed Description

A class that can handle events from the windowing system.

wxWindow is (and therefore all window classes are) derived from this class.

When events are received, wxEvtHandler invokes the method listed in the event table using itself as the object. When using multiple inheritance **it is imperative that the wxEvtHandler(-derived) class is the first class inherited** such that the `this` pointer for the overall object will be identical to the `this` pointer of the wxEvtHandler

portion.

Library: wxBase

Category: Events

See also

How Events are Processed, wxEventBlocker, wxEventLoopBase

Public Member Functions

- wxEvtHandler ()

  Constructor.
- virtual ~wxEvtHandler ()

  Destructor.

Event queuing and processing

- virtual void QueueEvent (wxEvent *event)

  Queue event for a later processing.
- virtual void AddPendingEvent (const wxEvent &event)

  Post an event to be processed later.
- template<typename T , typename T1 , ... >
  void CallAfter (void(T::*method)(T1,...), T1 x1,...)

  Asynchronously call the given method.
- template<typename T >
  void CallAfter (const T &functor)

  Asynchronously call the given functor.
- virtual bool ProcessEvent (wxEvent &event)

  Processes an event, searching event tables and calling zero or more suitable event handler function(s).
- bool ProcessEventLocally (wxEvent &event)

  Try to process the event in this handler and all those chained to it.
- bool SafelyProcessEvent (wxEvent &event)

  Processes an event by calling ProcessEvent() and handles any exceptions that occur in the process.
- void ProcessPendingEvents ()

  Processes the pending events previously queued using QueueEvent() or AddPendingEvent(); you must call this function only if you are sure there are pending events for this handler, otherwise a `wxCHECK` will fail.
- void DeletePendingEvents ()

  Deletes all events queued on this event handler using QueueEvent() or AddPendingEvent().
- virtual bool SearchEventTable (wxEventTable &table, wxEvent &event)

  Searches the event table, executing an event handler function if an appropriate one is found.

Connecting and disconnecting

- void Connect (int id, int lastId, wxEventType eventType, wxObjectEventFunction function, wxObject *userData=NULL, wxEvtHandler *eventSink=NULL)

  Connects the given function dynamically with the event handler, id and event type.
- void Connect (int id, wxEventType eventType, wxObjectEventFunction function, wxObject *userData=NULL, wxEvtHandler *eventSink=NULL)

  See the Connect(int, int, wxEventType, wxObjectEventFunction, wxObject, wxEvtHandler*) overload for more info.*
- void Connect (wxEventType eventType, wxObjectEventFunction function, wxObject *userData=NULL, wxEvtHandler *eventSink=NULL)

  See the Connect(int, int, wxEventType, wxObjectEventFunction, wxObject, wxEvtHandler*) overload for more info.*

- bool Disconnect (wxEventType eventType, wxObjectEventFunction function, wxObject *userData=NULL, wxEvtHandler *eventSink=NULL)

  Disconnects the given function dynamically from the event handler, using the specified parameters as search criteria and returning true if a matching function has been found and removed.
- bool Disconnect (int id=wxID_ANY, wxEventType eventType=wxEVT_NULL, wxObjectEventFunction function=NULL, wxObject *userData=NULL, wxEvtHandler *eventSink=NULL)

  See the Disconnect(wxEventType, wxObjectEventFunction, wxObject, wxEvtHandler*) overload for more info.*
- bool Disconnect (int id, int lastId, wxEventType eventType, wxObjectEventFunction function=NULL, wxObject *userData=NULL, wxEvtHandler *eventSink=NULL)

  See the Disconnect(wxEventType, wxObjectEventFunction, wxObject, wxEvtHandler*) overload for more info.*

Binding and Unbinding

- template<typename EventTag , typename Functor >
  void Bind (const EventTag &eventType, Functor functor, int id=wxID_ANY, int lastId=wxID_ANY, wxObject *userData=NULL)

  Binds the given function, functor or method dynamically with the event.
- template<typename EventTag , typename Class , typename EventArg , typename EventHandler >
  void Bind (const EventTag &eventType, void(Class::*method)(EventArg &), EventHandler *handler, int id=wxID_ANY, int lastId=wxID_ANY, wxObject *userData=NULL)

  See the Bind<>(const EventTag&, Functor, int, int, wxObject) overload for more info.*
- template<typename EventTag , typename Functor >
  bool Unbind (const EventTag &eventType, Functor functor, int id=wxID_ANY, int lastId=wxID_ANY, wxObject *userData=NULL)

  Unbinds the given function, functor or method dynamically from the event handler, using the specified parameters as search criteria and returning true if a matching function has been found and removed.
- template<typename EventTag , typename Class , typename EventArg , typename EventHandler >
  bool Unbind (const EventTag &eventType, void(Class::*method)(EventArg &), EventHandler *handler, int id=wxID_ANY, int lastId=wxID_ANY, wxObject *userData=NULL)

  See the Unbind<>(const EventTag&, Functor, int, int, wxObject) overload for more info.*

User-supplied data

- void * GetClientData () const

  Returns user-supplied client data.
- wxClientData * GetClientObject () const

  Returns a pointer to the user-supplied client data object.
- void SetClientData (void *data)

  Sets user-supplied client data.
- void SetClientObject (wxClientData *data)

  Set the client data object.

Event handler chaining

wxEvtHandler can be arranged in a double-linked list of handlers which is automatically iterated by ProcessEvent() if needed.

- bool GetEvtHandlerEnabled () const

  Returns true if the event handler is enabled, false otherwise.
- wxEvtHandler * GetNextHandler () const

  Returns the pointer to the next handler in the chain.
- wxEvtHandler * GetPreviousHandler () const

  Returns the pointer to the previous handler in the chain.
- void SetEvtHandlerEnabled (bool enabled)

  Enables or disables the event handler.
- virtual void SetNextHandler (wxEvtHandler *handler)

  Sets the pointer to the next handler.
- virtual void SetPreviousHandler (wxEvtHandler *handler)

  Sets the pointer to the previous handler.

- void Unlink ()

  Unlinks this event handler from the chain it's part of (if any); then links the "previous" event handler to the "next" one (so that the chain won't be interrupted).
- bool IsUnlinked () const

  Returns true if the next and the previous handler pointers of this event handler instance are NULL.

Static Public Member Functions

Global event filters.

Methods for working with the global list of event filters.

Event filters can be defined to pre-process all the events that happen in an application, see wxEventFilter documentation for more information.

- static void AddFilter (wxEventFilter *filter)

  Add an event filter whose FilterEvent() method will be called for each and every event processed by wxWidgets.
- static void RemoveFilter (wxEventFilter *filter)

  Remove a filter previously installed with AddFilter().

Protected Member Functions

- virtual bool TryBefore (wxEvent &event)

  Method called by ProcessEvent() before examining this object event tables.
- virtual bool TryAfter (wxEvent &event)

  Method called by ProcessEvent() as last resort.

Additional Inherited Members

21.222.2 Constructor & Destructor Documentation

wxEvtHandler::wxEvtHandler ( )

Constructor.

virtual wxEvtHandler::~wxEvtHandler ( ) [virtual]

Destructor.

If the handler is part of a chain, the destructor will unlink itself (see Unlink()).

21.222.3 Member Function Documentation

static void wxEvtHandler::AddFilter (wxEventFilter * *filter*) [static]

Add an event filter whose FilterEvent() method will be called for each and every event processed by wxWidgets.

The filters are called in LIFO order and wxApp is registered as an event filter by default. The pointer must remain valid until it's removed with RemoveFilter() and is not deleted by wxEvtHandler.

Since

2.9.3

virtual void wxEvtHandler::AddPendingEvent (const wxEvent & *event*) `[virtual]`

Post an event to be processed later.

This function is similar to QueueEvent() but can't be used to post events from worker threads for the event objects with wxString fields (i.e. in practice most of them) because of an unsafe use of the same wxString object which happens because the wxString field in the original *event* object and its copy made internally by this function share the same string buffer internally. Use QueueEvent() to avoid this.

A copy of *event* is made by the function, so the original can be deleted as soon as function returns (it is common that the original is created on the stack). This requires that the wxEvent::Clone() method be implemented by event so that it can be duplicated and stored until it gets processed.

Parameters

event	Event to add to the pending events queue.

Reimplemented in wxWindow.

template<typename EventTag , typename Functor > void wxEvtHandler::Bind (const EventTag & *eventType*, Functor *functor*, int *id* = wxID_ANY, int *lastId* = wxID_ANY, wxObject * *userData* = NULL)

Binds the given function, functor or method dynamically with the event.

This offers basically the same functionality as Connect(), but it is more flexible as it also allows you to use ordinary functions and arbitrary functors as event handlers. It is also less restrictive then Connect() because you can use an arbitrary method as an event handler, whereas Connect() requires a wxEvtHandler derived handler.

See Dynamic Event Handling for more detailed explanation of this function and the Event Sample sample for usage examples.

Parameters

eventType	The event type to be associated with this event handler.
functor	The event handler functor. This can be an ordinary function but also an arbitrary functor like boost::function<>.
id	The first ID of the identifier range to be associated with the event handler.
lastId	The last ID of the identifier range to be associated with the event handler.
userData	Optional data to be associated with the event table entry. wxWidgets will take ownership of this pointer, i.e. it will be destroyed when the event handler is disconnected or at the program termination. This pointer can be retrieved using wxEvent::GetEventUserData() later.

See also

Caveats When Not Using C++ RTTI

Since

2.9.0

template<typename EventTag , typename Class , typename EventArg , typename EventHandler > void wxEvtHandler::Bind (const EventTag & *eventType*, void(Class::*)(EventArg &) *method*, EventHandler * *handler*, int *id* = wxID_ANY, int *lastId* = wxID_ANY, wxObject * *userData* = NULL)

See the Bind<>(const EventTag&, Functor, int, int, wxObject*) overload for more info.

This overload will bind the given method as the event handler.

Parameters

eventType	The event type to be associated with this event handler.
method	The event handler method. This can be an arbitrary method (doesn't need to be from a wxEvtHandler derived class).
handler	Object whose method should be called. It must always be specified so it can be checked at compile time whether the given method is an actual member of the given handler.
id	The first ID of the identifier range to be associated with the event handler.
lastId	The last ID of the identifier range to be associated with the event handler.
userData	Optional data to be associated with the event table entry. wxWidgets will take ownership of this pointer, i.e. it will be destroyed when the event handler is disconnected or at the program termination. This pointer can be retrieved using wxEvent::GetEventUserData() later.

See also

Caveats When Not Using C++ RTTI

Since

2.9.0

template<typename T , typename T1 , ... > void wxEvtHandler::CallAfter (void(T::*)(T1,...) *method,* T1 *x1,* ...)

Asynchronously call the given method.

Calling this function on an object schedules an asynchronous call to the method specified as CallAfter() argument at a (slightly) later time. This is useful when processing some events as certain actions typically can't be performed inside their handlers, e.g. you shouldn't show a modal dialog from a mouse click event handler as this would break the mouse capture state – but you can call a method showing this message dialog after the current event handler completes.

The method being called must be the method of the object on which CallAfter() itself is called.

Notice that it is safe to use CallAfter() from other, non-GUI, threads, but that the method will be always called in the main, GUI, thread context.

Example of use:

```
class MyFrame : public wxFrame {
   void OnClick(wxMouseEvent& event) {
       CallAfter(&MyFrame::ShowPosition, event.GetPosition());
   }

   void ShowPosition(const wxPoint& pos) {
       if ( wxMessageBox(
               wxString::Format("Perform click at (%d, %d)?",
                                pos.x, pos.y), "", wxYES_NO) == wxYES )
       {
           ... do take this click into account ...
       }
   }
};
```

Parameters

method	The method to call.
x1	The (optional) first parameter to pass to the method. Currently, 0, 1 or 2 parameters can be passed. If you need to pass more than 2 arguments, you can use the CallAfter<T>(const T& fn) overload that can call any functor.

Note

This method is not available with Visual C++ before version 8 (Visual Studio 2005) as earlier versions of the compiler don't have the required support for C++ templates to implement it.

Since

2.9.5

template<typename T > void wxEvtHandler::CallAfter (const T & *functor*)

Asynchronously call the given functor.

Calling this function on an object schedules an asynchronous call to the functor specified as CallAfter() argument at a (slightly) later time. This is useful when processing some events as certain actions typically can't be performed inside their handlers, e.g. you shouldn't show a modal dialog from a mouse click event handler as this would break the mouse capture state – but you can call a function showing this message dialog after the current event handler completes.

Notice that it is safe to use CallAfter() from other, non-GUI, threads, but that the method will be always called in the main, GUI, thread context.

This overload is particularly useful in combination with C++11 lambdas:

```
wxGetApp().CallAfter([]{
    wxBell();
});
```

Parameters

functor	The functor to call.

Note

This method is not available with Visual C++ before version 8 (Visual Studio 2005) as earlier versions of the compiler don't have the required support for C++ templates to implement it.

Since

3.0

void wxEvtHandler::Connect (int *id,* int *lastId,* wxEventType *eventType,* wxObjectEventFunction *function,* wxObject * *userData* = NULL, wxEvtHandler * *eventSink* = NULL)

Connects the given function dynamically with the event handler, id and event type.

Notice that Bind() provides a more flexible and safer way to do the same thing as Connect(), please use it in any new code – while Connect() is not formally deprecated due to its existing widespread usage, it has no advantages compared to Bind().

This is an alternative to the use of static event tables. It is more flexible as it allows to connect events generated by some object to an event handler defined in a different object of a different class (which is impossible to do directly with the event tables – the events can be only handled in another object if they are propagated upwards to it). Do make sure to specify the correct *eventSink* when connecting to an event of a different object.

See Dynamic Event Handling for more detailed explanation of this function and the Event Sample sample for usage examples.

This specific overload allows you to connect an event handler to a *range* of *source* IDs. Do not confuse *source* IDs with event *types:* source IDs identify the event generator objects (typically wxMenuItem or wxWindow objects) while the event *type* identify which type of events should be handled by the given *function* (an event generator object may generate many different types of events!).

Parameters

id	The first ID of the identifier range to be associated with the event handler function.
lastId	The last ID of the identifier range to be associated with the event handler function.
eventType	The event type to be associated with this event handler.
function	The event handler function. Note that this function should be explicitly converted to the correct type which can be done using a macro called `wxFooEventHandler` for the handler for any `wxFooEvent`.
userData	Optional data to be associated with the event table entry. wxWidgets will take ownership of this pointer, i.e. it will be destroyed when the event handler is disconnected or at the program termination. This pointer can be retrieved using wxEvent::GetEventUserData() later.
eventSink	Object whose member function should be called. It must be specified when connecting an event generated by one object to a member function of a different object. If it is omitted, `this` is used.

wxPerl Note: In wxPerl this function takes 4 arguments: *id*, *lastid*, *type*, *method*; if *method* is undef, the handler is disconnected.}

See also

Bind<>()

void wxEvtHandler::Connect (int *id*, wxEventType *eventType*, wxObjectEventFunction *function*, wxObject * *userData* = NULL, wxEvtHandler * *eventSink* = NULL)

See the Connect(int, int, wxEventType, wxObjectEventFunction, wxObject*, wxEvtHandler*) overload for more info.

This overload can be used to attach an event handler to a single source ID:

Example:

```
frame->Connect( wxID_EXIT,
                wxEVT_MENU,
                wxCommandEventHandler(MyFrame::OnQuit) );
```

wxPerl Note: Not supported by wxPerl.

void wxEvtHandler::Connect (wxEventType *eventType*, wxObjectEventFunction *function*, wxObject * *userData* = NULL, wxEvtHandler * *eventSink* = NULL)

See the Connect(int, int, wxEventType, wxObjectEventFunction, wxObject*, wxEvtHandler*) overload for more info.

This overload will connect the given event handler so that regardless of the ID of the event source, the handler will be called.

wxPerl Note: Not supported by wxPerl.

void wxEvtHandler::DeletePendingEvents ( )

Deletes all events queued on this event handler using QueueEvent() or AddPendingEvent().

Use with care because the events which are deleted are (obviously) not processed and this may have unwanted consequences (e.g. user actions events will be lost).

bool wxEvtHandler::Disconnect (wxEventType *eventType*, wxObjectEventFunction *function*, wxObject * *userData* = NULL, wxEvtHandler * *eventSink* = NULL)

Disconnects the given function dynamically from the event handler, using the specified parameters as search criteria and returning true if a matching function has been found and removed.

This method can only disconnect functions which have been added using the Connect() method. There is no way to disconnect functions connected using the (static) event tables.

Parameters

eventType	The event type associated with this event handler.
function	The event handler function.
userData	Data associated with the event table entry.
eventSink	Object whose member function should be called.

wxPerl Note: Not supported by wxPerl.

bool wxEvtHandler::Disconnect (int *id* = wxID_ANY, wxEventType *eventType* = wxEVT_NULL, wxObjectEventFunction *function* = NULL, wxObject * *userData* = NULL, wxEvtHandler * *eventSink* = NULL)

See the Disconnect(wxEventType, wxObjectEventFunction, wxObject*, wxEvtHandler*) overload for more info.

This overload takes the additional *id* parameter.

wxPerl Note: Not supported by wxPerl.

bool wxEvtHandler::Disconnect (int *id*, int *lastId*, wxEventType *eventType*, wxObjectEventFunction *function* = NULL, wxObject * *userData* = NULL, wxEvtHandler * *eventSink* = NULL)

See the Disconnect(wxEventType, wxObjectEventFunction, wxObject*, wxEvtHandler*) overload for more info.

This overload takes an additional range of source IDs.

wxPerl Note: In wxPerl this function takes 3 arguments: *id*, *lastid*, *type*.

void* wxEvtHandler::GetClientData ( ) const

Returns user-supplied client data.

Remarks

Normally, any extra data the programmer wishes to associate with the object should be made available by deriving a new class with new data members.

See also

SetClientData()

wxClientData* wxEvtHandler::GetClientObject ( ) const

Returns a pointer to the user-supplied client data object.

See also

SetClientObject(), wxClientData

bool wxEvtHandler::GetEvtHandlerEnabled ( ) const

Returns true if the event handler is enabled, false otherwise.

See also

SetEvtHandlerEnabled()

wxEvtHandler* wxEvtHandler::GetNextHandler ( ) const

Returns the pointer to the next handler in the chain.

See also

SetNextHandler(), GetPreviousHandler(), SetPreviousHandler(), wxWindow::PushEventHandler, wxWindow::PopEventHandler

wxEvtHandler* wxEvtHandler::GetPreviousHandler ( ) const

Returns the pointer to the previous handler in the chain.

See also

SetPreviousHandler(), GetNextHandler(), SetNextHandler(), wxWindow::PushEventHandler, wxWindow::PopEventHandler

bool wxEvtHandler::IsUnlinked ( ) const

Returns true if the next and the previous handler pointers of this event handler instance are NULL.

Since

2.9.0

See also

SetPreviousHandler(), SetNextHandler()

virtual bool wxEvtHandler::ProcessEvent (wxEvent & *event*) [virtual]

Processes an event, searching event tables and calling zero or more suitable event handler function(s).

Normally, your application would not call this function: it is called in the wxWidgets implementation to dispatch incoming user interface events to the framework (and application).

However, you might need to call it if implementing new functionality (such as a new control) where you define new event types, as opposed to allowing the user to override virtual functions.

Notice that you don't usually need to override ProcessEvent() to customize the event handling, overriding the specially provided TryBefore() and TryAfter() functions is usually enough. For example, wxMDIParentFrame may override TryBefore() to ensure that the menu events are processed in the active child frame before being processed in the parent frame itself.

The normal order of event table searching is as follows:

1. wxApp::FilterEvent() is called. If it returns anything but `-1` (default) the processing stops here.
2. TryBefore() is called (this is where wxValidator are taken into account for wxWindow objects). If this returns true, the function exits.
3. If the object is disabled (via a call to wxEvtHandler::SetEvtHandlerEnabled) the function skips to step (7).
4. Dynamic event table of the handlers bound using Bind<>() is searched in the most-recently-bound to the most-early-bound order. If a handler is found, it is executed and the function returns true unless the handler used wxEvent::Skip() to indicate that it didn't handle the event in which case the search continues.

5. Static events table of the handlers bound using event table macros is searched for this event handler in the order of appearance of event table macros in the source code. If this fails, the base class event table is tried, and so on until no more tables exist or an appropriate function was found. If a handler is found, the same logic as in the previous step applies.
6. The search is applied down the entire chain of event handlers (usually the chain has a length of one). This chain can be formed using wxEvtHandler::SetNextHandler(): (referring to the image, if `A->ProcessEvent` is called and it doesn't handle the event, `B->ProcessEvent` will be called and so on...). Note that in the case of wxWindow you can build a stack of event handlers (see wxWindow::PushEventHandler() for more info). If any of the handlers of the chain return true, the function exits.
7. TryAfter() is called: for the wxWindow object this may propagate the event to the window parent (recursively). If the event is still not processed, ProcessEvent() on wxTheApp object is called as the last step.

Notice that steps (2)-(6) are performed in ProcessEventLocally() which is called by this function.

Parameters

event	Event to process.

Returns

true if a suitable event handler function was found and executed, and the function did not call wxEvent::Skip.

See also

SearchEventTable()

Reimplemented in wxWindow.

bool wxEvtHandler::ProcessEventLocally (wxEvent & *event*)

Try to process the event in this handler and all those chained to it.

As explained in ProcessEvent() documentation, the event handlers may be chained in a doubly-linked list. This function tries to process the event in this handler (including performing any pre-processing done in TryBefore(), e.g. applying validators) and all those following it in the chain until the event is processed or the chain is exhausted.

This function is called from ProcessEvent() and, in turn, calls TryBefore() and TryAfter(). It is not virtual and so cannot be overridden but can, and should, be called to forward an event to another handler instead of ProcessEvent() which would result in a duplicate call to TryAfter(), e.g. resulting in all unprocessed events being sent to the application object multiple times.

Since

2.9.1

Parameters

event	Event to process.

Returns

true if this handler of one of those chained to it processed the event.

void wxEvtHandler::ProcessPendingEvents ( )

Processes the pending events previously queued using QueueEvent() or AddPendingEvent(); you must call this function only if you are sure there are pending events for this handler, otherwise a `wxCHECK` will fail.

The real processing still happens in ProcessEvent() which is called by this function.

Note that this function needs a valid application object (see wxAppConsole::GetInstance()) because wxApp holds the list of the event handlers with pending events and this function manipulates that list.

virtual void wxEvtHandler::QueueEvent (wxEvent * *event*) [virtual]

Queue event for a later processing.

This method is similar to ProcessEvent() but while the latter is synchronous, i.e. the event is processed immediately, before the function returns, this one is asynchronous and returns immediately while the event will be processed at some later time (usually during the next event loop iteration).

Another important difference is that this method takes ownership of the *event* parameter, i.e. it will delete it itself. This implies that the event should be allocated on the heap and that the pointer can't be used any more after the function returns (as it can be deleted at any moment).

QueueEvent() can be used for inter-thread communication from the worker threads to the main thread, it is safe in the sense that it uses locking internally and avoids the problem mentioned in AddPendingEvent() documentation by ensuring that the *event* object is not used by the calling thread any more. Care should still be taken to avoid that some fields of this object are used by it, notably any wxString members of the event object must not be shallow copies of another wxString object as this would result in them still using the same string buffer behind the scenes. For example:

```
void FunctionInAWorkerThread(const wxString& str)
{
    wxCommandEvent* evt = new wxCommandEvent;

    // NOT evt->SetString(str) as this would be a shallow copy
    evt->SetString(str.c_str()); // make a deep copy

    wxTheApp->QueueEvent( evt );
}
```

Note that you can use wxThreadEvent instead of wxCommandEvent to avoid this problem:

```
void FunctionInAWorkerThread(const wxString& str)
{
    wxThreadEvent evt;
    evt.SetString(str);

    // wxThreadEvent::Clone() makes sure that the internal wxString
    // member is not shared by other wxString instances:
    wxTheApp->QueueEvent( evt.Clone() );
}
```

Finally notice that this method automatically wakes up the event loop if it is currently idle by calling wxWakeUpIdle() so there is no need to do it manually when using it.

Since

2.9.0

Parameters

event	A heap-allocated event to be queued, QueueEvent() takes ownership of it. This parameter shouldn't be NULL.

Reimplemented in wxWindow.

static void wxEvtHandler::RemoveFilter (wxEventFilter * *filter*) [static]

Remove a filter previously installed with AddFilter().

It's an error to remove a filter that hadn't been previously added or was already removed.

Since

2.9.3

bool wxEvtHandler::SafelyProcessEvent (wxEvent & *event*)

Processes an event by calling ProcessEvent() and handles any exceptions that occur in the process.

If an exception is thrown in event handler, wxApp::OnExceptionInMainLoop is called.

Parameters

event	Event to process.

Returns

true if the event was processed, false if no handler was found or an exception was thrown.

See also

wxWindow::HandleWindowEvent

virtual bool wxEvtHandler::SearchEventTable (wxEventTable & *table,* wxEvent & *event*) [virtual]

Searches the event table, executing an event handler function if an appropriate one is found.

Parameters

table	Event table to be searched.
event	Event to be matched against an event table entry.

Returns

true if a suitable event handler function was found and executed, and the function did not call wxEvent::Skip.

Remarks

This function looks through the object's event table and tries to find an entry that will match the event. An entry will match if:

- The event type matches, and
- the identifier or identifier range matches, or the event table entry's identifier is zero.

If a suitable function is called but calls wxEvent::Skip, this function will fail, and searching will continue.

Todo this function in the header is listed as an "implementation only" function; are we sure we want to document it?

See also

ProcessEvent()

void wxEvtHandler::SetClientData (void * *data*)

Sets user-supplied client data.

Parameters

data	Data to be associated with the event handler.

Remarks

Normally, any extra data the programmer wishes to associate with the object should be made available by deriving a new class with new data members. You must not call this method and SetClientObject on the same class - only one of them.

See also

GetClientData()

void wxEvtHandler::SetClientObject (wxClientData * *data*)

Set the client data object.

Any previous object will be deleted.

See also

GetClientObject(), wxClientData

void wxEvtHandler::SetEvtHandlerEnabled (bool *enabled*)

Enables or disables the event handler.

Parameters

enabled	true if the event handler is to be enabled, false if it is to be disabled.

Remarks

You can use this function to avoid having to remove the event handler from the chain, for example when implementing a dialog editor and changing from edit to test mode.

See also

GetEvtHandlerEnabled()

virtual void wxEvtHandler::SetNextHandler (wxEvtHandler * *handler*) [virtual]

Sets the pointer to the next handler.

Remarks

See ProcessEvent() for more info about how the chains of event handlers are internally used. Also remember that wxEvtHandler uses double-linked lists and thus if you use this function, you should also call SetPrevious↩ Handler() on the argument passed to this function:

```
handlerA->SetNextHandler(handlerB);
handlerB->SetPreviousHandler(handlerA);
```

Parameters

handler	The event handler to be set as the next handler. Cannot be NULL.

See also

How Events are Processed

Reimplemented in wxWindow.

virtual void wxEvtHandler::SetPreviousHandler (wxEvtHandler * *handler*) [virtual]

Sets the pointer to the previous handler.

All remarks about SetNextHandler() apply to this function as well.

Parameters

handler	The event handler to be set as the previous handler. Cannot be NULL.

See also

How Events are Processed

Reimplemented in wxWindow.

virtual bool wxEvtHandler::TryAfter (wxEvent & *event*) [protected], [virtual]

Method called by ProcessEvent() as last resort.

This method can be overridden to implement post-processing for the events which were not processed anywhere else.

The base class version handles forwarding the unprocessed events to wxApp at wxEvtHandler level and propagating them upwards the window child-parent chain at wxWindow level and so should usually be called when overriding this method:

```
class MyClass : public BaseClass // inheriting from wxEvtHandler
{
...
protected:
    virtual bool TryAfter(wxEvent& event)
    {
        if ( BaseClass::TryAfter(event) )
            return true;

        return MyPostProcess(event);
    }
};
```

See also

ProcessEvent()

virtual bool wxEvtHandler::TryBefore (wxEvent & *event*) [protected], [virtual]

Method called by ProcessEvent() before examining this object event tables.

This method can be overridden to hook into the event processing logic as early as possible. You should usually call the base class version when overriding this method, even if wxEvtHandler itself does nothing here, some derived classes do use this method, e.g. wxWindow implements support for wxValidator in it.

Example:

```
class MyClass : public BaseClass // inheriting from wxEvtHandler
{
...
protected:
    virtual bool TryBefore(wxEvent& event)
    {
        if ( MyPreProcess(event) )
            return true;

        return BaseClass::TryBefore(event);
    }
};
```

See also

ProcessEvent()

template<typename EventTag , typename Functor > bool wxEvtHandler::Unbind (const EventTag & *eventType*, Functor *functor*, int *id* = wxID_ANY, int *lastId* = wxID_ANY, wxObject * *userData* = NULL)

Unbinds the given function, functor or method dynamically from the event handler, using the specified parameters as search criteria and returning true if a matching function has been found and removed.

This method can only unbind functions, functors or methods which have been added using the Bind<>() method. There is no way to unbind functions bound using the (static) event tables.

Note

Currently functors are compared by their address which, unfortunately, doesn't work correctly if the same address is reused for two different functor objects. Because of this, using Unbind() is not recommended if there are multiple functors using the same *eventType* and *id* and *lastId* as a wrong one could be unbound.

Parameters

eventType	The event type associated with this event handler.
functor	The event handler functor. This can be an ordinary function but also an arbitrary functor like boost::function<>.
id	The first ID of the identifier range associated with the event handler.
lastId	The last ID of the identifier range associated with the event handler.
userData	Data associated with the event table entry.

See also

Caveats When Not Using C++ RTTI

Since

2.9.0

template<typename EventTag , typename Class , typename EventArg , typename EventHandler > bool wxEvtHandler::Unbind (const EventTag & *eventType*, void(Class::*)(EventArg &) *method*, EventHandler * *handler*, int *id* = wxID_ANY, int *lastId* = wxID_ANY, wxObject * *userData* = NULL)

See the Unbind<>(const EventTag&, Functor, int, int, wxObject*) overload for more info.

This overload unbinds the given method from the event..

Parameters

eventType	The event type associated with this event handler.
method	The event handler method associated with this event.
handler	Object whose method was called.
id	The first ID of the identifier range associated with the event handler.
lastId	The last ID of the identifier range associated with the event handler.
userData	Data associated with the event table entry.

See also

Caveats When Not Using C++ RTTI

Since

2.9.0

void wxEvtHandler::Unlink ( )

Unlinks this event handler from the chain it's part of (if any); then links the "previous" event handler to the "next" one (so that the chain won't be interrupted).

E.g. if before calling Unlink() you have the following chain: then after calling `B->Unlink()` you'll have:

Since

2.9.0

21.223 wxExecuteEnv Struct Reference

```
#include <wx/utils.h>
```

21.223.1 Detailed Description

This structure can optionally be passed to wxExecute() to specify additional options to use for the child process.

Since

2.9.2

Include file:

```
#include <wx/utils.h>
```

Public Attributes

- wxString cwd

  The initial working directory for the new process.

- wxEnvVariableHashMap env

  The environment variable map.

21.223.2 Member Data Documentation

wxString wxExecuteEnv::cwd

The initial working directory for the new process.

If this field is empty, the current working directory of this process is used.

wxEnvVariableHashMap wxExecuteEnv::env

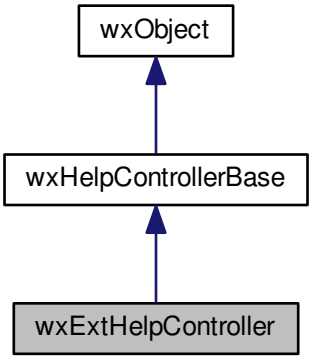
The environment variable map.

If the map is empty, the environment variables of the current process are also used for the child one, otherwise only the variables defined in this map are used.

21.224 wxExtHelpController Class Reference

`#include <wx/generic/helpext.h>`

Inheritance diagram for wxExtHelpController:



21.224.1 Detailed Description

This class implements help via an external browser.

It requires the name of a directory containing the documentation and a file mapping numerical Section numbers to relative URLS.

The map file contains two or three fields per line: numeric_id relative_URL [; comment/documentation]

The numeric_id is the id used to look up the entry in DisplaySection()/DisplayBlock(). The relative_URL is a filename of an html file, relative to the help directory. The optional comment/documentation field (after a ';') is used for keyword searches, so some meaningful text here does not hurt. If the documentation itself contains a ';', only the part before that will be displayed in the listbox, but all of it used for search.

Lines starting with ';' will be ignored.

Library: wxAdvanced

Category: Help

See also

wxHelpController

Public Member Functions

- wxExtHelpController (wxWindow *parentWindow=NULL)
- virtual ~wxExtHelpController ()
- virtual void SetViewer (const wxString &viewer=wxEmptyString, long flags=wxHELP_NETSCAPE)

  Tell it which browser to use.
- virtual bool Initialize (const wxString &dir)

  This must be called to tell the controller where to find the documentation.
- virtual bool LoadFile (const wxString &file=wxEmptyString)

  If file is "", reloads file given in Initialize.
- virtual bool DisplayContents ()

  Display list of all help entries.
- virtual bool DisplaySection (int sectionNo)

  Display help for id sectionNo.
- virtual bool DisplaySection (const wxString &section)

  Display help for id sectionNo – identical with DisplaySection().
- virtual bool DisplayBlock (long blockNo)

  Display help for URL (using DisplayHelp) or keyword (using KeywordSearch)
- virtual bool KeywordSearch (const wxString &k, wxHelpSearchMode mode=wxHELP_SEARCH_ALL)

  Search comment/documentation fields in map file and present a list to chose from.
- virtual bool Quit ()

  Does nothing.
- virtual void OnQuit ()

  Does nothing.
- virtual bool DisplayHelp (const wxString &relativeURL)

  Call the browser using a relative URL.
- virtual void SetFrameParameters (const wxString &titleFormat, const wxSize &size, const wxPoint &pos=wxDefaultPosition, bool newFrameEachTime=false)

  Allows one to override the default settings for the help frame.
- virtual wxFrame * GetFrameParameters (wxSize *size=NULL, wxPoint *pos=NULL, bool *newFrameEachTime=NULL)

  Obtains the latest settings used by the help frame and the help frame.

Additional Inherited Members

21.224.2 Constructor & Destructor Documentation

wxExtHelpController::wxExtHelpController (wxWindow * *parentWindow* = `NULL`)

virtual wxExtHelpController::~wxExtHelpController ( ) `[virtual]`

21.224.3 Member Function Documentation

virtual bool wxExtHelpController::DisplayBlock (long *blockNo*) `[virtual]`

Display help for URL (using DisplayHelp) or keyword (using KeywordSearch)

Returns

true on success

Implements wxHelpControllerBase.

virtual bool wxExtHelpController::DisplayContents ( ) `[virtual]`

Display list of all help entries.

Returns

true on success

Implements wxHelpControllerBase.

virtual bool wxExtHelpController::DisplayHelp (const wxString & *relativeURL*) `[virtual]`

Call the browser using a relative URL.

virtual bool wxExtHelpController::DisplaySection (int *sectionNo*) `[virtual]`

Display help for id sectionNo.

Returns

true on success

Implements wxHelpControllerBase.

virtual bool wxExtHelpController::DisplaySection (const wxString & *section*) `[virtual]`

Display help for id sectionNo – identical with DisplaySection().

Returns

true on success

Reimplemented from wxHelpControllerBase.

virtual wxFrame* wxExtHelpController::GetFrameParameters (wxSize * *size* = NULL, wxPoint * *pos* = NULL, bool * *newFrameEachTime* = NULL) `[virtual]`

Obtains the latest settings used by the help frame and the help frame.

Reimplemented from wxHelpControllerBase.

virtual bool wxExtHelpController::Initialize (const wxString & *dir*) `[virtual]`

This must be called to tell the controller where to find the documentation.

If a locale is set, look in file/localename, i.e. If passed "/usr/local/myapp/help" and the current wxLocale is set to be "de", then look in "/usr/local/myapp/help/de/" first and fall back to "/usr/local/myapp/help" if that doesn't exist.

Parameters

dir	directory name where to fine the help files

Returns

true on success

Reimplemented from wxHelpControllerBase.

virtual bool wxExtHelpController::KeywordSearch (const wxString & *k*, wxHelpSearchMode *mode* = wxHELP_SEARCH_ALL) [virtual]

Search comment/documentation fields in map file and present a list to chose from.

Parameters

k	string to search for, empty string will list all entries
mode	optional parameter allows the search the index (wxHELP_SEARCH_INDEX) but this currently only supported by the wxHtmlHelpController.

Returns

true on success

Implements wxHelpControllerBase.

virtual bool wxExtHelpController::LoadFile (const wxString & *file* = wxEmptyString) [virtual]

If file is "", reloads file given in Initialize.

Parameters

file	Name of help directory.

Returns

true on success

Implements wxHelpControllerBase.

virtual void wxExtHelpController::OnQuit ( ) [virtual]

Does nothing.

Reimplemented from wxHelpControllerBase.

virtual bool wxExtHelpController::Quit ( ) [virtual]

Does nothing.

Implements wxHelpControllerBase.

virtual void wxExtHelpController::SetFrameParameters (const wxString & *titleFormat*, const wxSize & *size*, const wxPoint & *pos* = wxDefaultPosition, bool *newFrameEachTime* = false) [virtual]

Allows one to override the default settings for the help frame.

Reimplemented from wxHelpControllerBase.

virtual void wxExtHelpController::SetViewer (const wxString & *viewer* = wxEmptyString, long *flags* = wxHELP_NETSCAPE) [virtual]

Tell it which browser to use.

The Netscape support will check whether Netscape is already running (by looking at the .netscape/lock file in the user's home directory) and tell it to load the page into the existing window.

Parameters

viewer	The command to call a browser/html viewer.
flags	Set this to wxHELP_NETSCAPE if the browser is some variant of Netscape.

Reimplemented from wxHelpControllerBase.

21.225 wxFFile Class Reference

`#include <wx/ffile.h>`

21.225.1 Detailed Description

wxFFile implements buffered file I/O.

This is a very small class designed to minimize the overhead of using it - in fact, there is hardly any overhead at all, but using it brings you automatic error checking and hides differences between platforms and compilers.

It wraps inside it a `FILE *` handle used by standard C IO library (also known as `stdio`).

Library: wxBase

Category: File Handling

See also

wxFFile::IsOpened

Public Member Functions

- wxFFile ()
- wxFFile (FILE *fp)

  Opens a file with the given file pointer, which has already been opened.
- wxFFile (const wxString &filename, const wxString &mode="r")

  Opens a file with the given mode.
- ~wxFFile ()

  Destructor will close the file.
- void Attach (FILE *fp, const wxString &name=wxEmptyString)

  Attaches an existing file pointer to the wxFFile object.
- bool Close ()

  Closes the file and returns true on success.
- FILE * Detach ()

  Get back a file pointer from wxFFile object – the caller is responsible for closing the file if this descriptor is opened.
- bool Eof () const

  Returns true if an attempt has been made to read past the end of the file.

- bool Error () const

  Returns true if an error has occurred on this file, similar to the standard `ferror()` function.
- bool Flush ()

  Flushes the file and returns true on success.
- wxFileKind GetKind () const

  Returns the type of the file.
- const wxString & GetName () const

  Returns the file name.
- bool IsOpened () const

  Returns true if the file is opened.
- wxFileOffset Length () const

  Returns the length of the file.
- bool Open (const wxString &filename, const wxString &mode="r")

  Opens the file, returning true if successful.
- size_t Read (void ∗buffer, size_t count)

  Reads the specified number of bytes into a buffer, returning the actual number read.
- bool ReadAll (wxString ∗str, const wxMBConv &conv=wxConvAuto())

  Reads the entire contents of the file into a string.
- bool Seek (wxFileOffset ofs, wxSeekMode mode=wxFromStart)

  Seeks to the specified position and returns true on success.
- bool SeekEnd (wxFileOffset ofs=0)

  Moves the file pointer to the specified number of bytes before the end of the file and returns true on success.
- wxFileOffset Tell () const

  Returns the current position.
- bool Write (const wxString &str, const wxMBConv &conv=wxConvAuto())

  Writes the contents of the string to the file, returns true on success.
- size_t Write (const void ∗buffer, size_t count)

  Writes the specified number of bytes from a buffer.
- FILE ∗ fp () const

  Returns the file pointer associated with the file.

21.225.2 Constructor & Destructor Documentation

wxFFile::wxFFile ( )

wxFFile::wxFFile (FILE ∗ *fp*)

Opens a file with the given file pointer, which has already been opened.

Parameters

fp	An existing file descriptor, such as stderr.

wxFFile::wxFFile (const wxString & *filename*, const wxString & *mode* = "r")

Opens a file with the given mode.

As there is no way to return whether the operation was successful or not from the constructor you should test the return value of IsOpened() to check that it didn't fail.

Parameters

filename	The filename.
mode	The mode in which to open the file using standard C strings. Note that you should use "b" flag if you use binary files under Windows or the results might be unexpected due to automatic newline conversion done for the text files.

wxFFile::~wxFFile ( )

Destructor will close the file.

Note

it is not virtual so you should *not* derive from wxFFile!

21.225.3 Member Function Documentation

void wxFFile::Attach (FILE * *fp*, const wxString & *name* = wxEmptyString)

Attaches an existing file pointer to the wxFFile object.

The descriptor should be already opened and it will be closed by wxFFile object.

bool wxFFile::Close ( )

Closes the file and returns true on success.

FILE* wxFFile::Detach ( )

Get back a file pointer from wxFFile object – the caller is responsible for closing the file if this descriptor is opened.

IsOpened() will return false after call to Detach().

Returns

The FILE pointer (this is new since wxWidgets 3.0.0, in the previous versions this method didn't return anything).

bool wxFFile::Eof ( ) const

Returns true if an attempt has been made to read *past* the end of the file.

Note that the behaviour of the file descriptor based class wxFile is different as wxFile::Eof() will return true here as soon as the last byte of the file has been read.

Also note that this method may only be called for opened files and may crash if the file is not opened.

Todo THIS METHOD MAY CRASH? DOESN'T SOUND GOOD

See also

IsOpened()

bool wxFFile::Error ( ) const

Returns true if an error has occurred on this file, similar to the standard `ferror()` function.

Please note that this method may only be called for opened files and may crash if the file is not opened.

Todo THIS METHOD MAY CRASH? DOESN'T SOUND GOOD

See also

IsOpened()

bool wxFFile::Flush ( )

Flushes the file and returns true on success.

FILE* wxFFile::fp ( ) const

Returns the file pointer associated with the file.

wxFileKind wxFFile::GetKind ( ) const

Returns the type of the file.

See also

wxFileKind

const wxString& wxFFile::GetName ( ) const

Returns the file name.

This is the name that was specified when the object was constructed or to the last call to Open(). Notice that it may be empty if Attach() was called without specifying the name.

bool wxFFile::IsOpened ( ) const

Returns true if the file is opened.

Most of the methods of this class may only be used for an opened file.

wxFileOffset wxFFile::Length ( ) const

Returns the length of the file.

bool wxFFile::Open (const wxString & *filename*, const wxString & *mode* = "r")

Opens the file, returning true if successful.

Parameters

filename	The filename.
mode	The mode in which to open the file.

size_t wxFFile::Read (void * *buffer,* size_t *count*)

Reads the specified number of bytes into a buffer, returning the actual number read.

Parameters

buffer	A buffer to receive the data.
count	The number of bytes to read.

Returns

The number of bytes read.

bool wxFFile::ReadAll (wxString * *str,* const wxMBConv & *conv* = wxConvAuto ())

Reads the entire contents of the file into a string.

Parameters

str	String to read data into.
conv	Conversion object to use in Unicode build; by default supposes that file contents is encoded in UTF-8.

Returns

true if file was read successfully, false otherwise.

bool wxFFile::Seek (wxFileOffset *ofs,* wxSeekMode *mode* = wxFromStart)

Seeks to the specified position and returns true on success.

Parameters

ofs	Offset to seek to.
mode	One of wxFromStart, wxFromEnd, wxFromCurrent.

bool wxFFile::SeekEnd (wxFileOffset *ofs* = 0)

Moves the file pointer to the specified number of bytes before the end of the file and returns true on success.

Parameters

ofs	Number of bytes before the end of the file.

wxFileOffset wxFFile::Tell ( ) const

Returns the current position.

bool wxFFile::Write (const wxString & *str,* const wxMBConv & *conv* = wxConvAuto())

Writes the contents of the string to the file, returns true on success.

The second argument is only meaningful in Unicode build of wxWidgets when *conv* is used to convert *str* to multibyte representation.

size_t wxFFile::Write (const void * *buffer,* size_t *count*)

Writes the specified number of bytes from a buffer.

Parameters

buffer	A buffer containing the data.
count	The number of bytes to write.

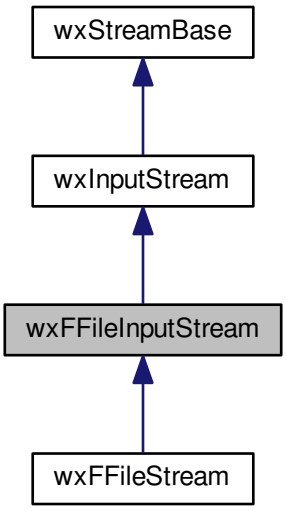
Returns

The number of bytes written.

21.226 wxFFileInputStream Class Reference

`#include <wx/wfstream.h>`

Inheritance diagram for wxFFileInputStream:



21.226.1 Detailed Description

This class represents data read in from a file.

There are actually two such groups of classes: this one is based on wxFFile whereas wxFileInputStream is based in the wxFile class.

Note that wxInputStream::SeekI() can seek beyond the end of the stream (file) and will thus not return wxInvalidOffset for that.

Library: wxBase

Category: Streams

See also

wxBufferedInputStream, wxFFileOutputStream, wxFileOutputStream

Public Member Functions

- wxFFileInputStream (const wxString &filename, const wxString &mode="rb")

  Opens the specified file using its filename name using the specified mode.
- wxFFileInputStream (wxFFile &file)

  Initializes a file stream in read-only mode using the file I/O object file.
- wxFFileInputStream (FILE *fp)

  Initializes a file stream in read-only mode using the specified file pointer fp.
- virtual ~wxFFileInputStream ()

  Destructor.
- bool IsOk () const

  Returns true if the stream is initialized and ready.
- wxFFile * GetFile () const

  Returns the underlying file object.

Additional Inherited Members

21.226.2 Constructor & Destructor Documentation

wxFFileInputStream::wxFFileInputStream (const wxString & *filename*, const wxString & *mode* = "rb")

Opens the specified file using its *filename* name using the specified *mode*.

Warning

You should use wxStreamBase::IsOk() to verify if the constructor succeeded.

wxFFileInputStream::wxFFileInputStream (wxFFile & *file*)

Initializes a file stream in read-only mode using the file I/O object file.

wxFFileInputStream::wxFFileInputStream (FILE * *fp*)

Initializes a file stream in read-only mode using the specified file pointer *fp*.

virtual wxFFileInputStream::~wxFFileInputStream ( ) [virtual]

Destructor.

21.226.3 Member Function Documentation

wxFFile* wxFFileInputStream::GetFile ( ) const

Returns the underlying file object.

Since

2.9.5

bool wxFFileInputStream::IsOk ( ) const `[virtual]`

Returns true if the stream is initialized and ready.

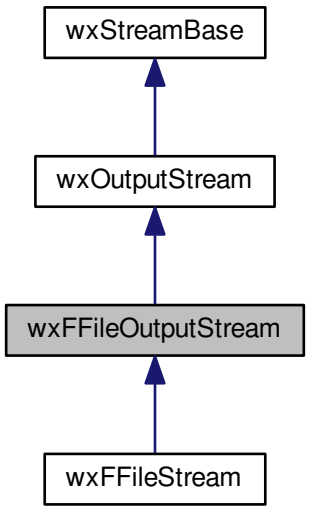
Reimplemented from wxStreamBase.

Reimplemented in wxFFileStream.

21.227 wxFFileOutputStream Class Reference

`#include <wx/wfstream.h>`

Inheritance diagram for wxFFileOutputStream:



21.227.1 Detailed Description

This class represents data written to a file.

There are actually two such groups of classes: this one is based on wxFFile whereas wxFileOutputStream is based in the wxFile class.

Note that wxOutputStream::SeekO() can seek beyond the end of the stream (file) and will thus not return wxInvalid↩ Offset for that.

Library: wxBase

Category: Streams

See also

wxBufferedOutputStream, wxFFileInputStream, wxFileOutputStream, wxFileInputStream

Public Member Functions

- wxFFileOutputStream (const wxString &filename, const wxString &mode="wb")

  Open the given file filename with mode mode.
- wxFFileOutputStream (wxFFile &file)

  Initializes a file stream in write-only mode using the file I/O object file.
- wxFFileOutputStream (FILE *fp)

  Initializes a file stream in write-only mode using the file descriptor fp.
- virtual ~wxFFileOutputStream ()

  Destructor.
- bool IsOk () const

  Returns true if the stream is initialized and ready.
- wxFFile * GetFile () const

  Returns the underlying file object.

Additional Inherited Members

21.227.2 Constructor & Destructor Documentation

wxFFileOutputStream::wxFFileOutputStream (const wxString & *filename,* const wxString & *mode* = "wb")

Open the given file *filename* with mode *mode*.

Warning

You should use wxStreamBase::IsOk() to verify if the constructor succeeded.

wxFFileOutputStream::wxFFileOutputStream (wxFFile & *file*)

Initializes a file stream in write-only mode using the file I/O object file.

wxFFileOutputStream::wxFFileOutputStream (FILE * *fp*)

Initializes a file stream in write-only mode using the file descriptor fp.

virtual wxFFileOutputStream::~wxFFileOutputStream ( ) [virtual]

Destructor.

21.227.3 Member Function Documentation

wxFFile* wxFFileOutputStream::GetFile ( ) const

Returns the underlying file object.

Since

2.9.5

bool wxFFileOutputStream::IsOk ( ) const `[virtual]`

Returns true if the stream is initialized and ready.

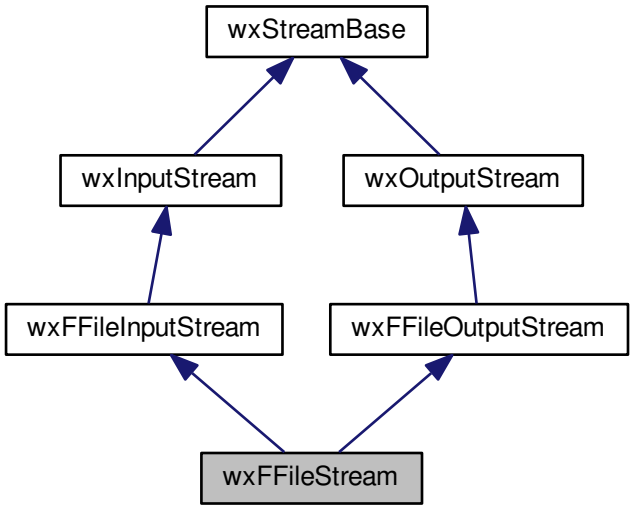
Reimplemented from wxStreamBase.

Reimplemented in wxFFileStream.

21.228 wxFFileStream Class Reference

`#include <wx/wfstream.h>`

Inheritance diagram for wxFFileStream:



21.228.1 Detailed Description

This stream allows to both read from and write to a file using buffered STDIO functions.

Library: wxBase

Category: Streams

See also

wxFFileInputStream, wxFFileOutputStream, wxFileStream

Public Member Functions

- wxFFileStream (const wxString &iofileName, const wxString &mode="w+b")

  Initializes a new file stream in the given mode using the specified iofileName name.
- bool IsOk () const

  Returns true if the stream is initialized and ready.

Additional Inherited Members

21.228.2 Constructor & Destructor Documentation

wxFFileStream::wxFFileStream (const wxString & *iofileName,* const wxString & *mode* = "w+b")

Initializes a new file stream in the given *mode* using the specified *iofileName* name.

Warning

You should use wxStreamBase::IsOk() to verify if the constructor succeeded.

21.228.3 Member Function Documentation

bool wxFFileStream::IsOk ( ) const [virtual]

Returns true if the stream is initialized and ready.

This method returns true if the stream can be both read from and written to.

Reimplemented from wxFFileOutputStream.

21.229 wxFile Class Reference

`#include <wx/file.h>`

21.229.1 Detailed Description

A wxFile performs raw file I/O.

This is a very small class designed to minimize the overhead of using it - in fact, there is hardly any overhead at all, but using it brings you automatic error checking and hides differences between platforms and compilers. wxFile also automatically closes the file in its destructor so you won't forget to do so. wxFile is a wrapper around `file` descriptor. - see also wxFFile for a wrapper around `FILE` structure.

wxFileOffset is used by the wxFile functions which require offsets as parameter or return them. If the platform supports it, wxFileOffset is a typedef for a native 64 bit integer, otherwise a 32 bit integer is used for wxFileOffset.

Library: wxBase

Category: File Handling

Public Types

- enum OpenMode {
  read,
  write,
  read_write,
  write_append,
  write_excl }

  The OpenMode enumeration defines the different modes for opening a file with wxFile.
- enum {
  fd_invalid = -1,
  fd_stdin,
  fd_stdout,
  fd_stderr }

  Standard file descriptors.

Public Member Functions

- wxFile ()

  Default constructor.
- wxFile (const wxString &filename, wxFile::OpenMode mode=wxFile::read)

  Opens a file with a filename.
- wxFile (int fd)

  Associates the file with the given file descriptor, which has already been opened.
- ~wxFile ()

  Destructor will close the file.
- int GetLastError () const

  Returns the error code for the last unsuccessful operation.
- void ClearLastError ()

  Resets the error code.
- void Attach (int fd)

  Attaches an existing file descriptor to the wxFile object.
- bool Close ()

  Closes the file.
- bool Create (const wxString &filename, bool overwrite=false, int access=wxS_DEFAULT)

  Creates a file for writing.
- int Detach ()

  Get back a file descriptor from wxFile object - the caller is responsible for closing the file if this descriptor is opened.
- bool Eof () const

  Returns true if the end of the file has been reached.
- bool Flush ()

  Flushes the file descriptor.
- wxFileKind GetKind () const

  Returns the type of the file.
- bool IsOpened () const

  Returns true if the file has been opened.
- wxFileOffset Length () const

  Returns the length of the file.
- bool Open (const wxString &filename, wxFile::OpenMode mode=wxFile::read, int access=wxS_DEFAULT)

  Opens the file, returning true if successful.
- ssize_t Read (void *buffer, size_t count)

  Reads from the file into a memory buffer.

- bool ReadAll (wxString *str, const wxMBConv &conv=wxConvAuto())

  Reads the entire contents of the file into a string.
- wxFileOffset Seek (wxFileOffset ofs, wxSeekMode mode=wxFromStart)

  Seeks to the specified position.
- wxFileOffset SeekEnd (wxFileOffset ofs=0)

  Moves the file pointer to the specified number of bytes relative to the end of the file.
- wxFileOffset Tell () const

  Returns the current position or wxInvalidOffset if file is not opened or if another error occurred.
- size_t Write (const void *buffer, size_t count)

  Write data to the file (descriptor).
- bool Write (const wxString &s, const wxMBConv &conv=wxConvUTF8)

  Writes the contents of the string to the file, returns true on success.
- int fd () const

  Returns the file descriptor associated with the file.

Static Public Member Functions

- static bool Access (const wxString &name, wxFile::OpenMode mode)

  This function verifies if we may access the given file in specified mode.
- static bool Exists (const wxString &filename)

  Returns true if the given name specifies an existing regular file (not a directory or a link).

21.229.2 Member Enumeration Documentation

anonymous enum

Standard file descriptors.

Enumerator

fd_invalid

fd_stdin

fd_stdout

fd_stderr

enum wxFile::OpenMode

The OpenMode enumeration defines the different modes for opening a file with wxFile.

It is also used with wxFile::Access function.

Enumerator

read Open file for reading or test if it can be opened for reading with Access()

write Open file for writing deleting the contents of the file if it already exists or test if it can be opened for writing with Access().

read_write Open file for reading and writing; cannot be used with Access()

write_append Open file for appending: the file is opened for writing, but the old contents of the file are not erased and the file pointer is initially placed at the end of the file; cannot be used with Access(). This is the same as OpenMode::write if the file doesn't exist.

write_excl Open the file securely for writing (Uses O_EXCL | O_CREAT). Will fail if the file already exists, else create and open it atomically. Useful for opening temporary files without being vulnerable to race exploits.

21.229.3 Constructor & Destructor Documentation

wxFile::wxFile ( )

Default constructor.

wxFile::wxFile (const wxString & *filename,* wxFile::OpenMode *mode* = wxFile::read)

Opens a file with a filename.

Parameters

filename	The filename.
mode	The mode in which to open the file.

Warning

You should use IsOpened() to verify that the constructor succeeded.

wxFile::wxFile (int *fd*)

Associates the file with the given file descriptor, which has already been opened.

See Attach() for the list of predefined descriptors.

Parameters

fd	An existing file descriptor.

wxFile::~wxFile ( )

Destructor will close the file.

Note

This destructor is not virtual so you should not use wxFile polymorphically.

21.229.4 Member Function Documentation

static bool wxFile::Access (const wxString & *name,* wxFile::OpenMode *mode*) [static]

This function verifies if we may access the given file in specified mode.

Only values of `wxFile::read` or `wxFile::write` really make sense here.

void wxFile::Attach (int *fd*)

Attaches an existing file descriptor to the wxFile object.

Examples of predefined file descriptors are 0, 1 and 2 which correspond to stdin, stdout and stderr (and have symbolic names of `wxFile::fd_stdin`, `wxFile::fd_stdout` and `wxFile::fd_stderr`).

The descriptor should be already opened and it will be closed by wxFile object.

void wxFile::ClearLastError ( )

Resets the error code.

GetLastError() will return 0 until the next error occurs.

Since

2.9.2

bool wxFile::Close ( )

Closes the file.

bool wxFile::Create (const wxString & *filename*, bool *overwrite* = `false`, int *access* = wxS_DEFAULT)

Creates a file for writing.

If the file already exists, setting **overwrite** to true will ensure it is overwritten.

access may be an OR combination of the wxPosixPermissions enumeration values.

int wxFile::Detach ( )

Get back a file descriptor from wxFile object - the caller is responsible for closing the file if this descriptor is opened.

IsOpened() will return false after call to Detach().

Returns

The file descriptor (this is new since wxWidgets 3.0.0, in the previous versions this method didn't return anything).

bool wxFile::Eof ( ) const

Returns true if the end of the file has been reached.

Note that the behaviour of the file pointer-based class wxFFile is different as wxFFile::Eof() will return true here only if an attempt has been made to read **past** the last byte of the file, while wxFile::Eof() will return true even before such attempt is made if the file pointer is at the last position in the file.

Note also that this function doesn't work on unseekable file descriptors (examples include pipes, terminals and sockets under Unix) and an attempt to use it will result in an error message.

So, to read the entire file into memory, you should write a loop which uses Read() repeatedly and tests its return condition instead of using Eof() as this will not work for special files under Unix.

static bool wxFile::Exists (const wxString & *filename*) `[static]`

Returns true if the given name specifies an existing regular file (not a directory or a link).

int wxFile::fd ( ) const

Returns the file descriptor associated with the file.

bool wxFile::Flush ( )

Flushes the file descriptor.

Note that Flush() is not implemented on some Windows compilers due to a missing fsync function, which reduces the usefulness of this function (it can still be called but it will do nothing on unsupported compilers).

wxFileKind wxFile::GetKind ( ) const

Returns the type of the file.

int wxFile::GetLastError ( ) const

Returns the error code for the last unsuccessful operation.

The error code is system-dependent and corresponds to the value of the standard `errno` variable when the last error occurred.

Notice that only simple accessors such as IsOpened() and Eof() (and this method itself) don't modify the last error value, all other methods can potentially change it if an error occurs, including the const ones such as Tell() or Length().

Since

2.9.2

See also

ClearLastError()

bool wxFile::IsOpened ( ) const

Returns true if the file has been opened.

wxFileOffset wxFile::Length ( ) const

Returns the length of the file.

bool wxFile::Open (const wxString & *filename*, wxFile::OpenMode *mode* = wxFile::read, int *access* = wxS_DEFAULT)

Opens the file, returning true if successful.

Parameters

filename	The filename.
mode	The mode in which to open the file.
access	An OR-combination of wxPosixPermissions enumeration values.

ssize_t wxFile::Read (void * *buffer*, size_t *count*)

Reads from the file into a memory buffer.

Parameters

buffer	Buffer to write in
count	Bytes to read

Returns

The number of bytes read, or the symbol wxInvalidOffset.

bool wxFile::ReadAll (wxString * *str*, const wxMBConv & *conv* = wxConvAuto())

Reads the entire contents of the file into a string.

Parameters

str	Non-NULL pointer to a string to read data into.
conv	Conversion object to use in Unicode build; by default supposes that file contents is encoded in UTF-8 but falls back to the current locale encoding (or Latin-1 if it is UTF-8 too) if it is not.

Returns

true if file was read successfully, false otherwise.

Since

2.9.5

wxFileOffset wxFile::Seek (wxFileOffset *ofs*, wxSeekMode *mode* = wxFromStart)

Seeks to the specified position.

Parameters

ofs	Offset to seek to.
mode	One of wxFromStart, wxFromEnd, wxFromCurrent.

Returns

The actual offset position achieved, or wxInvalidOffset on failure.

wxFileOffset wxFile::SeekEnd (wxFileOffset *ofs* = 0)

Moves the file pointer to the specified number of bytes relative to the end of the file.

For example, `SeekEnd`(-5) would position the pointer 5 bytes before the end.

Parameters

ofs	Number of bytes before the end of the file.

Returns

The actual offset position achieved, or wxInvalidOffset on failure.

wxFileOffset wxFile::Tell ( ) const

Returns the current position or wxInvalidOffset if file is not opened or if another error occurred.

size_t wxFile::Write (const void * *buffer*, size_t *count*)

Write data to the file (descriptor).

Parameters

buffer	Buffer from which to read data
count	Number of bytes to write

Returns

The number of bytes written.

bool wxFile::Write (const wxString & *s*, const wxMBConv & *conv* = `wxConvUTF8`)

Writes the contents of the string to the file, returns true on success.

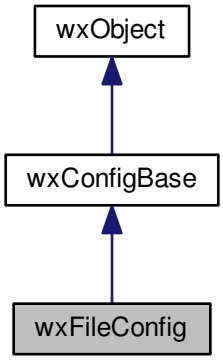
The second argument is only meaningful in Unicode build of wxWidgets when *conv* is used to convert *s* to a multibyte representation.

Note that this method only works with `NUL-terminated` strings, if you want to write data with embedded `NULs` to the file you should use the other Write() overload.

21.230 wxFileConfig Class Reference

`#include <wx/fileconf.h>`

Inheritance diagram for wxFileConfig:



21.230.1 Detailed Description

wxFileConfig implements wxConfigBase interface for storing and retrieving configuration information using plain text files.

The files have a simple format reminiscent of Windows INI files with lines of the form `"key = value"` defining the keys and lines of special form `"[group]"` indicating the start of each group.

This class is used by default for wxConfig on Unix platforms but may also be used explicitly if you want to use files and not the registry even under Windows.

Library: wxBase

Category: Application and System configuration

See also

wxFileConfig::Save

Public Member Functions

- wxFileConfig (const wxString &appName=wxEmptyString, const wxString &vendorName=wxEmptyString, const wxString &localFilename=wxEmptyString, const wxString &globalFilename=wxEmptyString, long style=wxCONFIG_USE_LOCAL_FILE|wxCONFIG_USE_GLOBAL_FILE, const wxMBConv &conv=wxConvAuto())

  Constructor allowing to choose the file names to use.
- wxFileConfig (wxInputStream &is, const wxMBConv &conv=wxConvAuto())

  Read the config data from the specified stream instead of the associated file, as usual.
- virtual bool Save (wxOutputStream &os, const wxMBConv &conv=wxConvAuto())

  Saves all config data to the given stream, returns true if data was saved successfully or false on error.
- void SetUmask (int mode)

  Allows to set the mode to be used for the config file creation.
- virtual void SetPath (const wxString &strPath)

  Set current path: if the first character is '/', it is the absolute path, otherwise it is a relative path.
- virtual const wxString & GetPath () const

  Retrieve the current path (always as absolute path).
- virtual bool GetFirstGroup (wxString &str, long &lIndex) const

  Gets the first group.
- virtual bool GetNextGroup (wxString &str, long &lIndex) const

  Gets the next group.
- virtual bool GetFirstEntry (wxString &str, long &lIndex) const

  Gets the first entry.
- virtual bool GetNextEntry (wxString &str, long &lIndex) const

  Gets the next entry.
- virtual size_t GetNumberOfEntries (bool bRecursive=false) const

  Get number of entries in the current group.
- virtual size_t GetNumberOfGroups (bool bRecursive=false) const

  Get number of entries/subgroups in the current group, with or without its subgroups.
- virtual bool HasGroup (const wxString &strName) const
- virtual bool HasEntry (const wxString &strName) const
- virtual bool Flush (bool bCurrentOnly=false)

  Permanently writes all changes (otherwise, they're only written from object's destructor).
- virtual bool RenameEntry (const wxString &oldName, const wxString &newName)

  Renames an entry in the current group.
- virtual bool RenameGroup (const wxString &oldName, const wxString &newName)

  Renames a subgroup of the current group.
- virtual bool DeleteEntry (const wxString &key, bool bGroupIfEmptyAlso=true)

  Deletes the specified entry and the group it belongs to if it was the last key in it and the second parameter is true.
- virtual bool DeleteGroup (const wxString &szKey)

  Delete the group (with all subgroups).
- virtual bool DeleteAll ()

  Delete the whole underlying object (disk file, registry key, ...).

Static Public Member Functions

- static wxFileName GetGlobalFile (const wxString &basename)

  Return the full path to the file which would be used by wxFileConfig as global, system-wide, file if it were constructed with basename as "global filename" parameter in the constructor.

- static wxFileName GetLocalFile (const wxString &basename, int style=0)

  Return the full path to the file which would be used by wxFileConfig as local, user-specific, file if it were constructed with basename as "local filename" parameter in the constructor.

- static wxString GetGlobalFileName (const wxString &szFile)
- static wxString GetLocalFileName (const wxString &szFile, int style=0)

Additional Inherited Members

21.230.2 Constructor & Destructor Documentation

wxFileConfig::wxFileConfig (const wxString & *appName* = wxEmptyString, const wxString & *vendorName* = wxEmptyString, const wxString & *localFilename* = wxEmptyString, const wxString & *globalFilename* = wxEmptyString, long *style* = wxCONFIG_USE_LOCAL_FILE|wxCONFIG_USE_GLOBAL_FILE, const wxMBConv & *conv* = wxConvAuto())

Constructor allowing to choose the file names to use.

If *localFilename* and/or *globalFilename* are explicitly specified, they are used as the names of the user and system-wide configuration files (the latter is only read by the program while the former is read from and written to). Otherwise the behaviour depends on *style* parameter. If it includes wxCONFIG_USE_LOCAL_FILE, then the local file name is constructed from the information in *appName* and *vendorName* arguments in a system-dependent way. If wxCONFIG_USE_GLOBAL_FILE is not specified at all (and *globalFilename* is empty) then the system-wide file is not used at all. Otherwise its name and path are also constructed in the way appropriate for the current platform from the application and vendor names.

wxFileConfig::wxFileConfig (wxInputStream & *is*, const wxMBConv & *conv* = wxConvAuto())

Read the config data from the specified stream instead of the associated file, as usual.

See also

Save()

21.230.3 Member Function Documentation

virtual bool wxFileConfig::DeleteAll ( ) [virtual]

Delete the whole underlying object (disk file, registry key, ...).

Primarily for use by uninstallation routine.

Implements wxConfigBase.

virtual bool wxFileConfig::DeleteEntry (const wxString & *key*, bool *bDeleteGroupIfEmpty* = true) [virtual]

Deletes the specified entry and the group it belongs to if it was the last key in it and the second parameter is true.

Implements wxConfigBase.

virtual bool wxFileConfig::DeleteGroup (const wxString & *key*) `[virtual]`

Delete the group (with all subgroups).

If the current path is under the group being deleted it is changed to its deepest still existing component. E.g. if the current path is `"/A/B/C/D"` and the group `C` is deleted, the path becomes `"/A/B"`.

Implements wxConfigBase.

virtual bool wxFileConfig::Flush (bool *bCurrentOnly* = `false`) `[virtual]`

Permanently writes all changes (otherwise, they're only written from object's destructor).

Implements wxConfigBase.

virtual bool wxFileConfig::GetFirstEntry (wxString & *str,* long & *index*) const `[virtual]`

Gets the first entry.

wxPerl Note: In wxPerl this method takes no parameters and returns a 3-element list (continue_flag, string, index↩ _for_getnextentry).

Implements wxConfigBase.

virtual bool wxFileConfig::GetFirstGroup (wxString & *str,* long & *index*) const `[virtual]`

Gets the first group.

wxPerl Note: In wxPerl this method takes no parameters and returns a 3-element list (continue_flag, string, index↩ _for_getnextentry).

Implements wxConfigBase.

static wxFileName wxFileConfig::GetGlobalFile (const wxString & *basename*) `[static]`

Return the full path to the file which would be used by wxFileConfig as global, system-wide, file if it were constructed with *basename* as "global filename" parameter in the constructor.

Notice that this function cannot be used if *basename* is already a full path name.

static wxString wxFileConfig::GetGlobalFileName (const wxString & *szFile*) `[static]`

static wxFileName wxFileConfig::GetLocalFile (const wxString & *basename,* int *style* = `0`) `[static]`

Return the full path to the file which would be used by wxFileConfig as local, user-specific, file if it were constructed with *basename* as "local filename" parameter in the constructor.

style has the same meaning as in wxConfig constructor and can contain any combination of styles but only wxC↩ ONFIG_USE_SUBDIR bit is examined by this function.

Notice that this function cannot be used if *basename* is already a full path name.

static wxString wxFileConfig::GetLocalFileName (const wxString & *szFile,* int *style* = `0`) `[static]`

virtual bool wxFileConfig::GetNextEntry (wxString & *str,* long & *index*) const `[virtual]`

Gets the next entry.

wxPerl Note: In wxPerl this method only takes the *index* parameter and returns a 3-element list (continue_flag, string, index_for_getnextentry).

Implements wxConfigBase.

virtual bool wxFileConfig::GetNextGroup (wxString & *str,* long & *index*) const `[virtual]`

Gets the next group.

wxPerl Note: In wxPerl this method only takes the *index* parameter and returns a 3-element list (continue_flag, string, index_for_getnextentry).

Implements wxConfigBase.

virtual size_t wxFileConfig::GetNumberOfEntries (bool *bRecursive* = `false`) const `[virtual]`

Get number of entries in the current group.

Implements wxConfigBase.

virtual size_t wxFileConfig::GetNumberOfGroups (bool *bRecursive* = `false`) const `[virtual]`

Get number of entries/subgroups in the current group, with or without its subgroups.

Implements wxConfigBase.

virtual const wxString& wxFileConfig::GetPath ( ) const `[virtual]`

Retrieve the current path (always as absolute path).

Implements wxConfigBase.

virtual bool wxFileConfig::HasEntry (const wxString & *strName*) const `[virtual]`

Returns

true if the entry by this name exists.

Implements wxConfigBase.

virtual bool wxFileConfig::HasGroup (const wxString & *strName*) const `[virtual]`

Returns

true if the group by this name exists.

Implements wxConfigBase.

virtual bool wxFileConfig::RenameEntry (const wxString & *oldName,* const wxString & *newName*) `[virtual]`

Renames an entry in the current group.

The entries names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns

false if *oldName* doesn't exist or if *newName* already exists.

Implements wxConfigBase.

virtual bool wxFileConfig::RenameGroup (const wxString & *oldName,* const wxString & *newName*) [virtual]

Renames a subgroup of the current group.

The subgroup names (both the old and the new one) shouldn't contain backslashes, i.e. only simple names and not arbitrary paths are accepted by this function.

Returns

false if *oldName* doesn't exist or if *newName* already exists.

Implements wxConfigBase.

virtual bool wxFileConfig::Save (wxOutputStream & *os,* const wxMBConv & *conv* = wxConvAuto()) [virtual]

Saves all config data to the given stream, returns true if data was saved successfully or false on error.

Note the interaction of this function with the internal "dirty flag": the data is saved unconditionally, i.e. even if the object is not dirty. However after saving it successfully, the dirty flag is reset so no changes will be written back to the file this object is associated with until you change its contents again.

See also

wxConfigBase::Flush

virtual void wxFileConfig::SetPath (const wxString & *strPath*) [virtual]

Set current path: if the first character is '/', it is the absolute path, otherwise it is a relative path.

'..' is supported. If *strPath* doesn't exist, it is created.

See also

wxConfigPathChanger

Implements wxConfigBase.

void wxFileConfig::SetUmask (int *mode*)

Allows to set the mode to be used for the config file creation.

For example, to create a config file which is not readable by other users (useful if it stores some sensitive information, such as passwords), you could use `SetUmask(0077)`.

This function doesn't do anything on non-Unix platforms.

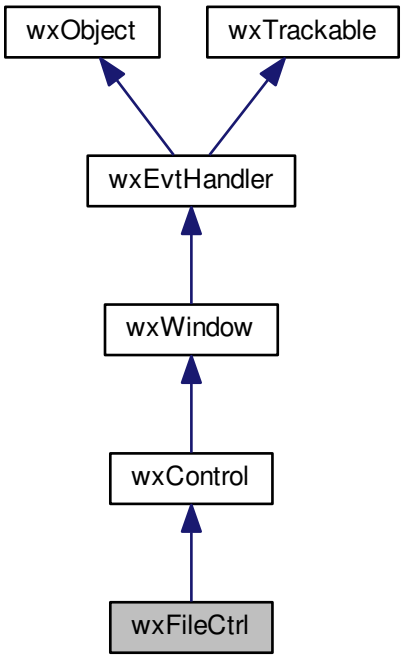
See also

wxCHANGE_UMASK()

21.231 wxFileCtrl Class Reference

`#include <wx/filectrl.h>`

Inheritance diagram for wxFileCtrl:



21.231.1 Detailed Description

This control allows the user to select a file.

Two implementations of this class exist, one for Gtk and another generic one for all the other ports.

This class is only available if `wxUSE_FILECTRL` is set to 1.

Styles

This class supports the following styles:

- wxFC_DEFAULT_STYLE: The default style: wxFC_OPEN
- wxFC_OPEN: Creates an file control suitable for opening files. Cannot be combined with wxFC_SAVE.
- wxFC_SAVE: Creates an file control suitable for saving files. Cannot be combined with wxFC_OPEN.
- wxFC_MULTIPLE: For open control only, Allows selecting multiple files. Cannot be combined with wxFC_↩ SAVE
- wxFC_NOSHOWHIDDEN: Hides the "Show Hidden Files" checkbox (Generic only)

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFileCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_FILECTRL_FILEACTIVATED(id, func): The user activated a file(by double-clicking or pressing Enter)
- EVT_FILECTRL_SELECTIONCHANGED(id, func): The user changed the current selection(by selecting or deselecting a file)
- EVT_FILECTRL_FOLDERCHANGED(id, func): The current folder of the file control has been changed
- EVT_FILECTRL_FILTERCHANGED(id, func): The current file filter of the file control has been changed.

  Since

  2.9.1.

Library: wxCore

Category: Controls

Implementations: native under wxGTK port; a generic implementation is used elsewhere.

See also

wxGenericDirCtrl

Public Member Functions

- wxFileCtrl ()
- wxFileCtrl (wxWindow *parent, wxWindowID id, const wxString &defaultDirectory=wxEmptyString, const wxString &defaultFilename=wxEmptyString, const wxString &wildCard=wxFileSelectorDefaultWildcardStr, long style=wxFC_DEFAULT_STYLE, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, const wxString &name=wxFileCtrlNameStr)

  Constructs the window.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &defaultDirectory=wxEmptyString, const wxString &defaultFilename=wxEmptyString, const wxString &wildCard=wxFileSelectorDefaultWildcardStr, long style=wxFC_DEFAULT_STYLE, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, const wxString &name=wxFileCtrlNameStr)

  Create function for two-step construction.
- virtual wxString GetDirectory () const

  Returns the current directory of the file control (i.e. the directory shown by it).
- virtual wxString GetFilename () const

  Returns the currently selected filename.
- virtual void GetFilenames (wxArrayString &filenames) const

  Fills the array filenames with the filenames only of selected items.
- virtual int GetFilterIndex () const

  Returns the zero-based index of the currently selected filter.
- virtual wxString GetPath () const

  Returns the full path (directory and filename) of the currently selected file.
- virtual void GetPaths (wxArrayString &paths) const

  Fills the array paths with the full paths of the files chosen.
- virtual wxString GetWildcard () const

Returns the current wildcard.

- virtual bool SetDirectory (const wxString &directory)

  Sets(changes) the current directory displayed in the control.
- virtual bool SetFilename (const wxString &filename)

  Selects a certain file.
- virtual bool SetPath (const wxString &path)

  Changes to a certain directory and selects a certain file.
- virtual void SetFilterIndex (int filterIndex)

  Sets the current filter index, starting from zero.
- virtual void SetWildcard (const wxString &wildCard)

  Sets the wildcard, which can contain multiple file types, for example: "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.gif".
- virtual void ShowHidden (bool show)

  Sets whether hidden files and folders are shown or not.

Additional Inherited Members

21.231.2 Constructor & Destructor Documentation

wxFileCtrl::wxFileCtrl ( )

wxFileCtrl::wxFileCtrl (wxWindow ∗ *parent*, wxWindowID *id*, const wxString & *defaultDirectory* = wxEmptyString, const wxString & *defaultFilename* = wxEmptyString, const wxString & *wildCard* = wxFileSelectorDefaultWildcardStr, long *style* = wxFC_DEFAULT_STYLE, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, const wxString & *name* = `wxFileCtrlNameStr`)

Constructs the window.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
defaultDirectory	The initial directory shown in the control. Must be a valid path to a directory or the empty string. In case it is the empty string, the current working directory is used.
defaultFilename	The default filename, or the empty string.
wildCard	A wildcard specifying which files can be selected, such as "∗.∗" or "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.gif".
style	The window style, see wxFC_∗ flags.
pos	Initial position.
size	Initial size.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

21.231.3 Member Function Documentation

bool wxFileCtrl::Create (wxWindow ∗ *parent*, wxWindowID *id*, const wxString & *defaultDirectory* = wxEmptyString, const wxString & *defaultFilename* = wxEmptyString, const wxString & *wildCard* = wxFileSelectorDefaultWildcardStr, long *style* = wxFC_DEFAULT_STYLE, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, const wxString & *name* = `wxFileCtrlNameStr`)

Create function for two-step construction.

See wxFileCtrl() for details.

virtual wxString wxFileCtrl::GetDirectory ( ) const `[virtual]`

Returns the current directory of the file control (i.e. the directory shown by it).

virtual wxString wxFileCtrl::GetFilename ( ) const `[virtual]`

Returns the currently selected filename.

For the controls having the `wxFC_MULTIPLE` style, use GetFilenames() instead.

virtual void wxFileCtrl::GetFilenames (wxArrayString & *filenames*) const `[virtual]`

Fills the array *filenames* with the filenames only of selected items.

This function should only be used with the controls having the `wxFC_MULTIPLE` style, use GetFilename() for the others.

Remarks

filenames is emptied first.

virtual int wxFileCtrl::GetFilterIndex ( ) const `[virtual]`

Returns the zero-based index of the currently selected filter.

virtual wxString wxFileCtrl::GetPath ( ) const `[virtual]`

Returns the full path (directory and filename) of the currently selected file.

For the controls having the `wxFC_MULTIPLE` style, use GetPaths() instead.

virtual void wxFileCtrl::GetPaths (wxArrayString & *paths*) const `[virtual]`

Fills the array *paths* with the full paths of the files chosen.

This function should be used with the controls having the `wxFC_MULTIPLE` style, use GetPath() otherwise.

Remarks

paths is emptied first.

virtual wxString wxFileCtrl::GetWildcard ( ) const `[virtual]`

Returns the current wildcard.

virtual bool wxFileCtrl::SetDirectory (const wxString & *directory*) `[virtual]`

Sets(changes) the current directory displayed in the control.

Returns

Returns true on success, false otherwise.

virtual bool wxFileCtrl::SetFilename (const wxString & *filename*) `[virtual]`

Selects a certain file.

Returns

Returns true on success, false otherwise

virtual void wxFileCtrl::SetFilterIndex (int *filterIndex*) `[virtual]`

Sets the current filter index, starting from zero.

virtual bool wxFileCtrl::SetPath (const wxString & *path*) `[virtual]`

Changes to a certain directory and selects a certain file.

In case the filename specified isn't found/couldn't be shown with currently selected filter, false is returned.

Returns

Returns true on success, false otherwise

virtual void wxFileCtrl::SetWildcard (const wxString & *wildCard*) `[virtual]`

Sets the wildcard, which can contain multiple file types, for example: "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.↩ gif".

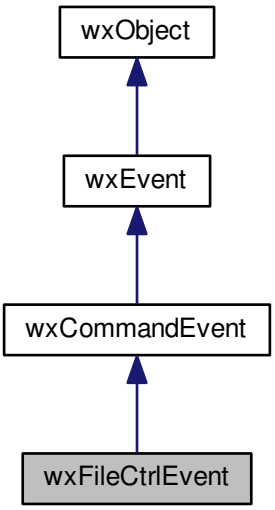
virtual void wxFileCtrl::ShowHidden (bool *show*) `[virtual]`

Sets whether hidden files and folders are shown or not.

21.232 wxFileCtrlEvent Class Reference

`#include <wx/filectrl.h>`

Inheritance diagram for wxFileCtrlEvent:



21.232.1 Detailed Description

A file control event holds information about events associated with wxFileCtrl objects.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFileCtrlEvent& event)

Event macros:

- EVT_FILECTRL_FILEACTIVATED(id, func): The user activated a file(by double-clicking or pressing Enter)
- EVT_FILECTRL_SELECTIONCHANGED(id, func): The user changed the current selection(by selecting or deselecting a file)
- EVT_FILECTRL_FOLDERCHANGED(id, func): The current folder of the file control has been changed
- EVT_FILECTRL_FILTERCHANGED(id, func): The current file filter of the file control has been changed

Library: wxCore

Category: Events

Public Member Functions

- wxFileCtrlEvent (wxEventType type, wxObject *evtObject, int id)

  Constructor.

- wxString GetDirectory () const

  Returns the current directory.
- wxString GetFile () const

  Returns the file selected (assuming it is only one file).
- wxArrayString GetFiles () const

  Returns the files selected.
- int GetFilterIndex () const

  Returns the current file filter index.
- void SetFiles (const wxArrayString &files)

  Sets the files changed by this event.
- void SetDirectory (const wxString &directory)

  Sets the directory of this event.
- void SetFilterIndex (int index)

  Sets the filter index changed by this event.

Additional Inherited Members

21.232.2 Constructor & Destructor Documentation

wxFileCtrlEvent::wxFileCtrlEvent (wxEventType *type,* wxObject * *evtObject,* int *id*)

Constructor.

21.232.3 Member Function Documentation

wxString wxFileCtrlEvent::GetDirectory ( ) const

Returns the current directory.

In case of a **EVT_FILECTRL_FOLDERCHANGED**, this method returns the new directory.

wxString wxFileCtrlEvent::GetFile ( ) const

Returns the file selected (assuming it is only one file).

wxArrayString wxFileCtrlEvent::GetFiles ( ) const

Returns the files selected.

In case of a **EVT_FILECTRL_SELECTIONCHANGED**, this method returns the files selected after the event.

int wxFileCtrlEvent::GetFilterIndex ( ) const

Returns the current file filter index.

For a **EVT_FILECTRL_FILTERCHANGED** event, this method returns the new file filter index.

Since

2.9.1

void wxFileCtrlEvent::SetDirectory (const wxString & *directory*)

Sets the directory of this event.

void wxFileCtrlEvent::SetFiles (const wxArrayString & *files*)

Sets the files changed by this event.

void wxFileCtrlEvent::SetFilterIndex (int *index*)

Sets the filter index changed by this event.

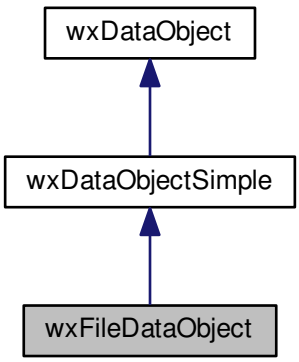
Since

2.9.1

21.233 wxFileDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxFileDataObject:



21.233.1 Detailed Description

wxFileDataObject is a specialization of wxDataObject for file names.

The program works with it just as if it were a list of absolute file names, but internally it uses the same format as Explorer and other compatible programs under Windows or GNOME/KDE filemanager under Unix which makes it possible to receive files from them using this class.

Warning

Under all non-Windows platforms this class is currently "input-only", i.e. you can receive the files from another application, but copying (or dragging) file(s) from a wxWidgets application is not currently supported. PS: GTK2 should work as well.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

wxDataObject, wxDataObjectSimple, wxTextDataObject, wxBitmapDataObject, wxDataObject

Public Member Functions

- wxFileDataObject ()

  Constructor.

- void AddFile (const wxString &file)

  Adds a file to the file list represented by this data object (Windows only).

- const wxArrayString & GetFilenames () const

  Returns the array of file names.

Additional Inherited Members

21.233.2 Constructor & Destructor Documentation

wxFileDataObject::wxFileDataObject ( )

Constructor.

21.233.3 Member Function Documentation

void wxFileDataObject::AddFile (const wxString & *file*)

Adds a file to the file list represented by this data object (Windows only).

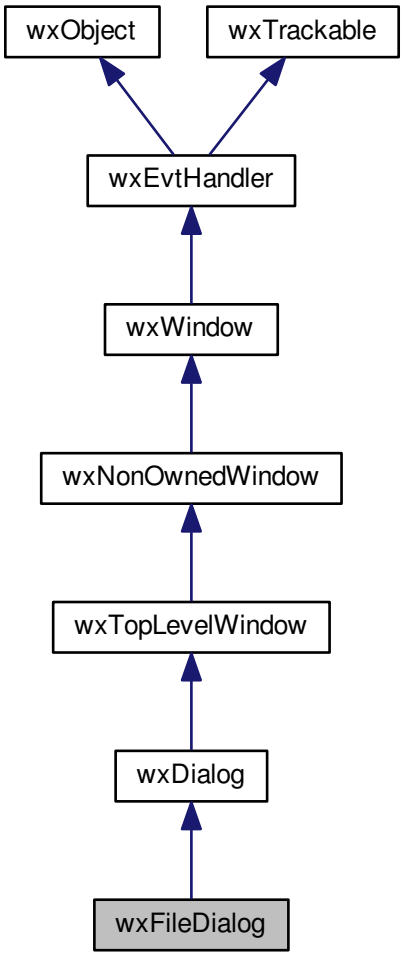
const wxArrayString& wxFileDataObject::GetFilenames ( ) const

Returns the array of file names.

21.234 wxFileDialog Class Reference

```
#include <wx/filedlg.h>
```

Inheritance diagram for wxFileDialog:



21.234.1 Detailed Description

This class represents the file chooser dialog.

The path and filename are distinct elements of a full file pathname. If path is wxEmptyString, the current directory will be used. If filename is wxEmptyString, no default filename will be supplied. The wildcard determines what files are displayed in the file selector, and file extension supplies a type extension for the required filename.

The typical usage for the open file dialog is:

```
void MyFrame::OnOpen(wxCommandEvent& WXUNUSED(event))
{
    if (...current content has not been saved...)
    {
        if (wxMessageBox(_("Current content has not been saved! Proceed?"),
      _("Please confirm"),
                         wxICON_QUESTION | wxYES_NO, this) ==
      wxNO )
            return;
        //else: proceed asking to the user the new file to open
    }
```

```
    wxFileDialog
        openFileDialog(this, _("Open XYZ file"), "", "",
                       "XYZ files (*.xyz)|*.xyz", wxFD_OPEN|
      wxFD_FILE_MUST_EXIST);

    if (openFileDialog.ShowModal() == wxID_CANCEL)
        return;     // the user changed idea...

    // proceed loading the file chosen by the user;
    // this can be done with e.g. wxWidgets input streams:
    wxFileInputStream input_stream(openFileDialog.GetPath());
    if (!input_stream.IsOk())
    {
        wxLogError("Cannot open file '%s'.", openFileDialog.GetPath());
        return;
    }

    ...
}
```

The typical usage for the save file dialog is instead somewhat simpler:

```
void MyFrame::OnSaveAs(wxCommandEvent& WXUNUSED(event))
{
    wxFileDialog
        saveFileDialog(this, _("Save XYZ file"), "", "",
                       "XYZ files (*.xyz)|*.xyz", wxFD_SAVE|
      wxFD_OVERWRITE_PROMPT);

    if (saveFileDialog.ShowModal() == wxID_CANCEL)
        return;     // the user changed idea...

    // save the current contents in the file;
    // this can be done with e.g. wxWidgets output streams:
    wxFileOutputStream output_stream(saveFileDialog.GetPath());
    if (!output_stream.IsOk())
    {
        wxLogError("Cannot save current contents in file '%s'.", saveFileDialog.GetPath());
        return;
    }

    ...
}
```

Remarks

All implementations of the wxFileDialog provide a wildcard filter. Typing a filename containing wildcards (*, ?) in the filename text item, and clicking on Ok, will result in only those files matching the pattern being displayed. The wildcard may be a specification for multiple types of file with a description for each, such as:

```
"BMP and GIF files (*.bmp;*.gif)|*.bmp;*.gif|PNG files (*.png)|*.png"
```

It must be noted that wildcard support in the native Motif file dialog is quite limited: only one file type is supported, and it is displayed without the descriptive test; "BMP files (*.bmp)|*.bmp" is displayed as "*.bmp", and both "BMP files (*.bmp)|*.bmp|GIF files (*.gif)|*.gif" and "Image files|*.bmp;*.gif" are errors.

Styles

This class supports the following styles:

- wxFD_DEFAULT_STYLE: Equivalent to `wxFD_OPEN`.
- wxFD_OPEN: This is an open dialog; usually this means that the default button's label of the dialog is "Open". Cannot be combined with `wxFD_SAVE`.
- wxFD_SAVE: This is a save dialog; usually this means that the default button's label of the dialog is "Save". Cannot be combined with `wxFD_OPEN`.
- wxFD_OVERWRITE_PROMPT: For save dialog only: prompt for a confirmation if a file will be overwritten.
- wxFD_NO_FOLLOW: Directs the dialog to return the path and file name of the selected shortcut file, not its target as it does by default. Currently this flag is only implemented in wxMSW and the non-dereferenced link path is always returned, even without this flag, under Unix and so using it there doesn't do anything. This flag was added in wxWidgets 3.1.0.

- wxFD_FILE_MUST_EXIST: For open dialog only: the user may only select files that actually exist. Notice that under OS X the file dialog with `wxFD_OPEN` style always behaves as if this style was specified, because it is impossible to choose a file that doesn't exist from a standard OS X file dialog.
- wxFD_MULTIPLE: For open dialog only: allows selecting multiple files.
- wxFD_CHANGE_DIR: Change the current working directory (when the dialog is dismissed) to the directory where the file(s) chosen by the user are.
- wxFD_PREVIEW: Show the preview of the selected files (currently only supported by wxGTK).

Library: wxCore

Category: Common Dialogs

See also

wxFileDialog Overview, wxFileSelector()

Public Types

- typedef wxWindow *(* ExtraControlCreatorFunction)(wxWindow *)

  The type of function used as an argument for SetExtraControlCreator().

Public Member Functions

- wxFileDialog (wxWindow *parent, const wxString &message=wxFileSelectorPromptStr, const wxString &defaultDir=wxEmptyString, const wxString &defaultFile=wxEmptyString, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, long style=wxFD_DEFAULT_STYLE, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, const wxString &name=wxFileDialogNameStr)

  Constructor.
- virtual ~wxFileDialog ()

  Destructor.
- virtual wxString GetCurrentlySelectedFilename () const

  Returns the path of the file currently selected in dialog.
- virtual wxString GetDirectory () const

  Returns the default directory.
- wxWindow * GetExtraControl () const

  If functions SetExtraControlCreator() and ShowModal() were called, returns the extra window.
- virtual wxString GetFilename () const

  Returns the default filename.
- virtual void GetFilenames (wxArrayString &filenames) const

  Fills the array filenames with the names of the files chosen.
- virtual int GetFilterIndex () const

  Returns the index into the list of filters supplied, optionally, in the wildcard parameter.
- virtual wxString GetMessage () const

  Returns the message that will be displayed on the dialog.
- virtual wxString GetPath () const

  Returns the full path (directory and filename) of the selected file.
- virtual void GetPaths (wxArrayString &paths) const

  Fills the array paths with the full paths of the files chosen.
- virtual wxString GetWildcard () const

Returns the file dialog wildcard.

- virtual void SetDirectory (const wxString &directory)

  Sets the default directory.
- bool SetExtraControlCreator (ExtraControlCreatorFunction creator)

  Customize file dialog by adding extra window, which is typically placed below the list of files and above the buttons.
- virtual void SetFilename (const wxString &setfilename)

  Sets the default filename.
- virtual void SetFilterIndex (int filterIndex)

  Sets the default filter index, starting from zero.
- virtual void SetMessage (const wxString &message)

  Sets the message that will be displayed on the dialog.
- virtual void SetPath (const wxString &path)

  Sets the path (the combined directory and filename that will be returned when the dialog is dismissed).
- virtual void SetWildcard (const wxString &wildCard)

  Sets the wildcard, which can contain multiple file types, for example: "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.gif".
- virtual int ShowModal ()

  Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise.

Additional Inherited Members

21.234.2 Member Typedef Documentation

typedef wxWindow∗(∗ wxFileDialog::ExtraControlCreatorFunction)(wxWindow ∗)

The type of function used as an argument for SetExtraControlCreator().

Since

2.9.0

21.234.3 Constructor & Destructor Documentation

wxFileDialog::wxFileDialog (wxWindow ∗ *parent*, const wxString & *message* = `wxFileSelectorPromptStr`, const wxString & *defaultDir* = wxEmptyString, const wxString & *defaultFile* = wxEmptyString, const wxString & *wildcard* = wxFileSelectorDefaultWildcardStr, long *style* = wxFD_DEFAULT_STYLE, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, const wxString & *name* = `wxFileDialogNameStr`)

Constructor.

Use ShowModal() to show the dialog.

Parameters

parent	Parent window.
message	Message to show on the dialog.
defaultDir	The default directory, or the empty string.
defaultFile	The default filename, or the empty string.
wildcard	A wildcard, such as "∗.∗" or "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.gif". Note that the native Motif dialog has some limitations with respect to wildcards; see the Remarks section above.

style	A dialog style. See `wxFD_*` styles for more info.
pos	Dialog position. Not implemented.
size	Dialog size. Not implemented.
name	Dialog name. Not implemented.

virtual wxFileDialog::~wxFileDialog ( ) [virtual]

Destructor.

21.234.4 Member Function Documentation

virtual wxString wxFileDialog::GetCurrentlySelectedFilename ( ) const [virtual]

Returns the path of the file currently selected in dialog.

Notice that this file is not necessarily going to be accepted by the user, so calling this function mostly makes sense from an update UI event handler of a custom file dialog extra control to update its state depending on the currently selected file.

Currently this function is fully implemented under GTK and MSW and always returns an empty string elsewhere.

Since

2.9.5

Returns

The path of the currently selected file or an empty string if nothing is selected.

See also

SetExtraControlCreator()

virtual wxString wxFileDialog::GetDirectory ( ) const [virtual]

Returns the default directory.

wxWindow* wxFileDialog::GetExtraControl ( ) const

If functions SetExtraControlCreator() and ShowModal() were called, returns the extra window.

Otherwise returns NULL.

Since

2.9.0

virtual wxString wxFileDialog::GetFilename ( ) const [virtual]

Returns the default filename.

virtual void wxFileDialog::GetFilenames (wxArrayString & *filenames*) const `[virtual]`

Fills the array *filenames* with the names of the files chosen.

This function should only be used with the dialogs which have `wxFD_MULTIPLE` style, use GetFilename() for the others.

Note that under Windows, if the user selects shortcuts, the filenames include paths, since the application cannot determine the full path of each referenced file by appending the directory containing the shortcuts to the filename.

virtual int wxFileDialog::GetFilterIndex ( ) const `[virtual]`

Returns the index into the list of filters supplied, optionally, in the wildcard parameter.

Before the dialog is shown, this is the index which will be used when the dialog is first displayed.

After the dialog is shown, this is the index selected by the user.

virtual wxString wxFileDialog::GetMessage ( ) const `[virtual]`

Returns the message that will be displayed on the dialog.

virtual wxString wxFileDialog::GetPath ( ) const `[virtual]`

Returns the full path (directory and filename) of the selected file.

virtual void wxFileDialog::GetPaths (wxArrayString & *paths*) const `[virtual]`

Fills the array *paths* with the full paths of the files chosen.

This function should only be used with the dialogs which have `wxFD_MULTIPLE` style, use GetPath() for the others.

virtual wxString wxFileDialog::GetWildcard ( ) const `[virtual]`

Returns the file dialog wildcard.

virtual void wxFileDialog::SetDirectory (const wxString & *directory*) `[virtual]`

Sets the default directory.

bool wxFileDialog::SetExtraControlCreator (ExtraControlCreatorFunction *creator*)

Customize file dialog by adding extra window, which is typically placed below the list of files and above the buttons.

SetExtraControlCreator() can be called only once, before calling ShowModal().

The `creator` function should take pointer to parent window (file dialog) and should return a window allocated with operator new.

Supported platforms: wxGTK, wxMSW, wxUniv.

Since

2.9.0

virtual void wxFileDialog::SetFilename (const wxString & *setfilename*) `[virtual]`

Sets the default filename.

In wxGTK this will have little effect unless a default directory has previously been set.

virtual void wxFileDialog::SetFilterIndex (int *filterIndex*) `[virtual]`

Sets the default filter index, starting from zero.

virtual void wxFileDialog::SetMessage (const wxString & *message*) `[virtual]`

Sets the message that will be displayed on the dialog.

virtual void wxFileDialog::SetPath (const wxString & *path*) `[virtual]`

Sets the path (the combined directory and filename that will be returned when the dialog is dismissed).

virtual void wxFileDialog::SetWildcard (const wxString & *wildCard*) `[virtual]`

Sets the wildcard, which can contain multiple file types, for example: "BMP files (∗.bmp)|∗.bmp|GIF files (∗.gif)|∗.↩ gif".

Note that the native Motif dialog has some limitations with respect to wildcards; see the Remarks section above.

virtual int wxFileDialog::ShowModal ( ) `[virtual]`

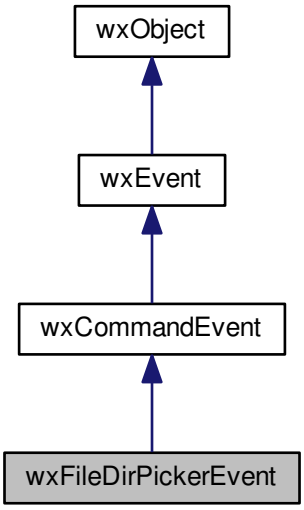
Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise.

Reimplemented from wxDialog.

21.235 wxFileDirPickerEvent Class Reference

`#include <wx/filepicker.h>`

Inheritance diagram for wxFileDirPickerEvent:



21.235.1 Detailed Description

This event class is used for the events generated by wxFilePickerCtrl and by wxDirPickerCtrl.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFileDirPickerEvent& event)

Event macros:

- EVT_FILEPICKER_CHANGED(id, func): Generated whenever the selected file changes.
- EVT_DIRPICKER_CHANGED(id, func): Generated whenever the selected directory changes.

Library: wxCore

Category: Events

See also

wxFilePickerCtrl, wxDirPickerCtrl

Public Member Functions

- wxFileDirPickerEvent ()
- wxFileDirPickerEvent (wxEventType type, wxObject *generator, int id, const wxString &path)

  The constructor is not normally used by the user code.

- wxString GetPath () const

  Retrieve the absolute path of the file/directory the user has just selected.

- void SetPath (const wxString &path)

  Set the absolute path of the file/directory associated with the event.

Additional Inherited Members

21.235.2 Constructor & Destructor Documentation

wxFileDirPickerEvent::wxFileDirPickerEvent ( )

wxFileDirPickerEvent::wxFileDirPickerEvent (wxEventType *type*, wxObject * *generator*, int *id*, const wxString & *path*)

The constructor is not normally used by the user code.

21.235.3 Member Function Documentation

wxString wxFileDirPickerEvent::GetPath ( ) const

Retrieve the absolute path of the file/directory the user has just selected.

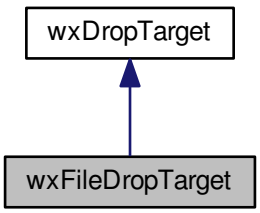
void wxFileDirPickerEvent::SetPath (const wxString & *path*)

Set the absolute path of the file/directory associated with the event.

21.236 wxFileDropTarget Class Reference

`#include <wx/dnd.h>`

Inheritance diagram for wxFileDropTarget:



21.236.1 Detailed Description

This is a drop target which accepts files (dragged from File Manager or Explorer).

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDropSource, wxDropTarget, wxTextDropTarget

Public Member Functions

- wxFileDropTarget ()

  Constructor.
- virtual bool OnDrop (wxCoord x, wxCoord y)

  See wxDropTarget::OnDrop().
- virtual bool OnDropFiles (wxCoord x, wxCoord y, const wxArrayString &filenames)=0

  Override this function to receive dropped files.

21.236.2 Constructor & Destructor Documentation

wxFileDropTarget::wxFileDropTarget ( )

Constructor.

21.236.3 Member Function Documentation

virtual bool wxFileDropTarget::OnDrop (wxCoord *x*, wxCoord *y*) `[virtual]`

See wxDropTarget::OnDrop().

This function is implemented appropriately for files, and calls OnDropFiles().

Reimplemented from wxDropTarget.

virtual bool wxFileDropTarget::OnDropFiles (wxCoord *x*, wxCoord *y*, const wxArrayString & *filenames*) `[pure virtual]`

Override this function to receive dropped files.

Parameters

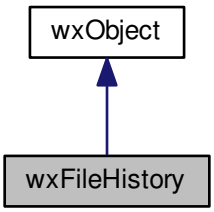
x	The x coordinate of the mouse.
y	The y coordinate of the mouse.
filenames	An array of filenames.

Return true to accept the data, or false to veto the operation.

21.237 wxFileHistory Class Reference

`#include <wx/filehistory.h>`

Inheritance diagram for wxFileHistory:



21.237.1 Detailed Description

The wxFileHistory encapsulates a user interface convenience, the list of most recently visited files as shown on a menu (usually the File menu).

wxFileHistory can manage one or more file menus. More than one menu may be required in an MDI application, where the file history should appear on each MDI child menu as well as the MDI parent frame.

Library: wxCore

Category: Document/View Framework

See also

Document/View Framework, wxDocManager

Public Member Functions

- wxFileHistory (size_t maxFiles=9, wxWindowID idBase=wxID_FILE1)

  Constructor.
- virtual ~wxFileHistory ()

  Destructor.
- virtual void AddFileToHistory (const wxString &filename)

  Adds a file to the file history list, if the object has a pointer to an appropriate file menu.
- virtual void AddFilesToMenu ()

  Appends the files in the history list, to all menus managed by the file history object.
- virtual void AddFilesToMenu (wxMenu *menu)

  Appends the files in the history list, to the given menu only.
- wxWindowID GetBaseId () const

  Returns the base identifier for the range used for appending items.
- virtual size_t GetCount () const

  Returns the number of files currently stored in the file history.
- virtual wxString GetHistoryFile (size_t index) const

  Returns the file at this index (zero-based).
- virtual int GetMaxFiles () const

Returns the maximum number of files that can be stored.

- const wxList & GetMenus () const

  Returns the list of menus that are managed by this file history object.

- virtual void Load (const wxConfigBase &config)

  Loads the file history from the given config object.

- virtual void RemoveFileFromHistory (size_t i)

  Removes the specified file from the history.

- virtual void RemoveMenu (wxMenu *menu)

  Removes this menu from the list of those managed by this object.

- virtual void Save (wxConfigBase &config)

  Saves the file history into the given config object.

- void SetBaseId (wxWindowID baseId)

  Sets the base identifier for the range used for appending items.

- virtual void UseMenu (wxMenu *menu)

  Adds this menu to the list of those menus that are managed by this file history object.

Additional Inherited Members

21.237.2 Constructor & Destructor Documentation

wxFileHistory::wxFileHistory (size_t *maxFiles* = 9, wxWindowID *idBase* = wxID_FILE1)

Constructor.

Pass the maximum number of files that should be stored and displayed.

idBase defaults to wxID_FILE1 and represents the id given to the first history menu item. Since menu items can't share the same ID you should change *idBase* (to one of your own defined IDs) when using more than one wxFileHistory in your application.

virtual wxFileHistory::~wxFileHistory ( ) [virtual]

Destructor.

21.237.3 Member Function Documentation

virtual void wxFileHistory::AddFilesToMenu ( ) [virtual]

Appends the files in the history list, to all menus managed by the file history object.

virtual void wxFileHistory::AddFilesToMenu (wxMenu * *menu*) [virtual]

Appends the files in the history list, to the given menu only.

virtual void wxFileHistory::AddFileToHistory (const wxString & *filename*) [virtual]

Adds a file to the file history list, if the object has a pointer to an appropriate file menu.

wxWindowID wxFileHistory::GetBaseId ( ) const

Returns the base identifier for the range used for appending items.

virtual size_t wxFileHistory::GetCount ( ) const `[virtual]`

Returns the number of files currently stored in the file history.

virtual wxString wxFileHistory::GetHistoryFile (size_t *index*) const `[virtual]`

Returns the file at this index (zero-based).

virtual int wxFileHistory::GetMaxFiles ( ) const `[virtual]`

Returns the maximum number of files that can be stored.

const wxList& wxFileHistory::GetMenus ( ) const

Returns the list of menus that are managed by this file history object.

See also

UseMenu()

virtual void wxFileHistory::Load (const wxConfigBase & *config*) `[virtual]`

Loads the file history from the given config object.

This function should be called explicitly by the application.

See also

wxConfigBase

virtual void wxFileHistory::RemoveFileFromHistory (size_t *i*) `[virtual]`

Removes the specified file from the history.

virtual void wxFileHistory::RemoveMenu (wxMenu * *menu*) `[virtual]`

Removes this menu from the list of those managed by this object.

virtual void wxFileHistory::Save (wxConfigBase & *config*) `[virtual]`

Saves the file history into the given config object.

This must be called explicitly by the application.

See also

wxConfigBase

void wxFileHistory::SetBaseId (wxWindowID *baseId*)

Sets the base identifier for the range used for appending items.

virtual void wxFileHistory::UseMenu (wxMenu * *menu*) [virtual]

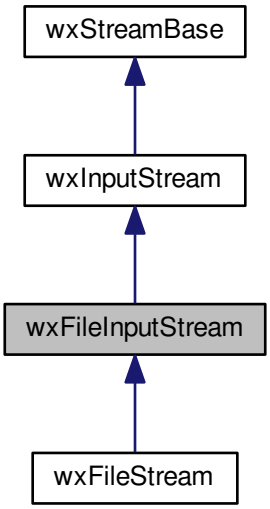
Adds this menu to the list of those menus that are managed by this file history object.

Also see AddFilesToMenu() for initializing the menu with filenames that are already in the history when this function is called, as this is not done automatically.

21.238 wxFileInputStream Class Reference

`#include <wx/wfstream.h>`

Inheritance diagram for wxFileInputStream:



21.238.1 Detailed Description

This class represents data read in from a file.

There are actually two such groups of classes: this one is based on wxFile whereas wxFFileInputStream is based in the wxFFile class.

Note that wxInputStream::SeekI() can seek beyond the end of the stream (file) and will thus not return wxInvalidOffset for that.

Library: wxBase

Category: Streams

See also

wxBufferedInputStream, wxFileOutputStream, wxFFileOutputStream

Public Member Functions

- wxFileInputStream (const wxString &ifileName)

  Opens the specified file using its ifileName name in read-only mode.
- wxFileInputStream (wxFile &file)

  Initializes a file stream in read-only mode using the file I/O object file.
- wxFileInputStream (int fd)

  Initializes a file stream in read-only mode using the specified file descriptor.
- virtual ~wxFileInputStream ()

  Destructor.
- bool IsOk () const

  Returns true if the stream is initialized and ready.
- wxFile * GetFile () const

  Returns the underlying file object.

Additional Inherited Members

21.238.2 Constructor & Destructor Documentation

wxFileInputStream::wxFileInputStream (const wxString & *ifileName*)

Opens the specified file using its *ifileName* name in read-only mode.

Warning

You should use wxStreamBase::IsOk() to verify if the constructor succeeded.

wxFileInputStream::wxFileInputStream (wxFile & *file*)

Initializes a file stream in read-only mode using the file I/O object file.

wxFileInputStream::wxFileInputStream (int *fd*)

Initializes a file stream in read-only mode using the specified file descriptor.

virtual wxFileInputStream::~wxFileInputStream ( ) [virtual]

Destructor.

21.238.3 Member Function Documentation

wxFile* wxFileInputStream::GetFile ( ) const

Returns the underlying file object.

Since

2.9.5

bool wxFileInputStream::IsOk ( ) const `[virtual]`

Returns true if the stream is initialized and ready.

Reimplemented from wxStreamBase.

Reimplemented in wxFileStream.

21.239 wxFileName Class Reference

`#include <wx/filename.h>`

21.239.1 Detailed Description

wxFileName encapsulates a file name.

This class serves two purposes: first, it provides the functions to split the file names into components and to recombine these components in the full file name which can then be passed to the OS file functions (and wxWidgets functions wrapping them). Second, it includes the functions for working with the files itself. Note that to change the file data you should use wxFile class instead. wxFileName provides functions for working with the file attributes.

When working with directory names (i.e. without filename and extension) make sure not to misuse the file name part of this class with the last directory. Instead initialize the wxFileName instance like this:

```
wxFileName dirname( "C:\mydir", "" );
MyMethod( dirname.GetPath() );
```

The same can be done using the static method wxFileName::DirName():

```
wxFileName dirname = wxFileName::DirName( "C:\mydir" );
MyMethod( dirname.GetPath() );
```

Accordingly, methods dealing with directories or directory names like wxFileName::IsDirReadable() use wxFileName::GetPath() whereas methods dealing with file names like wxFileName::IsFileReadable() use wxFileName::GetFullPath().

If it is not known whether a string contains a directory name or a complete file name (such as when interpreting user input) you need to use the static function wxFileName::DirExists() (or its identical variants wxDir::Exists() and wxDirExists()) and construct the wxFileName instance accordingly. This will only work if the directory actually exists, of course:

```
wxString user_input;
// get input from user

wxFileName fname;
if (wxDirExists(user_input))
    fname.AssignDir( user_input );
else
    fname.Assign( user_input );
```

Please note that many wxFileName methods accept the path format argument which is by `wxPATH_NATIVE` by default meaning to use the path format native for the current platform. The path format affects the operation of wxFileName functions in several ways: first and foremost, it defines the path separator character to use, but it also affects other things such as whether the path has the drive part or not. See wxPathFormat for more info.

21.239.2 File name format

wxFileName currently supports the file names in the Unix, DOS/Windows, Mac OS and VMS formats. Although these formats are quite different, wxFileName tries to treat them all in the same generic way. It supposes that all file

names consist of the following parts: the volume (also known as drive under Windows or device under VMS), the path which is a sequence of directory names separated by the path separators and the full filename itself which, in turn, is composed from the base file name and the extension. All of the individual components of the file name may be empty and, for example, the volume name is always empty under Unix, but if they are all empty simultaneously, the filename object is considered to be in an invalid state and wxFileName::IsOk() returns false for it.

File names can be case-sensitive or not, the function wxFileName::IsCaseSensitive() allows to determine this. The rules for determining whether the file name is absolute or relative also depend on the file name format and the only portable way to answer this question is to use wxFileName::IsAbsolute() or wxFileName::IsRelative() method.

Note that on Windows,"X:" refers to the current working directory on drive X. Therefore, a wxFileName instance constructed from for example "X:dir/file.ext" treats the portion beyond drive separator as being relative to that directory. To ensure that the filename is absolute, you may use wxFileName::MakeAbsolute(). There is also an inverse function wxFileName::MakeRelativeTo() which undoes what wxFileName::Normalize(wxPATH_NORM_DOTS) does. Other functions returning information about the file format provided by this class are wxFileName::GetVolumeSeparator(), wxFileName::IsPathSeparator().

21.239.3 File name construction

You can initialize a wxFileName instance using one of the following functions:

- wxFileName::wxFileName()
- wxFileName::Assign()
- wxFileName::AssignCwd()
- wxFileName::AssignDir()
- wxFileName::AssignHomeDir()
- wxFileName::AssignTempFileName()
- wxFileName::DirName()
- wxFileName::FileName()
- wxFileName::operator=()

21.239.4 File name tests

Before doing other tests, you should use wxFileName::IsOk() to verify that the filename is well defined. If it is, FileExists() can be used to test whether a file with such name exists and wxFileName::DirExists() can be used to test for directory existence. File names should be compared using the wxFileName::SameAs() method or wxFileName::operator==(). For testing basic access modes, you can use:

- wxFileName::IsDirWritable()
- wxFileName::IsDirReadable()
- wxFileName::IsFileWritable()
- wxFileName::IsFileReadable()
- wxFileName::IsFileExecutable()

21.239.5 File name components

These functions allow to examine and modify the individual directories of the path:

- wxFileName::AppendDir()
- wxFileName::InsertDir()
- wxFileName::GetDirCount()
- wxFileName::PrependDir()
- wxFileName::RemoveDir()
- wxFileName::RemoveLastDir()

To change the components of the file name individually you can use the following functions:

- wxFileName::GetExt()
- wxFileName::GetName()
- wxFileName::GetVolume()
- wxFileName::HasExt()
- wxFileName::HasName()
- wxFileName::HasVolume()
- wxFileName::SetExt()
- wxFileName::ClearExt()
- wxFileName::SetEmptyExt()
- wxFileName::SetName()
- wxFileName::SetVolume()

You can initialize a wxFileName instance using one of the following functions:

21.239.6 File name operations

These methods allow to work with the file creation, access and modification times. Note that not all filesystems under all platforms implement these times in the same way. For example, the access time under Windows has a resolution of one day (so it is really the access date and not time). The access time may be updated when the file is executed or not depending on the platform.

- wxFileName::GetModificationTime()
- wxFileName::GetTimes()
- wxFileName::SetTimes()
- wxFileName::Touch()

Other file system operations functions are:

- wxFileName::Mkdir()
- wxFileName::Rmdir()

Library: wxBase

Category: File Handling

- wxString GetHumanReadableSize (const wxString &failmsg=_("Not available"), int precision=1, wxSizeConvention conv=wxSIZE_CONV_TRADITIONAL) const

  Returns the representation of the file size in a human-readable form.
- static wxString GetHumanReadableSize (const wxULongLong &bytes, const wxString &nullsize=_("Not available"), int precision=1, wxSizeConvention conv=wxSIZE_CONV_TRADITIONAL)

  Returns the representation of the file size in a human-readable form.

Public Member Functions

- wxFileName ()

  Default constructor.
- wxFileName (const wxFileName &filename)

  Copy constructor.
- wxFileName (const wxString &fullpath, wxPathFormat format=wxPATH_NATIVE)

  Constructor taking a full filename.
- wxFileName (const wxString &path, const wxString &name, wxPathFormat format=wxPATH_NATIVE)

  Constructor a directory name and file name.
- wxFileName (const wxString &path, const wxString &name, const wxString &ext, wxPathFormat format=wxPATH_NATIVE)

  Constructor from a directory name, base file name and extension.
- wxFileName (const wxString &volume, const wxString &path, const wxString &name, const wxString &ext, wxPathFormat format=wxPATH_NATIVE)

  Constructor from a volume name, a directory name, base file name and extension.
- bool AppendDir (const wxString &dir)

  Appends a directory component to the path.
- void Assign (const wxFileName &filepath)

  Creates the file name from another filename object.
- void Assign (const wxString &fullpath, wxPathFormat format=wxPATH_NATIVE)

  Creates the file name from a full file name with a path.
- void Assign (const wxString &volume, const wxString &path, const wxString &name, const wxString &ext, bool hasExt, wxPathFormat format=wxPATH_NATIVE)

  Creates the file name from volume, path, name and extension.
- void Assign (const wxString &volume, const wxString &path, const wxString &name, const wxString &ext, wxPathFormat format=wxPATH_NATIVE)

  Creates the file name from volume, path, name and extension.
- void Assign (const wxString &path, const wxString &name, wxPathFormat format=wxPATH_NATIVE)

  Creates the file name from file path and file name.
- void Assign (const wxString &path, const wxString &name, const wxString &ext, wxPathFormat format=wxPATH_NATIVE)

  Creates the file name from path, name and extension.
- void AssignCwd (const wxString &volume=wxEmptyString)

  Makes this object refer to the current working directory on the specified volume (or current volume if volume is empty).
- void AssignDir (const wxString &dir, wxPathFormat format=wxPATH_NATIVE)

  Sets this file name object to the given directory name.
- void AssignHomeDir ()

  Sets this file name object to the home directory.

- void AssignTempFileName (const wxString &prefix)

  The function calls CreateTempFileName() to create a temporary file and sets this object to the name of the file.
- void AssignTempFileName (const wxString &prefix, wxFile *fileTemp)

  The function calls CreateTempFileName() to create a temporary file name and open fileTemp with it.
- void AssignTempFileName (const wxString &prefix, wxFFile *fileTemp)

  The function calls CreateTempFileName() to create a temporary file name and open fileTemp with it.
- void Clear ()

  Reset all components to default, uninitialized state.
- void ClearExt ()

  Removes the extension from the file name resulting in a file name with no trailing dot.
- bool DirExists () const

  Returns true if the directory with this name exists.
- void DontFollowLink ()

  Turns off symlink dereferencing.
- bool Exists (int flags=wxFILE_EXISTS_ANY) const

  Calls the static overload of this function with the full path of this object.
- bool FileExists () const

  Returns true if the file with this name exists.
- size_t GetDirCount () const

  Returns the number of directories in the file name.
- const wxArrayString & GetDirs () const

  Returns the directories in string array form.
- wxString GetExt () const

  Returns the file name extension.
- wxString GetFullName () const

  Returns the full name (including extension but excluding directories).
- wxString GetFullPath (wxPathFormat format=wxPATH_NATIVE) const

  Returns the full path with name and extension.
- wxString GetLongPath () const

  Return the long form of the path (returns identity on non-Windows platforms).
- wxDateTime GetModificationTime () const

  Returns the last time the file was last modified.
- wxString GetName () const

  Returns the name part of the filename (without extension).
- wxString GetPath (int flags=wxPATH_GET_VOLUME, wxPathFormat format=wxPATH_NATIVE) const

  Returns the path part of the filename (without the name or extension).
- wxString GetPathWithSep (wxPathFormat format=wxPATH_NATIVE) const

  Returns the path with the trailing separator, useful for appending the name to the given path.
- wxString GetShortPath () const

  Return the short form of the path (returns identity on non-Windows platforms).
- wxULongLong GetSize () const

  Returns the size of the file If the file does not exist or its size could not be read (because e.g.
- bool GetTimes (wxDateTime *dtAccess, wxDateTime *dtMod, wxDateTime *dtCreate) const

  Returns the last access, last modification and creation times.
- wxString GetVolume () const

  Returns the string containing the volume for this file name, empty if it doesn't have one or if the file system doesn't support volumes at all (for example, Unix).
- bool HasExt () const

  Returns true if an extension is present.
- bool HasName () const

  Returns true if a name is present.

- bool HasVolume () const

  Returns true if a volume specifier is present.
- bool InsertDir (size_t before, const wxString &dir)

  Inserts a directory component before the zero-based position in the directory list.
- bool IsAbsolute (wxPathFormat format=wxPATH_NATIVE) const

  Returns true if this filename is absolute.
- bool IsDir () const

  Returns true if this object represents a directory, false otherwise (i.e.
- bool IsDirReadable () const

  Returns true if the directory component of this instance is an existing directory and this process has read permissions on it.
- bool IsDirWritable () const

  Returns true if the directory component of this instance is an existing directory and this process has write permissions on it.
- bool IsFileExecutable () const

  Returns true if a file with this name exists and if this process has execute permissions on it.
- bool IsFileReadable () const

  Returns true if a file with this name exists and if this process has read permissions on it.
- bool IsFileWritable () const

  Returns true if a file with this name exists and if this process has write permissions on it.
- bool IsOk () const

  Returns true if the filename is valid, false if it is not initialized yet.
- bool IsRelative (wxPathFormat format=wxPATH_NATIVE) const

  Returns true if this filename is not absolute.
- bool MacSetDefaultTypeAndCreator ()

  On Mac OS, looks up the appropriate type and creator from the registration and then sets it.
- bool MakeAbsolute (const wxString &cwd=wxEmptyString, wxPathFormat format=wxPATH_NATIVE)

  Make the file name absolute.
- bool MakeRelativeTo (const wxString &pathBase=wxEmptyString, wxPathFormat format=wxPATH_NATIVE)

  This function tries to put this file name in a form relative to pathBase.
- bool Mkdir (int perm=wxS_DIR_DEFAULT, int flags=0) const

  Creates a directory.
- bool Normalize (int flags=wxPATH_NORM_ALL, const wxString &cwd=wxEmptyString, wxPathFormat format=wxPATH_NATIVE)

  Normalize the path.
- void PrependDir (const wxString &dir)

  Prepends a directory to the file path.
- void RemoveDir (size_t pos)

  Removes the specified directory component from the path.
- void RemoveLastDir ()

  Removes last directory component from the path.
- bool ReplaceEnvVariable (const wxString &envname, const wxString &replacementFmtString="$%s", wxPathFormat format=wxPATH_NATIVE)

  If the path contains the value of the environment variable named envname then this function replaces it with the string obtained from wxString::Format(replacementFmtString, value_of_envname_variable).
- bool ReplaceHomeDir (wxPathFormat format=wxPATH_NATIVE)

  Replaces, if present in the path, the home directory for the given user (see wxGetHomeDir) with a tilde (~).
- bool Rmdir (int flags=0) const

  Deletes the specified directory from the file system.
- bool SameAs (const wxFileName &filepath, wxPathFormat format=wxPATH_NATIVE) const

  Compares the filename using the rules of this platform.

- bool SetCwd () const

  Changes the current working directory.
- void SetEmptyExt ()

  Sets the extension of the file name to be an empty extension.
- void SetExt (const wxString &ext)

  Sets the extension of the file name.
- void SetFullName (const wxString &fullname)

  The full name is the file name and extension (but without the path).
- void SetName (const wxString &name)

  Sets the name part (without extension).
- void SetPath (const wxString &path, wxPathFormat format=wxPATH_NATIVE)

  Sets the path.
- bool SetPermissions (int permissions)

  Sets permissions for this file or directory.
- bool SetTimes (const wxDateTime *dtAccess, const wxDateTime *dtMod, const wxDateTime *dtCreate) const

  Sets the file creation and last access/modification times (any of the pointers may be NULL).
- void SetVolume (const wxString &volume)

  Sets the volume specifier.
- bool ShouldFollowLink () const

  Return whether some operations will follow symlink.
- bool Touch () const

  Sets the access and modification times to the current moment.
- bool operator!= (const wxFileName &filename) const

  Returns true if the filenames are different.
- bool operator!= (const wxString &filename) const

  Returns true if the filenames are different.
- bool operator== (const wxFileName &filename) const

  Returns true if the filenames are equal.
- bool operator== (const wxString &filename) const

  Returns true if the filenames are equal.
- wxFileName & operator= (const wxFileName &filename)

  Assigns the new value to this filename object.
- wxFileName & operator= (const wxString &filename)

  Assigns the new value to this filename object.

Static Public Member Functions

- static wxString CreateTempFileName (const wxString &prefix, wxFile *fileTemp=NULL)

  Returns a temporary file name starting with the given prefix.
- static wxString CreateTempFileName (const wxString &prefix, wxFFile *fileTemp=NULL)

  *This is the same as CreateTempFileName(const wxString &prefix, wxFile *fileTemp) but takes a wxFFile parameter instead of wxFile.*
- static bool DirExists (const wxString &dir)

  Returns true if the directory with name dir exists.
- static wxFileName DirName (const wxString &dir, wxPathFormat format=wxPATH_NATIVE)

  Returns the object corresponding to the directory with the given name.
- static bool Exists (const wxString &path, int flags=wxFILE_EXISTS_ANY)

  Returns true if either a file or a directory or something else with this name exists in the file system.
- static bool FileExists (const wxString &file)

  Returns true if the file with name file exists.

- static wxFileName FileName (const wxString &file, wxPathFormat format=wxPATH_NATIVE)

  Returns the file name object corresponding to the given file.
- static wxString GetCwd (const wxString &volume=wxEmptyString)

  Retrieves the value of the current working directory on the specified volume.
- static wxString GetForbiddenChars (wxPathFormat format=wxPATH_NATIVE)

  Returns the characters that can't be used in filenames and directory names for the specified format.
- static wxPathFormat GetFormat (wxPathFormat format=wxPATH_NATIVE)

  Returns the canonical path format for this platform.
- static wxString GetHomeDir ()

  Returns the home directory.
- static wxUniChar GetPathSeparator (wxPathFormat format=wxPATH_NATIVE)

  Returns the usually used path separator for this format.
- static wxString GetPathSeparators (wxPathFormat format=wxPATH_NATIVE)

  Returns the string containing all the path separators for this format.
- static wxString GetPathTerminators (wxPathFormat format=wxPATH_NATIVE)

  Returns the string of characters which may terminate the path part.
- static wxULongLong GetSize (const wxString &filename)

  Returns the size of the file If the file does not exist or its size could not be read (because e.g.
- static wxString GetTempDir ()

  Returns the directory used for temporary files.
- static wxString GetVolumeSeparator (wxPathFormat format=wxPATH_NATIVE)

  Returns the string separating the volume from the path for this format.
- static wxString GetVolumeString (char drive, int flags=wxPATH_GET_SEPARATOR)

  This function builds a volume path string, for example "C:\\".
- static bool IsCaseSensitive (wxPathFormat format=wxPATH_NATIVE)

  Returns true if the file names of this type are case-sensitive.
- static bool IsDirReadable (const wxString &dir)

  Returns true if the given dir is an existing directory and this process has read permissions on it.
- static bool IsDirWritable (const wxString &dir)

  Returns true if the given dir is an existing directory and this process has write permissions on it.
- static bool IsFileExecutable (const wxString &file)

  Returns true if a file with this name exists and if this process has execute permissions on it.
- static bool IsFileReadable (const wxString &file)

  Returns true if a file with this name exists and if this process has read permissions on it.
- static bool IsFileWritable (const wxString &file)

  Returns true if a file with this name exists and if this process has write permissions on it.
- static bool IsPathSeparator (wxChar ch, wxPathFormat format=wxPATH_NATIVE)

  Returns true if the char is a path separator for this format.
- static bool IsMSWUniqueVolumeNamePath (const wxString &path, wxPathFormat format=wxPATH_NATIVE)

  Returns true if the volume part of the path is a unique volume name.
- static bool MacFindDefaultTypeAndCreator (const wxString &ext, wxUint32 *type, wxUint32 *creator)

  On Mac OS, gets the common type and creator for the given extension.
- static void MacRegisterDefaultTypeAndCreator (const wxString &ext, wxUint32 type, wxUint32 creator)

  On Mac OS, registers application defined extensions and their default type and creator.
- static bool Mkdir (const wxString &dir, int perm=wxS_DIR_DEFAULT, int flags=0)

  Creates a directory.
- static bool Rmdir (const wxString &dir, int flags=0)

  Deletes the specified directory from the file system.
- static bool SetCwd (const wxString &cwd)

  Changes the current working directory.

- static void SplitVolume (const wxString &fullpath, wxString ∗volume, wxString ∗path, wxPathFormat format=wxPATH_NATIVE)

  Splits the given fullpath into the volume part (which may be empty) and the pure path part, not containing any volume.
- static wxString StripExtension (const wxString &fullname)

  Strip the file extension.

- static void SplitPath (const wxString &fullpath, wxString ∗volume, wxString ∗path, wxString ∗name, wxString ∗ext, bool ∗hasExt=NULL, wxPathFormat format=wxPATH_NATIVE)

  This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension.
- static void SplitPath (const wxString &fullpath, wxString ∗volume, wxString ∗path, wxString ∗name, wxString ∗ext, wxPathFormat format)

  This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension.
- static void SplitPath (const wxString &fullpath, wxString ∗path, wxString ∗name, wxString ∗ext, wxPathFormat format=wxPATH_NATIVE)

  This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension.

21.239.7 Constructor & Destructor Documentation

wxFileName::wxFileName ( )

Default constructor.

wxFileName::wxFileName (const wxFileName & *filename*)

Copy constructor.

wxFileName::wxFileName (const wxString & *fullpath,* wxPathFormat *format* = wxPATH_NATIVE)

Constructor taking a full filename.

If it terminates with a '/', a directory path is constructed (the name will be empty), otherwise a file name and extension are extracted from it.

wxFileName::wxFileName (const wxString & *path,* const wxString & *name,* wxPathFormat *format* = wxPATH_NATIVE)

Constructor a directory name and file name.

wxFileName::wxFileName (const wxString & *path,* const wxString & *name,* const wxString & *ext,* wxPathFormat *format* = wxPATH_NATIVE)

Constructor from a directory name, base file name and extension.

wxFileName::wxFileName (const wxString & *volume,* const wxString & *path,* const wxString & *name,* const wxString & *ext,* wxPathFormat *format* = wxPATH_NATIVE)

Constructor from a volume name, a directory name, base file name and extension.

21.239.8 Member Function Documentation

bool wxFileName::AppendDir (const wxString & *dir*)

Appends a directory component to the path.

This component should contain a single directory name level, i.e. not contain any path or volume separators nor should it be empty, otherwise the function does nothing and returns false (and generates an assert failure in debug build).

Notice that the return value is only available in wxWidgets 2.9.5 or later.

void wxFileName::Assign (const wxFileName & *filepath*)

Creates the file name from another filename object.

void wxFileName::Assign (const wxString & *fullpath,* wxPathFormat *format* = wxPATH_NATIVE)

Creates the file name from a full file name with a path.

void wxFileName::Assign (const wxString & *volume,* const wxString & *path,* const wxString & *name,* const wxString & *ext,* bool *hasExt,* wxPathFormat *format* = wxPATH_NATIVE)

Creates the file name from volume, path, name and extension.

void wxFileName::Assign (const wxString & *volume,* const wxString & *path,* const wxString & *name,* const wxString & *ext,* wxPathFormat *format* = wxPATH_NATIVE)

Creates the file name from volume, path, name and extension.

void wxFileName::Assign (const wxString & *path,* const wxString & *name,* wxPathFormat *format* = wxPATH_NATIVE)

Creates the file name from file path and file name.

void wxFileName::Assign (const wxString & *path,* const wxString & *name,* const wxString & *ext,* wxPathFormat *format* = wxPATH_NATIVE)

Creates the file name from path, name and extension.

void wxFileName::AssignCwd (const wxString & *volume* = wxEmptyString)

Makes this object refer to the current working directory on the specified volume (or current volume if *volume* is empty).

See also

GetCwd()

void wxFileName::AssignDir (const wxString & *dir,* wxPathFormat *format* = wxPATH_NATIVE)

Sets this file name object to the given directory name.

The name and extension will be empty.

void wxFileName::AssignHomeDir ( )

Sets this file name object to the home directory.

void wxFileName::AssignTempFileName (const wxString & *prefix*)

The function calls CreateTempFileName() to create a temporary file and sets this object to the name of the file.

If a temporary file couldn't be created, the object is put into an invalid state (see IsOk()).

void wxFileName::AssignTempFileName (const wxString & *prefix,* wxFile * *fileTemp*)

The function calls CreateTempFileName() to create a temporary file name and open *fileTemp* with it.

If the file couldn't be opened, the object is put into an invalid state (see IsOk()).

void wxFileName::AssignTempFileName (const wxString & *prefix,* wxFFile * *fileTemp*)

The function calls CreateTempFileName() to create a temporary file name and open *fileTemp* with it.

If the file couldn't be opened, the object is put into an invalid state (see IsOk()).

void wxFileName::Clear ( )

Reset all components to default, uninitialized state.

void wxFileName::ClearExt ( )

Removes the extension from the file name resulting in a file name with no trailing dot.

See also

SetExt(), SetEmptyExt()

static wxString wxFileName::CreateTempFileName (const wxString & *prefix,* wxFile * *fileTemp* = NULL) [static]

Returns a temporary file name starting with the given *prefix*.

If *prefix* is an absolute path and ends in a separator, the temporary file is created in this directory; if it is an absolute filepath or there is no separator, the temporary file is created in its path, with the 'name' segment prepended to the temporary filename; otherwise it is created in the default system directory for temporary files or in the current directory.

If the function succeeds, the temporary file is actually created. If *fileTemp* is not NULL, this wxFile will be opened using the name of the temporary file. Where possible this is done in an atomic way to ensure that no race condition occurs between creating the temporary file name and opening it, which might lead to a security compromise on multiuser systems. If *fileTemp* is NULL, the file is created but not opened. Under Unix, the temporary file will have read and write permissions for the owner only, to minimize security problems.

Parameters

prefix	Location to use for the temporary file name construction. If *prefix* is a directory it must have a terminal separator

fileTemp	The file to open, or NULL just to get the name

Returns

The full temporary filepath, or an empty string on error.

static wxString wxFileName::CreateTempFileName (const wxString & *prefix,* wxFFile * *fileTemp* = NULL) [static]

This is the same as CreateTempFileName(const wxString &prefix, wxFile *fileTemp) but takes a wxFFile parameter instead of wxFile.

bool wxFileName::DirExists ( ) const

Returns true if the directory with this name exists.

Notice that this function tests the directory part of this object, i.e. the string returned by GetPath(), and not the full path returned by GetFullPath().

See also

FileExists(), Exists()

static bool wxFileName::DirExists (const wxString & *dir*) [static]

Returns true if the directory with name *dir* exists.

See also

FileExists(), Exists()

static wxFileName wxFileName::DirName (const wxString & *dir,* wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the object corresponding to the directory with the given name.

The *dir* parameter may have trailing path separator or not.

void wxFileName::DontFollowLink ( )

Turns off symlink dereferencing.

By default, all operations in this class work on the target of a symbolic link (symlink) if the path of the file is actually a symlink. Using this method allows to turn off this "symlink following" behaviour and apply the operations to this path itself, even if it is a symlink.

The following methods are currently affected by this option:

- GetTimes() (but not SetTimes() as there is no portable way to change the time of symlink itself).
- Existence checks: FileExists(), DirExists() and Exists() (notice that static versions of these methods always follow symlinks).
- IsSameAs().

See also

ShouldFollowLink()

Since

2.9.5

bool wxFileName::Exists (int *flags* = `wxFILE_EXISTS_ANY`) const

Calls the static overload of this function with the full path of this object.

Since

2.9.4 (*flags* is new since 2.9.5)

static bool wxFileName::Exists (const wxString & *path*, int *flags* = `wxFILE_EXISTS_ANY`) [static]

Returns true if either a file or a directory or something else with this name exists in the file system.

Don't dereference *path* if it is a symbolic link and *flags* argument contains wxFILE_EXISTS_NO_FOLLOW.

This method is equivalent to

```
FileExists() || DirExists()
```

under Windows, but under Unix it also returns true if the file identifies a special file system object such as a device, a socket or a FIFO.

Alternatively you may check for the existence of a file system entry of a specific type by passing the appropriate *flags* (this parameter is new since wxWidgets 2.9.5). E.g. to test for a symbolic link existence you could use wxFILE_EXISTS_SYMLINK.

Since

2.9.4

See also

FileExists(), DirExists()

bool wxFileName::FileExists ( ) const

Returns true if the file with this name exists.

See also

DirExists(), Exists()

static bool wxFileName::FileExists (const wxString & *file*) [static]

Returns true if the file with name *file* exists.

See also

DirExists(), Exists()

static wxFileName wxFileName::FileName (const wxString & *file,* wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the file name object corresponding to the given *file*.

This function exists mainly for symmetry with DirName().

static wxString wxFileName::GetCwd (const wxString & *volume* = wxEmptyString) [static]

Retrieves the value of the current working directory on the specified volume.

If the volume is empty, the program's current working directory is returned for the current volume.

Returns

The string containing the current working directory or an empty string on error.

See also

AssignCwd()

size_t wxFileName::GetDirCount ( ) const

Returns the number of directories in the file name.

const wxArrayString& wxFileName::GetDirs ( ) const

Returns the directories in string array form.

wxString wxFileName::GetExt ( ) const

Returns the file name extension.

static wxString wxFileName::GetForbiddenChars (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the characters that can't be used in filenames and directory names for the specified format.

static wxPathFormat wxFileName::GetFormat (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the canonical path format for this platform.

wxString wxFileName::GetFullName ( ) const

Returns the full name (including extension but excluding directories).

wxString wxFileName::GetFullPath (wxPathFormat *format* = wxPATH_NATIVE) const

Returns the full path with name and extension.

static wxString wxFileName::GetHomeDir ( ) [static]

Returns the home directory.

wxString wxFileName::GetHumanReadableSize (const wxString & *failmsg* = _("Not available"), int *precision* = 1, wxSizeConvention *conv* = wxSIZE_CONV_TRADITIONAL) const

Returns the representation of the file size in a human-readable form.

In the first version, the size of this file is used. In the second one, the specified size *bytes* is used.

If the file size could not be retrieved or *bytes* is wxInvalidSize or zero, the `failmsg` string is returned.

Otherwise the returned string is a floating-point number with `precision` decimal digits followed by the abbreviation of the unit used. By default the traditional, although incorrect, convention of using SI units for multiples of 1024 is used, i.e. returned string will use suffixes of B, KB, MB, GB, TB for bytes, kilobytes, megabytes, gigabytes and terabytes respectively. With the IEC convention the names of the units are changed to B, KiB, MiB, GiB and TiB for bytes, kibibytes, mebibytes, gibibytes and tebibytes. Finally, with SI convention the same B, KB, MB, GB and TB suffixes are used but in their correct SI meaning, i.e. as multiples of 1000 and not 1024.

Support for the different size conventions is new in wxWidgets 2.9.1, in previous versions only the traditional convention was implemented.

static wxString wxFileName::GetHumanReadableSize (const wxULongLong & *bytes*, const wxString & *nullsize* = _("Not available"), int *precision* = 1, wxSizeConvention *conv* = wxSIZE_CONV_TRADITIONAL) [static]

Returns the representation of the file size in a human-readable form.

In the first version, the size of this file is used. In the second one, the specified size *bytes* is used.

If the file size could not be retrieved or *bytes* is wxInvalidSize or zero, the `failmsg` string is returned.

Otherwise the returned string is a floating-point number with `precision` decimal digits followed by the abbreviation of the unit used. By default the traditional, although incorrect, convention of using SI units for multiples of 1024 is used, i.e. returned string will use suffixes of B, KB, MB, GB, TB for bytes, kilobytes, megabytes, gigabytes and terabytes respectively. With the IEC convention the names of the units are changed to B, KiB, MiB, GiB and TiB for bytes, kibibytes, mebibytes, gibibytes and tebibytes. Finally, with SI convention the same B, KB, MB, GB and TB suffixes are used but in their correct SI meaning, i.e. as multiples of 1000 and not 1024.

Support for the different size conventions is new in wxWidgets 2.9.1, in previous versions only the traditional convention was implemented.

wxString wxFileName::GetLongPath ( ) const

Return the long form of the path (returns identity on non-Windows platforms).

wxDateTime wxFileName::GetModificationTime ( ) const

Returns the last time the file was last modified.

wxString wxFileName::GetName ( ) const

Returns the name part of the filename (without extension).

See also

GetFullName()

wxString wxFileName::GetPath (int *flags* = wxPATH_GET_VOLUME, wxPathFormat *format* = wxPATH_NATIVE) const

Returns the path part of the filename (without the name or extension).

The possible flags values are:

- **wxPATH_GET_VOLUME:** Return the path with the volume (does nothing for the filename formats without volumes), otherwise the path without volume part is returned.
- **wxPATH_GET_SEPARATOR:** Return the path with the trailing separator, if this flag is not given there will be no separator at the end of the path.
- **wxPATH_NO_SEPARATOR:** Don't include the trailing separator in the returned string. This is the default (the value of this flag is 0) and exists only for symmetry with wxPATH_GET_SEPARATOR.

Note

If the path is a toplevel one (e.g. `"/"` on Unix or `"C:\"` on Windows), then the returned path will contain trailing separator even with `wxPATH_NO_SEPARATOR`.

static wxUniChar wxFileName::GetPathSeparator (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the usually used path separator for this format.

For all formats but `wxPATH_DOS` there is only one path separator anyhow, but for DOS there are two of them and the native one, i.e. the backslash is returned by this method.

See also

GetPathSeparators()

static wxString wxFileName::GetPathSeparators (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the string containing all the path separators for this format.

For all formats but `wxPATH_DOS` this string contains only one character but for DOS and Windows both `'/'` and `'\'` may be used as separators.

See also

GetPathSeparator()

static wxString wxFileName::GetPathTerminators (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the string of characters which may terminate the path part.

This is the same as GetPathSeparators() except for VMS path format where] is used at the end of the path part.

wxString wxFileName::GetPathWithSep (wxPathFormat *format* = wxPATH_NATIVE) const

Returns the path with the trailing separator, useful for appending the name to the given path.

This is the same as calling

```
GetPath(wxPATH_GET_VOLUME | wxPATH_GET_SEPARATOR, format)
```

wxString wxFileName::GetShortPath ( ) const

Return the short form of the path (returns identity on non-Windows platforms).

wxULongLong wxFileName::GetSize ( ) const

Returns the size of the file If the file does not exist or its size could not be read (because e.g.

the file is locked by another process) the returned value is wxInvalidSize.

static wxULongLong wxFileName::GetSize (const wxString & *filename*) [static]

Returns the size of the file If the file does not exist or its size could not be read (because e.g.

the file is locked by another process) the returned value is wxInvalidSize.

static wxString wxFileName::GetTempDir ( ) [static]

Returns the directory used for temporary files.

bool wxFileName::GetTimes (wxDateTime * *dtAccess,* wxDateTime * *dtMod,* wxDateTime * *dtCreate*) const

Returns the last access, last modification and creation times.

The last access time is updated whenever the file is read or written (or executed in the case of Windows), last modification time is only changed when the file is written to. Finally, the creation time is indeed the time when the file was created under Windows and the inode change time under Unix (as it is impossible to retrieve the real file creation time there anyhow) which can also be changed by many operations after the file creation.

If no filename or extension is specified in this instance of wxFileName (and therefore IsDir() returns true) then this function will return the directory times of the path specified by GetPath(), otherwise the file times of the file specified by GetFullPath(). Any of the pointers may be NULL if the corresponding time is not needed.

Returns

true on success, false if we failed to retrieve the times.

wxString wxFileName::GetVolume ( ) const

Returns the string containing the volume for this file name, empty if it doesn't have one or if the file system doesn't support volumes at all (for example, Unix).

static wxString wxFileName::GetVolumeSeparator (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns the string separating the volume from the path for this format.

static wxString wxFileName::GetVolumeString (char *drive,* int *flags* = wxPATH_GET_SEPARATOR) [static]

This function builds a volume path string, for example "C:\\".

Implemented for the platforms which use drive letters, i.e. DOS, MSW and OS/2 only.

Since

2.9.0

Parameters

drive	The drive letter, 'A' through 'Z' or 'a' through 'z'.
flags	`wxPATH_NO_SEPARATOR` or `wxPATH_GET_SEPARATOR` to omit or include the trailing path separator, the default is to include it.

Returns

Volume path string.

bool wxFileName::HasExt ( ) const

Returns true if an extension is present.

bool wxFileName::HasName ( ) const

Returns true if a name is present.

bool wxFileName::HasVolume ( ) const

Returns true if a volume specifier is present.

bool wxFileName::InsertDir (size_t *before,* const wxString & *dir*)

Inserts a directory component before the zero-based position in the directory list.

As with AppendDir(), *dir* must be a single directory name and the function returns false and does nothing else if it isn't.

Notice that the return value is only available in wxWidgets 2.9.5 or later.

bool wxFileName::IsAbsolute (wxPathFormat *format* = wxPATH_NATIVE) const

Returns true if this filename is absolute.

static bool wxFileName::IsCaseSensitive (wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns true if the file names of this type are case-sensitive.

bool wxFileName::IsDir ( ) const

Returns true if this object represents a directory, false otherwise (i.e.

if it is a file).

Note that this method doesn't test whether the directory or file really exists, you should use DirExists() or FileExists() for this.

bool wxFileName::IsDirReadable ( ) const

Returns true if the directory component of this instance is an existing directory and this process has read permissions on it.

Read permissions on a directory mean that you can list the directory contents but it doesn't imply that you have read permissions on the files contained.

static bool wxFileName::IsDirReadable (const wxString & *dir*) [static]

Returns true if the given *dir* is an existing directory and this process has read permissions on it.

Read permissions on a directory mean that you can list the directory contents but it doesn't imply that you have read permissions on the files contained.

bool wxFileName::IsDirWritable ( ) const

Returns true if the directory component of this instance is an existing directory and this process has write permissions on it.

Write permissions on a directory mean that you can create new files in the directory.

static bool wxFileName::IsDirWritable (const wxString & *dir*) [static]

Returns true if the given *dir* is an existing directory and this process has write permissions on it.

Write permissions on a directory mean that you can create new files in the directory.

bool wxFileName::IsFileExecutable ( ) const

Returns true if a file with this name exists and if this process has execute permissions on it.

static bool wxFileName::IsFileExecutable (const wxString & *file*) [static]

Returns true if a file with this name exists and if this process has execute permissions on it.

bool wxFileName::IsFileReadable ( ) const

Returns true if a file with this name exists and if this process has read permissions on it.

static bool wxFileName::IsFileReadable (const wxString & *file*) [static]

Returns true if a file with this name exists and if this process has read permissions on it.

bool wxFileName::IsFileWritable ( ) const

Returns true if a file with this name exists and if this process has write permissions on it.

static bool wxFileName::IsFileWritable (const wxString & *file*) [static]

Returns true if a file with this name exists and if this process has write permissions on it.

static bool wxFileName::IsMSWUniqueVolumeNamePath (const wxString & *path,* wxPathFormat *format =* wxPATH_NATIVE) [static]

Returns true if the volume part of the path is a unique volume name.

This function will always return false if the path format is not wxPATH_DOS.

Unique volume names are Windows volume identifiers which remain the same regardless of where the volume is actually mounted. Example of a path using a volume name could be

```
\\?\Volume{8089d7d7-d0ac-11db-9dd0-806d6172696f}\Program Files\setup.exe
```

Since

2.9.1

bool wxFileName::IsOk ( ) const

Returns true if the filename is valid, false if it is not initialized yet.

The assignment functions and Clear() may reset the object to the uninitialized, invalid state (the former only do it on failure).

static bool wxFileName::IsPathSeparator (wxChar *ch,* wxPathFormat *format* = wxPATH_NATIVE) [static]

Returns true if the char is a path separator for this format.

bool wxFileName::IsRelative (wxPathFormat *format* = wxPATH_NATIVE) const

Returns true if this filename is not absolute.

static bool wxFileName::MacFindDefaultTypeAndCreator (const wxString & *ext,* wxUint32 * *type,* wxUint32 * *creator*) [static]

On Mac OS, gets the common type and creator for the given extension.

Availability: only available for the wxOSX port.

static void wxFileName::MacRegisterDefaultTypeAndCreator (const wxString & *ext,* wxUint32 *type,* wxUint32 *creator*) [static]

On Mac OS, registers application defined extensions and their default type and creator.

Availability: only available for the wxOSX port.

bool wxFileName::MacSetDefaultTypeAndCreator ( )

On Mac OS, looks up the appropriate type and creator from the registration and then sets it.

Availability: only available for the wxOSX port.

bool wxFileName::MakeAbsolute (const wxString & *cwd* = wxEmptyString, wxPathFormat *format* = wxPATH_NATIVE)

Make the file name absolute.

This is a shortcut for

```
wxFileName::Normalize(wxPATH_NORM_DOTS |
      wxPATH_NORM_ABSOLUTE |
                    wxPATH_NORM_TILDE, cwd, format)
```

See also

MakeRelativeTo(), Normalize(), IsAbsolute()

bool wxFileName::MakeRelativeTo (const wxString & *pathBase* = wxEmptyString, wxPathFormat *format* = wxPATH_NATIVE)

This function tries to put this file name in a form relative to *pathBase*.

In other words, it returns the file name which should be used to access this file if the current directory were pathBase.

Parameters

pathBase	The directory to use as root, current directory is used by default
format	The file name format, native by default

Returns

true if the file name has been changed, false if we failed to do anything with it (currently this only happens if the file name is on a volume different from the volume specified by *pathBase*).

See also

Normalize()

bool wxFileName::Mkdir (int *perm* = wxS_DIR_DEFAULT, int *flags* = 0) const

Creates a directory.

Parameters

perm	The permissions for the newly created directory. See the wxPosixPermissions enumeration for more info.
flags	If the flags contain `wxPATH_MKDIR_FULL` flag, try to create each directory in the path and also don't return an error if the target directory already exists.

Returns

Returns true if the directory was successfully created, false otherwise.

static bool wxFileName::Mkdir (const wxString & *dir*, int *perm* = wxS_DIR_DEFAULT, int *flags* = 0) [static]

Creates a directory.

Parameters

dir	The directory to create
perm	The permissions for the newly created directory. See the wxPosixPermissions enumeration for more info.
flags	If the flags contain `wxPATH_MKDIR_FULL` flag, try to create each directory in the path and also don't return an error if the target directory already exists.

Returns

Returns true if the directory was successfully created, false otherwise.

bool wxFileName::Normalize (int *flags* = wxPATH_NORM_ALL, const wxString & *cwd* = wxEmptyString, wxPathFormat *format* = wxPATH_NATIVE)

Normalize the path.

With the default flags value, the path will be made absolute, without any ".." and "." and all environment variables will be expanded in it.

Notice that in some rare cases normalizing a valid path may result in an invalid wxFileName object. E.g. normalizing "./" path using wxPATH_NORM_DOTS but not wxPATH_NORM_ABSOLUTE will result in a completely empty and thus invalid object. As long as there is a non empty file name the result of normalization will be valid however.

Parameters

flags	The kind of normalization to do with the file name. It can be any or-combination of the wxPathNormalize enumeration values.
cwd	If not empty, this directory will be used instead of current working directory in normalization (see `wxPATH_NORM_ABSOLUTE`).
format	The file name format to use when processing the paths, native by default.

Returns

true if normalization was successfully or false otherwise.

bool wxFileName::operator!= (const wxFileName & *filename*) const

Returns true if the filenames are different.

The string *filenames* is interpreted as a path in the native filename format.

bool wxFileName::operator!= (const wxString & *filename*) const

Returns true if the filenames are different.

The string *filenames* is interpreted as a path in the native filename format.

wxFileName& wxFileName::operator= (const wxFileName & *filename*)

Assigns the new value to this filename object.

wxFileName& wxFileName::operator= (const wxString & *filename*)

Assigns the new value to this filename object.

bool wxFileName::operator== (const wxFileName & *filename*) const

Returns true if the filenames are equal.

The string *filenames* is interpreted as a path in the native filename format.

bool wxFileName::operator== (const wxString & *filename*) const

Returns true if the filenames are equal.

The string *filenames* is interpreted as a path in the native filename format.

void wxFileName::PrependDir (const wxString & *dir*)

Prepends a directory to the file path.

Please see AppendDir() for important notes.

void wxFileName::RemoveDir (size_t *pos*)

Removes the specified directory component from the path.

See also

GetDirCount()

void wxFileName::RemoveLastDir ( )

Removes last directory component from the path.

bool wxFileName::ReplaceEnvVariable (const wxString & *envname*, const wxString & *replacementFmtString* = "$%s", wxPathFormat *format* = wxPATH_NATIVE)

If the path contains the value of the environment variable named *envname* then this function replaces it with the string obtained from wxString::Format(replacementFmtString, value_of_envname_variable).

This function is useful to make the path shorter or to make it dependent from a certain environment variable. Normalize() with `wxPATH_NORM_ENV_VARS` can perform the opposite of this function (depending on the value of *replacementFmtString*).

The name and extension of this filename are not modified.

Example:

```
wxFileName fn("/usr/openwin/lib/someFile");
fn.ReplaceEnvVariable("OPENWINHOME");
        // now fn.GetFullPath() == "$OPENWINHOME/lib/someFile"
```

Since

2.9.0

Returns

true if the operation was successful (which doesn't mean that something was actually replaced, just that wxGetEnv didn't fail).

bool wxFileName::ReplaceHomeDir (wxPathFormat *format* = wxPATH_NATIVE)

Replaces, if present in the path, the home directory for the given user (see wxGetHomeDir) with a tilde (~).

Normalize() with `wxPATH_NORM_TILDE` performs the opposite of this function.

The name and extension of this filename are not modified.

Since

2.9.0

Returns

true if the operation was successful (which doesn't mean that something was actually replaced, just that wxGetHomeDir didn't fail).

bool wxFileName::Rmdir (int *flags* = 0) const

Deletes the specified directory from the file system.

Parameters

flags	Can contain one of wxPATH_RMDIR_FULL or wxPATH_RMDIR_RECURSIVE. By default contains neither so the directory will not be removed unless it is empty.

Returns

Returns true if the directory was successfully deleted, false otherwise.

static bool wxFileName::Rmdir (const wxString & *dir,* int *flags* = 0) [static]

Deletes the specified directory from the file system.

Parameters

dir	The directory to delete
flags	Can contain one of wxPATH_RMDIR_FULL or wxPATH_RMDIR_RECURSIVE. By default contains neither so the directory will not be removed unless it is empty.

Returns

Returns true if the directory was successfully deleted, false otherwise.

bool wxFileName::SameAs (const wxFileName & *filepath,* wxPathFormat *format* = wxPATH_NATIVE) const

Compares the filename using the rules of this platform.

bool wxFileName::SetCwd ( ) const

Changes the current working directory.

static bool wxFileName::SetCwd (const wxString & *cwd*) [static]

Changes the current working directory.

void wxFileName::SetEmptyExt ( )

Sets the extension of the file name to be an empty extension.

This is different from having no extension at all as the file name will have a trailing dot after a call to this method.

See also

SetExt(), ClearExt()

void wxFileName::SetExt (const wxString & *ext*)

Sets the extension of the file name.

Setting an empty string as the extension will remove the extension resulting in a file name without a trailing dot, unlike a call to SetEmptyExt().

See also

SetEmptyExt(), ClearExt()

void wxFileName::SetFullName (const wxString & *fullname*)

The full name is the file name and extension (but without the path).

void wxFileName::SetName (const wxString & *name*)

Sets the name part (without extension).

See also

SetFullName()

void wxFileName::SetPath (const wxString & *path,* wxPathFormat *format* = wxPATH_NATIVE)

Sets the path.

The *path* argument includes both the path and the volume, if supported by *format*.

Calling this function doesn't affect the name and extension components, to change them as well you can use Assign() or just an assignment operator.

See also

GetPath()

bool wxFileName::SetPermissions (int *permissions*)

Sets permissions for this file or directory.

Parameters

permissions	The new permissions: this should be a combination of wxPosixPermissions enum elements.

Since

3.0

Note

If this is a symbolic link and it should not be followed this call will fail.

Returns

true on success, false if an error occurred (for example, the file doesn't exist).

bool wxFileName::SetTimes (const wxDateTime * *dtAccess,* const wxDateTime * *dtMod,* const wxDateTime * *dtCreate*) const

Sets the file creation and last access/modification times (any of the pointers may be NULL).

Notice that the file creation time can't be changed under Unix, so *dtCreate* is ignored there (but true is still returned). Under Windows all three times can be set.

void wxFileName::SetVolume (const wxString & *volume*)

Sets the volume specifier.

bool wxFileName::ShouldFollowLink ( ) const

Return whether some operations will follow symlink.

By default, file operations "follow symlink", i.e. operate on its target and not on the symlink itself. See DontFollowLink() for more information.

Since

2.9.5

static void wxFileName::SplitPath (const wxString & *fullpath*, wxString * *volume*, wxString * *path*, wxString * *name*, wxString * *ext*, bool * *hasExt* = NULL, wxPathFormat *format* = wxPATH_NATIVE) [static]

This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension.

Any of the output parameters (*volume*, *path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component. Also, *fullpath* may be empty on entry. On return, *path* contains the file path (without the trailing separator), *name* contains the file name and *ext* contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

Note that for a filename "foo." the extension is present, as indicated by the trailing dot, but empty. If you need to cope with such cases, you should use *hasExt* instead of relying on testing whether *ext* is empty or not.

static void wxFileName::SplitPath (const wxString & *fullpath*, wxString * *volume*, wxString * *path*, wxString * *name*, wxString * *ext*, wxPathFormat *format*) [static]

This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension.

Any of the output parameters (*volume*, *path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component. Also, *fullpath* may be empty on entry. On return, *path* contains the file path (without the trailing separator), *name* contains the file name and *ext* contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

Note that for a filename "foo." the extension is present, as indicated by the trailing dot, but empty. If you need to cope with such cases, you should use *hasExt* instead of relying on testing whether *ext* is empty or not.

static void wxFileName::SplitPath (const wxString & *fullpath*, wxString * *path*, wxString * *name*, wxString * *ext*, wxPathFormat *format* = wxPATH_NATIVE) [static]

This function splits a full file name into components: the volume (with the first version) path (including the volume in the second version), the base name and the extension.

Any of the output parameters (*volume*, *path*, *name* or *ext*) may be NULL if you are not interested in the value of a particular component. Also, *fullpath* may be empty on entry. On return, *path* contains the file path (without the trailing separator), *name* contains the file name and *ext* contains the file extension without leading dot. All three of them may be empty if the corresponding component is. The old contents of the strings pointed to by these parameters will be overwritten in any case (if the pointers are not NULL).

Note that for a filename "foo." the extension is present, as indicated by the trailing dot, but empty. If you need to cope with such cases, you should use *hasExt* instead of relying on testing whether *ext* is empty or not.

static void wxFileName::SplitVolume (const wxString & *fullpath,* wxString * *volume,* wxString * *path,* wxPathFormat *format* = wxPATH_NATIVE) [static]

Splits the given *fullpath* into the volume part (which may be empty) and the pure path part, not containing any volume.

See also

SplitPath()

static wxString wxFileName::StripExtension (const wxString & *fullname*) [static]

Strip the file extension.

This function does more than just removing everything after the last period from the string, for example it will return the string ".vimrc" unchanged because the part after the period is not an extension but the file name in this case. You can use wxString::BeforeLast() to really get just the part before the last period (but notice that that function returns empty string if period is not present at all unlike this function which returns the *fullname* unchanged in this case).

Parameters

fullname	File path including name and, optionally, extension.

Returns

File path without extension

Since

2.9.0

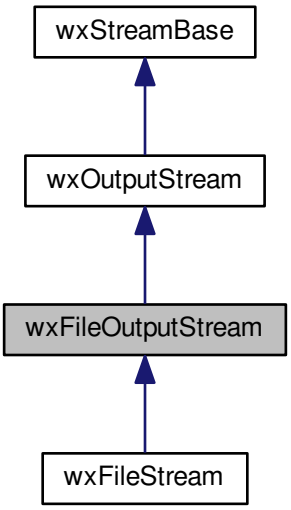
bool wxFileName::Touch ( ) const

Sets the access and modification times to the current moment.

21.240 wxFileOutputStream Class Reference

```
#include <wx/wfstream.h>
```

Inheritance diagram for wxFileOutputStream:



21.240.1 Detailed Description

This class represents data written to a file.

There are actually two such groups of classes: this one is based on wxFile whereas wxFFileOutputStream is based in the wxFFile class.

Note that wxOutputStream::SeekO() can seek beyond the end of the stream (file) and will thus not return wxInvalid↩ Offset for that.

Library: wxBase

Category: Streams

See also

wxBufferedOutputStream, wxFileInputStream, wxFFileOutputStream, wxFFileInputStream

Public Member Functions

- wxFileOutputStream (const wxString &ofileName)

  Creates a new file with ofileName name and initializes the stream in write-only mode.
- wxFileOutputStream (wxFile &file)

  Initializes a file stream in write-only mode using the file I/O object file.
- wxFileOutputStream (int fd)

  Initializes a file stream in write-only mode using the file descriptor fd.
- virtual ~wxFileOutputStream ()

  Destructor.

- bool IsOk () const

  Returns true if the stream is initialized and ready.
- wxFile * GetFile () const

  Returns the underlying file object.

Additional Inherited Members

21.240.2 Constructor & Destructor Documentation

wxFileOutputStream::wxFileOutputStream (const wxString & *ofileName*)

Creates a new file with *ofileName* name and initializes the stream in write-only mode.

Warning

You should use wxStreamBase::IsOk() to verify if the constructor succeeded.

wxFileOutputStream::wxFileOutputStream (wxFile & *file*)

Initializes a file stream in write-only mode using the file I/O object file.

wxFileOutputStream::wxFileOutputStream (int *fd*)

Initializes a file stream in write-only mode using the file descriptor *fd*.

virtual wxFileOutputStream::~wxFileOutputStream ( ) [virtual]

Destructor.

21.240.3 Member Function Documentation

wxFile* wxFileOutputStream::GetFile ( ) const

Returns the underlying file object.

Since

2.9.5

bool wxFileOutputStream::IsOk ( ) const [virtual]

Returns true if the stream is initialized and ready.

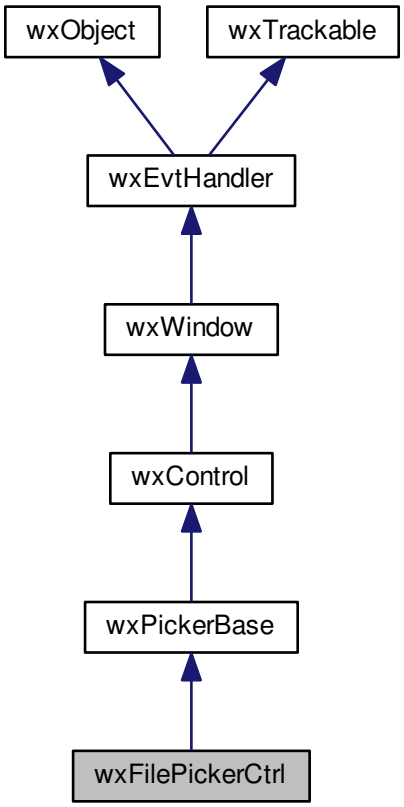
Reimplemented from wxStreamBase.

Reimplemented in wxFileStream.

21.241 wxFilePickerCtrl Class Reference

```
#include <wx/filepicker.h>
```

Inheritance diagram for wxFilePickerCtrl:



21.241.1 Detailed Description

This control allows the user to select a file.

The generic implementation is a button which brings up a wxFileDialog when clicked. Native implementation may differ but this is usually a (small) widget which give access to the file-chooser dialog. It is only available if `wxUSE_FILEPICKERCTRL` is set to 1 (the default).

Styles

This class supports the following styles:

- wxFLP_DEFAULT_STYLE: The default style: includes wxFLP_OPEN | wxFLP_FILE_MUST_EXIST and, under wxMSW only, wxFLP_USE_TEXTCTRL.
- wxFLP_USE_TEXTCTRL: Creates a text control to the left of the picker button which is completely managed by the wxFilePickerCtrl and which can be used by the user to specify a path (see SetPath). The text control is automatically synchronized with button's value. Use functions defined in wxPickerBase to modify the text control.
- wxFLP_OPEN: Creates a picker which allows the user to select a file to open.
- wxFLP_SAVE: Creates a picker which allows the user to select a file to save.

- wxFLP_OVERWRITE_PROMPT: Can be combined with wxFLP_SAVE only: ask confirmation to the user before selecting a file.
- wxFLP_FILE_MUST_EXIST: Can be combined with wxFLP_OPEN only: the selected file must be an existing file.
- wxFLP_CHANGE_DIR: Change current working directory on each user file selection change.
- wxFLP_SMALL: Use smaller version of the control with a small "..." button instead of the normal "Browse" one. This flag is new since wxWidgets 2.9.3.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFileDirPickerEvent& event)

Event macros for events emitted by this class:

- EVT_FILEPICKER_CHANGED(id, func): The user changed the file selected in the control either using the button or using text control (see wxFLP_USE_TEXTCTRL; note that in this case the event is fired only if the user's input is valid, e.g. an existing file path if wxFLP_FILE_MUST_EXIST was given).

Library: wxCore

Category: Picker Controls

See also

wxFileDialog, wxFileDirPickerEvent

Public Member Functions

- wxFilePickerCtrl ()
- wxFilePickerCtrl (wxWindow *parent, wxWindowID id, const wxString &path=wxEmptyString, const wxString &message=wxFileSelectorPromptStr, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxFLP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxFilePickerCtrlNameStr)

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &path=wxEmptyString, const wxString &message=wxFileSelectorPromptStr, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxFLP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxFilePickerCtrlNameStr)

  Creates this widget with the given parameters.
- wxFileName GetFileName () const

  Similar to GetPath() but returns the path of the currently selected file as a wxFileName object.
- wxString GetPath () const

  Returns the absolute path of the currently selected file.
- void SetFileName (const wxFileName &filename)

  This method does the same thing as SetPath() but takes a wxFileName object instead of a string.
- void SetInitialDirectory (const wxString &dir)

  Set the directory to show when starting to browse for files.
- void SetPath (const wxString &filename)

  Sets the absolute path of the currently selected file.

Additional Inherited Members

21.241.2 Constructor & Destructor Documentation

wxFilePickerCtrl::wxFilePickerCtrl ( )

wxFilePickerCtrl::wxFilePickerCtrl (wxWindow * *parent*, wxWindowID *id*, const wxString & *path* = wxEmptyString, const wxString & *message* = `wxFileSelectorPromptStr`, const wxString & *wildcard* = wxFileSelectorDefaultWildcardStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxFLP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxFilePickerCtrlNameStr`)

Initializes the object and calls Create() with all the parameters.

21.241.3 Member Function Documentation

bool wxFilePickerCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *path* = wxEmptyString, const wxString & *message* = `wxFileSelectorPromptStr`, const wxString & *wildcard* = wxFileSelectorDefaultWildcardStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxFLP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxFilePickerCtrlNameStr`)

Creates this widget with the given parameters.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
path	The initial file shown in the control. Must be a valid path to a file or the empty string.
message	The message shown to the user in the wxFileDialog shown by the control.
wildcard	A wildcard which defines user-selectable files (use the same syntax as for wxFileDialog's wildcards).
pos	Initial position.
size	Initial size.
style	The window style, see wxFLP_* flags.
validator	Validator which can be used for additional data checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

wxFileName wxFilePickerCtrl::GetFileName ( ) const

Similar to GetPath() but returns the path of the currently selected file as a wxFileName object.

wxString wxFilePickerCtrl::GetPath ( ) const

Returns the absolute path of the currently selected file.

void wxFilePickerCtrl::SetFileName (const wxFileName & *filename*)

This method does the same thing as SetPath() but takes a wxFileName object instead of a string.

void wxFilePickerCtrl::SetInitialDirectory (const wxString & *dir*)

Set the directory to show when starting to browse for files.

This function is mostly useful for the file picker controls which have no selection initially to configure the directory that should be shown if the user starts browsing for files as otherwise the directory of initially selected file is used, which is usually the desired behaviour and so the directory specified by this function is ignored in this case.

Since

2.9.4

void wxFilePickerCtrl::SetPath (const wxString & *filename*)

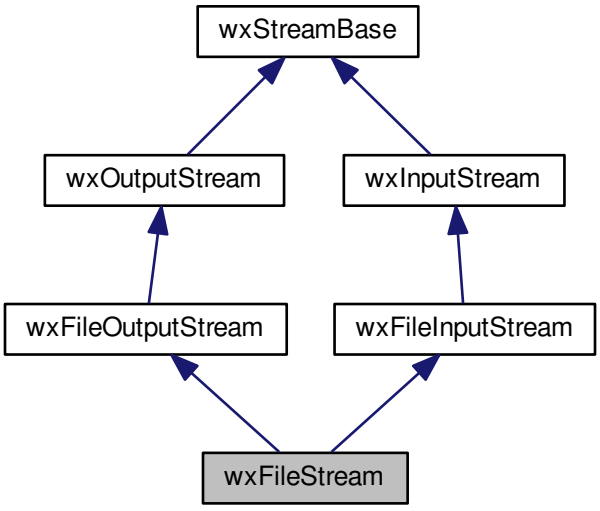
Sets the absolute path of the currently selected file.

This must be a valid file if the `wxFLP_FILE_MUST_EXIST` style was given.

21.242 wxFileStream Class Reference

`#include <wx/wfstream.h>`

Inheritance diagram for wxFileStream:



21.242.1 Detailed Description

This class represents data that can be both read from and written to a file.

There are actually two such groups of classes: this one is based on wxFile whereas wxFFileStream is based in the wxFFile class.

Library: wxBase

Category: Streams

See also

wxFileInputStream, wxFileOutputStream, wxFFileStream

Public Member Functions

- wxFileStream (const wxString &iofileName)

  Initializes a new file stream in read-write mode using the specified iofileName name.

- bool IsOk () const

  Returns true if the stream is initialized and ready.

Additional Inherited Members

21.242.2 Constructor & Destructor Documentation

wxFileStream::wxFileStream (const wxString & *iofileName*)

Initializes a new file stream in read-write mode using the specified *iofileName* name.

Warning

You should use IsOk() to verify if the constructor succeeded.

21.242.3 Member Function Documentation

bool wxFileStream::IsOk ( ) const `[virtual]`

Returns true if the stream is initialized and ready.

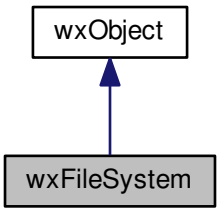
This method returns true if the stream can be both read from and written to.

Reimplemented from wxFileOutputStream.

21.243 wxFileSystem Class Reference

```
#include <wx/filesys.h>
```

Inheritance diagram for wxFileSystem:



21.243.1 Detailed Description

This class provides an interface for opening files on different file systems.

It can handle absolute and/or local filenames.

It uses a system of handlers (see wxFileSystemHandler) to provide access to user-defined virtual file systems.

Library: wxBase

Category: Virtual File System

See also

wxFileSystemHandler, wxFSFile, wxFileSystem Overview

Public Member Functions

- wxFileSystem ()

  Constructor.
- void ChangePathTo (const wxString &location, bool is_dir=false)

  Sets the current location.
- bool FindFileInPath (wxString *pStr, const wxString &path, const wxString &file)

  Looks for the file with the given name file in a colon or semi-colon (depending on the current platform) separated list of directories in path.
- wxString FindFirst (const wxString &wildcard, int flags=0)

  Works like wxFindFirstFile().
- wxString FindNext ()

  Returns the next filename that matches the parameters passed to FindFirst().
- wxString GetPath () const

  Returns the actual path (set by wxFileSystem::ChangePathTo).
- wxFSFile * OpenFile (const wxString &location, int flags=wxFS_READ)

  Opens the file and returns a pointer to a wxFSFile object or NULL if failed.

Static Public Member Functions

- static void AddHandler (wxFileSystemHandler *handler)

  This static function adds new handler into the list of handlers (see wxFileSystemHandler) which provide access to virtual FS.
- static wxFileSystemHandler * RemoveHandler (wxFileSystemHandler *handler)

  Remove a filesystem handler from the list of handlers.
- static wxString FileNameToURL (const wxFileName &filename)

  Converts a wxFileName into an URL.
- static bool HasHandlerForPath (const wxString &location)

  This static function returns true if there is a registered handler which can open the given location.
- static wxFileName URLToFileName (const wxString &url)

  Converts URL into a well-formed filename.

Additional Inherited Members

21.243.2 Constructor & Destructor Documentation

wxFileSystem::wxFileSystem ( )

Constructor.

The initial current path of this object will be empty (i.e. GetPath() == wxEmptyString) which means that e.g. OpenFile() or FindFirst() functions will use current working directory as current path (see also wxGetCwd).

21.243.3 Member Function Documentation

static void wxFileSystem::AddHandler (wxFileSystemHandler * *handler*) [static]

This static function adds new handler into the list of handlers (see wxFileSystemHandler) which provide access to virtual FS.

Note that if two handlers for the same protocol are added, the last added one takes precedence.

Note

You can call:

```
wxFileSystem::AddHandler(new My_FS_Handler);
```

This is because (a) AddHandler is a static method, and (b) the handlers are deleted in wxFileSystem's destructor so that you don't have to care about it.

void wxFileSystem::ChangePathTo (const wxString & *location*, bool *is_dir* = false)

Sets the current location.

location parameter passed to OpenFile() is relative to this path.

Remarks

Unless *is_dir* is true the *location* parameter is not the directory name but the name of the file in this directory.

All these commands change the path to "dir/subdir/":

```
ChangePathTo("dir/subdir/xh.htm");
ChangePathTo("dir/subdir", true);
ChangePathTo("dir/subdir/", true);
```

Example:

```
f = fs->OpenFile("hello.htm"); // opens file 'hello.htm'
fs->ChangePathTo("subdir/folder", true);
f = fs->OpenFile("hello.htm"); // opens file 'subdir/folder/hello.htm' !!
```

Parameters

location	the new location. Its meaning depends on the value of is_dir
is_dir	if true location is new directory. If false (the default) location is file in the new directory.

static wxString wxFileSystem::FileNameToURL (const wxFileName & *filename*) [static]

Converts a wxFileName into an URL.

See also

URLToFileName(), wxFileName

bool wxFileSystem::FindFileInPath (wxString * *pStr*, const wxString & *path*, const wxString & *file*)

Looks for the file with the given name *file* in a colon or semi-colon (depending on the current platform) separated list of directories in *path*.

If the file is found in any directory, returns true and the full path of the file in *str*, otherwise returns false and doesn't modify *str*.

Parameters

pStr	Receives the full path of the file, must not be NULL
path	wxPATH_SEP-separated list of directories
file	the name of the file to look for

wxString wxFileSystem::FindFirst (const wxString & *wildcard*, int *flags* = 0)

Works like wxFindFirstFile().

Returns the name of the first filename (within filesystem's current path) that matches *wildcard*.

Parameters

wildcard	The wildcard that the filename must match
flags	One of wxFILE (only files), wxDIR (only directories) or 0 (both).

wxString wxFileSystem::FindNext ( )

Returns the next filename that matches the parameters passed to FindFirst().

wxString wxFileSystem::GetPath ( ) const

Returns the actual path (set by wxFileSystem::ChangePathTo).

static bool wxFileSystem::HasHandlerForPath (const wxString & *location*) [static]

This static function returns true if there is a registered handler which can open the given location.

wxFSFile* wxFileSystem::OpenFile (const wxString & *location*, int *flags* = wxFS_READ)

Opens the file and returns a pointer to a wxFSFile object or NULL if failed.

It first tries to open the file in relative scope (based on value passed to ChangePathTo() method) and then as an absolute path.

Note that the user is responsible for deleting the returned wxFSFile. *flags* can be one or more of the wxFileSystem↩ OpenFlags values combined together.

A stream opened with just the default *wxFS_READ* flag may or may not be seekable depending on the underlying source.

Passing *"wxFS_READ | wxFS_SEEKABLE"* for *flags* will back a stream that is not natively seekable with memory or a file and return a stream that is always seekable.

Note

The *location* argument is, despite this method's name *not* a filename. It is a "location", aka wxFileSystem URL (see wxFileSystem Overview).

static wxFileSystemHandler* wxFileSystem::RemoveHandler (wxFileSystemHandler * *handler*) [static]

Remove a filesystem handler from the list of handlers.

static wxFileName wxFileSystem::URLToFileName (const wxString & *url*) [static]

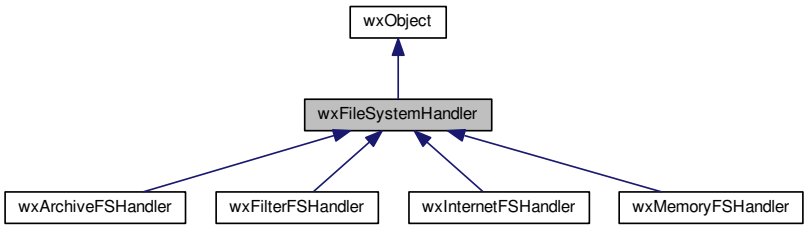
Converts URL into a well-formed filename.

The URL must use the `file` protocol.

21.244 wxFileSystemHandler Class Reference

`#include <wx/filesys.h>`

Inheritance diagram for wxFileSystemHandler:



21.244.1 Detailed Description

Classes derived from wxFileSystemHandler are used to access virtual file systems.

Its public interface consists of two methods: wxFileSystemHandler::CanOpen and wxFileSystemHandler::OpenFile.

It provides additional protected methods to simplify the process of opening the file: GetProtocol(), GetLeftLocation(), GetRightLocation(), GetAnchor(), GetMimeTypeFromExt().

Please have a look at overview (see wxFileSystem) if you don't know how locations are constructed.

Also consult the list of available handlers.

Note that the handlers are shared by all instances of wxFileSystem.

Remarks

wxHTML library provides handlers for local files and HTTP or FTP protocol.

Note

The location parameter passed to OpenFile() or CanOpen() methods is always an absolute path. You don't need to check the FS's current path.

wxPerl Note: In wxPerl, you need to derive your file system handler class from `Wx::PlFileSystemHandler`.

Library: wxBase

Category: Virtual File System

See also

wxFileSystem, wxFSFile, wxFileSystem Overview

Public Member Functions

- wxFileSystemHandler ()

  Constructor.
- virtual bool CanOpen (const wxString &location)=0

  Returns true if the handler is able to open this file.
- virtual wxString FindFirst (const wxString &wildcard, int flags=0)

  Works like wxFindFirstFile().
- virtual wxString FindNext ()

  Returns next filename that matches parameters passed to wxFileSystem::FindFirst.
- virtual wxFSFile * OpenFile (wxFileSystem &fs, const wxString &location)=0

  Opens the file and returns wxFSFile pointer or NULL if failed.

Static Public Member Functions

- static wxString GetMimeTypeFromExt (const wxString &location)

  *Returns the MIME type based on **extension** of location.*

Static Protected Member Functions

- static wxString GetAnchor (const wxString &location)

  Returns the anchor if present in the location.
- static wxString GetLeftLocation (const wxString &location)

  Returns the left location string extracted from location.
- static wxString GetProtocol (const wxString &location)

  Returns the protocol string extracted from location.
- static wxString GetRightLocation (const wxString &location)

  Returns the right location string extracted from location.

Additional Inherited Members

21.244.2 Constructor & Destructor Documentation

wxFileSystemHandler::wxFileSystemHandler ( )

Constructor.

21.244.3 Member Function Documentation

virtual bool wxFileSystemHandler::CanOpen (const wxString & *location*) `[pure virtual]`

Returns true if the handler is able to open this file.

This function doesn't check whether the file exists or not, it only checks if it knows the protocol. Example:

```
bool MyHand::CanOpen(const wxString& location)
{
    return (GetProtocol(location) == "http");
}
```

Must be overridden in derived handlers.

virtual wxString wxFileSystemHandler::FindFirst (const wxString & *wildcard*, int *flags* = 0) `[virtual]`

Works like wxFindFirstFile().

Returns the name of the first filename (within filesystem's current path) that matches *wildcard*. *flags* may be one of wxFILE (only files), wxDIR (only directories) or 0 (both).

This method is only called if CanOpen() returns true.

virtual wxString wxFileSystemHandler::FindNext ( ) `[virtual]`

Returns next filename that matches parameters passed to wxFileSystem::FindFirst.

This method is only called if CanOpen() returns true and FindFirst() returned a non-empty string.

static wxString wxFileSystemHandler::GetAnchor (const wxString & *location*) `[static], [protected]`

Returns the anchor if present in the location.

See wxFSFile::GetAnchor for details.

Example:

```
GetAnchor("index.htm#chapter2") == "chapter2"
```

Note

the anchor is NOT part of the left location.

static wxString wxFileSystemHandler::GetLeftLocation (const wxString & *location*) `[static], [protected]`

Returns the left location string extracted from *location*.

Example:

```
GetLeftLocation("file:myzipfile.zip#zip:index.htm") == "file:myzipfile.zip"
```

static wxString wxFileSystemHandler::GetMimeTypeFromExt (const wxString & *location*) `[static]`

Returns the MIME type based on **extension** of *location*.

(While wxFSFile::GetMimeType() returns real MIME type - either extension-based or queried from HTTP.)

Example:

```
GetMimeTypeFromExt("index.htm") == "text/html"
```

static wxString wxFileSystemHandler::GetProtocol (const wxString & *location*) `[static], [protected]`

Returns the protocol string extracted from *location*.

Example:

```
GetProtocol("file:myzipfile.zip#zip:index.htm") == "zip"
```

static wxString wxFileSystemHandler::GetRightLocation (const wxString & *location*) `[static], [protected]`

Returns the right location string extracted from *location*.

Example:

```
GetRightLocation("file:myzipfile.zip#zip:index.htm") == "index.htm"
```

virtual wxFSFile* wxFileSystemHandler::OpenFile (wxFileSystem & *fs*, const wxString & *location*) `[pure virtual]`

Opens the file and returns wxFSFile pointer or NULL if failed.

Must be overridden in derived handlers.

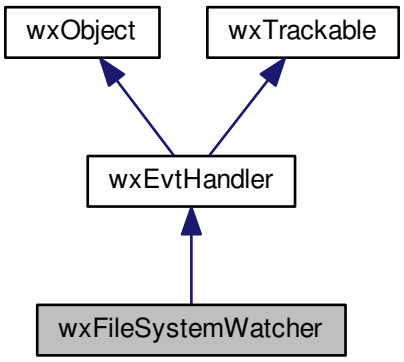
Parameters

fs	Parent FS (the FS from that OpenFile was called). See the ZIP handler for details of how to use it.
location	The absolute location of file.

21.245 wxFileSystemWatcher Class Reference

```
#include <wx/fswatcher.h>
```

Inheritance diagram for wxFileSystemWatcher:



21.245.1 Detailed Description

The wxFileSystemWatcher class allows to receive notifications of file system changes.

Note

Implementation limitations: this class is currently implemented for MSW, OS X and GTK ports but doesn't detect all changes correctly everywhere: under MSW accessing the file is not detected (only modifying it is) and under OS X neither accessing nor modifying is detected (only creating and deleting files is). Moreover, OS X version doesn't currently collapse pairs of create/delete events in a rename event, unlike the other ones.

For the full list of change types that are reported see wxFSWFlags.

This class notifies the application about the file system changes by sending events of wxFileSystemWatcherEvent class. By default these events are sent to the wxFileSystemWatcher object itself so you can derive from it and use the event table `EVT_FSWATCHER` macro to handle these events in a derived class method. Alternatively, you can use wxFileSystemWatcher::SetOwner() to send the events to another object. Or you could use wxEvtHandler::Connect() with `wxEVT_FSWATCHER` to handle these events in any other object. See the fswatcher sample for an example of the latter approach.

Library: wxBase

Category: File Handling

Since

2.9.1

Public Member Functions

- wxFileSystemWatcher ()

  Default constructor.
- virtual ~wxFileSystemWatcher ()

Destructor.

- virtual bool Add (const wxFileName &path, int events=wxFSW_EVENT_ALL)

  Adds path to currently watched files.

- virtual bool AddTree (const wxFileName &path, int events=wxFSW_EVENT_ALL, const wxString &filter=wxEmptyString)

  This is the same as Add(), but also recursively adds every file/directory in the tree rooted at path.

- virtual bool Remove (const wxFileName &path)

  Removes path from the list of watched paths.

- virtual bool RemoveTree (const wxFileName &path)

  This is the same as Remove(), but also removes every file/directory belonging to the tree rooted at path.

- virtual bool RemoveAll ()

  Clears the list of currently watched paths.

- int GetWatchedPathsCount () const

  Returns the number of currently watched paths.

- int GetWatchedPaths (wxArrayString *paths) const

  Retrieves all watched paths and places them in paths.

- void SetOwner (wxEvtHandler *handler)

  Associates the file system watcher with the given handler object.

Additional Inherited Members

21.245.2 Constructor & Destructor Documentation

wxFileSystemWatcher::wxFileSystemWatcher ( )

Default constructor.

virtual wxFileSystemWatcher::~wxFileSystemWatcher ( ) [virtual]

Destructor.

Stops all paths from being watched and frees any system resources used by this file system watcher object.

21.245.3 Member Function Documentation

virtual bool wxFileSystemWatcher::Add (const wxFileName & *path*, int *events* = wxFSW_EVENT_ALL) [virtual]

Adds *path* to currently watched files.

The *path* argument can currently only be a directory and any changes to this directory itself or its immediate children will generate the events. Use AddTree() to monitor the directory recursively.

Note that on platforms that use symbolic links, you should consider the possibility that *path* is a symlink. To watch the symlink itself and not its target you may call wxFileName::DontFollowLink() on *path*.

Parameters

path	The name of the path to watch.
events	An optional filter to receive only events of particular types. This is currently implemented only for GTK.

virtual bool wxFileSystemWatcher::AddTree (const wxFileName & *path*, int *events* = wxFSW_EVENT_ALL, const wxString & *filter* = wxEmptyString) [virtual]

This is the same as Add(), but also recursively adds every file/directory in the tree rooted at *path*.

Additionally a file mask can be specified to include only files matching that particular mask.

This method is implemented efficiently on MSW, but should be used with care on other platforms for directories with lots of children (e.g. the root directory) as it calls Add() for each subdirectory, potentially creating a lot of watches and taking a long time to execute.

Note that on platforms that use symbolic links, you will probably want to have called wxFileName::DontFollowLink on *path*. This is especially important if the symlink targets may themselves be watched.

int wxFileSystemWatcher::GetWatchedPaths (wxArrayString * *paths*) const

Retrieves all watched paths and places them in *paths*.

Returns the number of watched paths, which is also the number of entries added to *paths*.

int wxFileSystemWatcher::GetWatchedPathsCount ( ) const

Returns the number of currently watched paths.

See also

GetWatchedPaths()

virtual bool wxFileSystemWatcher::Remove (const wxFileName & *path*) [virtual]

Removes *path* from the list of watched paths.

See the comment in Add() about symbolic links. *path* should treat symbolic links in the same way as in the original Add() call.

virtual bool wxFileSystemWatcher::RemoveAll ( ) [virtual]

Clears the list of currently watched paths.

virtual bool wxFileSystemWatcher::RemoveTree (const wxFileName & *path*) [virtual]

This is the same as Remove(), but also removes every file/directory belonging to the tree rooted at *path*.

See the comment in AddTree() about symbolic links. *path* should treat symbolic links in the same way as in the original AddTree() call.

void wxFileSystemWatcher::SetOwner (wxEvtHandler * *handler*)

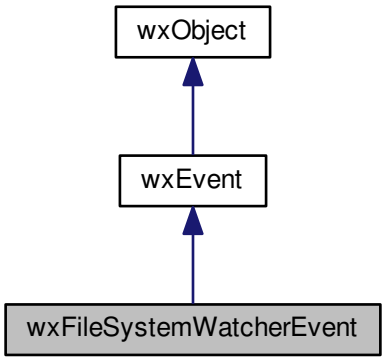
Associates the file system watcher with the given *handler* object.

All the events generated by this object will be passed to the specified owner.

21.246 wxFileSystemWatcherEvent Class Reference

`#include <wx/fswatcher.h>`

Inheritance diagram for wxFileSystemWatcherEvent:



21.246.1 Detailed Description

A class of events sent when a file system event occurs.

Types of events reported may vary depending on a platform, however all platforms report at least creation of new file/directory and access, modification, move (rename) or deletion of an existing one.

Library: wxBase

Category: Events

See also

wxFileSystemWatcher
Events and Event Handling

Since

2.9.1

Public Member Functions

- wxFileSystemWatcherEvent (int changeType=0, int watchid=wxID_ANY)
- wxFileSystemWatcherEvent (int changeType, wxFSWWarningType warningType, const wxString &errorMsg, int watchid=wxID_ANY)
- wxFileSystemWatcherEvent (int changeType, const wxFileName &path, const wxFileName &newPath, int watchid=wxID_ANY)
- const wxFileName & GetPath () const

  Returns the path at which the event occurred.
- const wxFileName & GetNewPath () const

  Returns the new path of the renamed file/directory if this is a rename event.
- int GetChangeType () const

Returns the type of file system change that occurred.

- bool IsError () const

  Returns `true` *if this error is an error event.*

- wxString GetErrorDescription () const

  Return a description of the warning or error if this is an error event.

- wxFSWWarningType GetWarningType () const

  Return the type of the warning if this event is a warning one.

- wxString ToString () const

  Returns a wxString describing an event, useful for logging, debugging or testing.

Additional Inherited Members

21.246.2 Constructor & Destructor Documentation

wxFileSystemWatcherEvent::wxFileSystemWatcherEvent (int *changeType* = `0`, int *watchid* = wxID_ANY)

wxFileSystemWatcherEvent::wxFileSystemWatcherEvent (int *changeType*, wxFSWWarningType *warningType*, const wxString & *errorMsg*, int *watchid* = wxID_ANY)

wxFileSystemWatcherEvent::wxFileSystemWatcherEvent (int *changeType*, const wxFileName & *path*, const wxFileName & *newPath*, int *watchid* = wxID_ANY)

21.246.3 Member Function Documentation

int wxFileSystemWatcherEvent::GetChangeType ( ) const

Returns the type of file system change that occurred.

See wxFSWFlags for the list of possible file system change types.

wxString wxFileSystemWatcherEvent::GetErrorDescription ( ) const

Return a description of the warning or error if this is an error event.

This string may be empty if the exact reason for the error or the warning is not known.

const wxFileName& wxFileSystemWatcherEvent::GetNewPath ( ) const

Returns the new path of the renamed file/directory if this is a rename event.

Otherwise it returns the same path as GetPath().

const wxFileName& wxFileSystemWatcherEvent::GetPath ( ) const

Returns the path at which the event occurred.

wxFSWWarningType wxFileSystemWatcherEvent::GetWarningType ( ) const

Return the type of the warning if this event is a warning one.

If this is not a warning event, i.e. if GetChangeType() doesn't include wxFSW_EVENT_WARNING, returns wxFS↩W_WARNING_NONE.

Since

3.0

bool wxFileSystemWatcherEvent::IsError ( ) const

Returns `true` if this error is an error event.

Error event is an event generated when a warning or error condition arises.

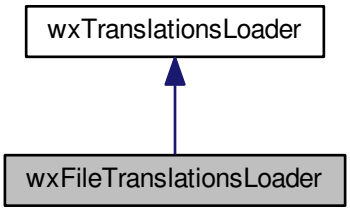
wxString wxFileSystemWatcherEvent::ToString ( ) const

Returns a wxString describing an event, useful for logging, debugging or testing.

21.247 wxFileTranslationsLoader Class Reference

`#include <wx/translation.h>`

Inheritance diagram for wxFileTranslationsLoader:



21.247.1 Detailed Description

Standard wxTranslationsLoader implementation.

This finds catalogs in the filesystem, using the standard Unix layout. This is the default unless you change the loader with wxTranslations::SetLoader().

Catalogs are searched for in standard places (system locales directory, `LC_PATH` on Unix systems, Resources subdirectory of the application bundle on OS X, executable's directory on Windows), but you may also prepend additional directories to the search path with AddCatalogLookupPathPrefix().

Since

2.9.1

Static Public Member Functions

- static void AddCatalogLookupPathPrefix (const wxString &prefix)

  Add a prefix to the catalog lookup path: the message catalog files will be looked up under prefix/lang/LC_MESSAGES and prefix/lang directories (in this order).

Additional Inherited Members

21.247.2 Member Function Documentation

static void wxFileTranslationsLoader::AddCatalogLookupPathPrefix (const wxString & *prefix*) `[static]`

Add a prefix to the catalog lookup path: the message catalog files will be looked up under prefix/lang/LC_MESSAGES and prefix/lang directories (in this order).

This only applies to subsequent invocations of wxTranslations::AddCatalog().

21.248 wxFileType Class Reference

`#include <wx/mimetype.h>`

21.248.1 Detailed Description

This class holds information about a given *file* type.

File type is the same as MIME type under Unix, but under Windows it corresponds more to an extension than to MIME type (in fact, several extensions may correspond to a file type).

This object may be created in several different ways: the program might know the file extension and wish to find out the corresponding MIME type or, conversely, it might want to find the right extension for the file to which it writes the contents of given MIME type. Depending on how it was created some fields may be unknown so the return value of all the accessors **must** be checked: false will be returned if the corresponding information couldn't be found.

The objects of this class are never created by the application code but are returned by wxMimeTypesManager::GetFileTypeFromMimeType and wxMimeTypesManager::GetFileTypeFromExtension methods. But it is your responsibility to delete the returned pointer when you're done with it!

A brief reminder about what the MIME types are (see the RFC 1341 for more information): basically, it is just a pair category/type (for example, "text/plain") where the category is a basic indication of what a file is. Examples of categories are "application", "image", "text", "binary", and type is a precise definition of the document format: "plain" in the example above means just ASCII text without any formatting, while "text/html" is the HTML document source.

A MIME type may have one or more associated extensions: "text/plain" will typically correspond to the extension ".txt", but may as well be associated with ".ini" or ".conf".

21.248.2 MessageParameters class

One of the most common usages of MIME is to encode an e-mail message. The MIME type of the encoded message is an example of a message parameter. These parameters are found in the message headers ("Content-XXX").

At the very least, they must specify the MIME type and the version of MIME used, but almost always they provide additional information about the message such as the original file name or the charset (for the text documents). These parameters may be useful to the program used to open, edit, view or print the message, so, for example, an e-mail client program will have to pass them to this program. Because wxFileType itself cannot know about these parameters, it uses MessageParameters class to query them.

The default implementation only requires the caller to provide the file name (always used by the program to be called - it must know which file to open) and the MIME type and supposes that there are no other parameters.

If you wish to supply additional parameters, you must derive your own class from MessageParameters and override GetParamValue() function, for example:

```
// provide the message parameters for the MIME type manager
class MailMessageParameters : public wxFileType::MessageParameters
{
public:
```

```
    MailMessageParameters(const wxString& filename,
                          const wxString& mimetype)
        : wxFileType::MessageParameters(filename, mimetype)
    {
    }

    virtual wxString GetParamValue(const wxString& name) const
    {
        // parameter names are not case-sensitive
        if ( name.CmpNoCase("charset") == 0 )
            return "US-ASCII";
        else
            return wxFileType::MessageParameters::GetParamValue
      (name);
    }
};
```

Now you only need to create an object of this class and pass it to, for example, GetOpenCommand like this:

```
wxString command;
if ( filetype->GetOpenCommand(&command,
                              MailMessageParameters("foo.txt", "text/plain")) )
{
    // the full command for opening the text documents is in 'command'
    // (it might be "notepad foo.txt" under Windows or "cat foo.txt" under Unix)
}
else
{
    // we don't know how to handle such files...
}
```

Windows: As only the file name is used by the program associated with the given extension anyhow (but no other message parameters), there is no need to ever derive from MessageParameters class for a Windows-only program.

Library: wxBase

Category: Data Structures

See also

wxMimeTypesManager

Classes

- class MessageParameters

  Class representing message parameters.

Public Member Functions

- wxFileType (const wxFileTypeInfo &ftInfo)

  Copy ctor.
- ~wxFileType ()

  The destructor of this class is not virtual, so it should not be derived from.
- bool GetDescription (wxString *desc) const

  If the function returns true, the string pointed to by desc is filled with a brief description for this file type: for example, "text document" for the "text/plain" MIME type.
- bool GetExtensions (wxArrayString &extensions)

  If the function returns true, the array extensions is filled with all extensions associated with this file type: for example, it may contain the following two elements for the MIME type "text/html" (notice the absence of the leading dot): "html" and "htm".
- bool GetIcon (wxIconLocation *iconLoc) const

If the function returns true, the `iconLoc` is filled with the location of the icon for this MIME type.

- bool GetMimeType (wxString *mimeType) const

  If the function returns true, the string pointed to by mimeType is filled with full MIME type specification for this file type: for example, "text/plain".

- bool GetMimeTypes (wxArrayString &mimeTypes) const

  Same as GetMimeType() but returns array of MIME types.

- bool GetPrintCommand (wxString *command, const MessageParameters &params) const

  If the function returns true, the string pointed to by command is filled with the command which must be executed (see wxExecute()) in order to print the file of the given type.

- size_t GetAllCommands (wxArrayString *verbs, wxArrayString *commands, const wxFileType::MessageParameters &params) const

  Returns the number of commands for this mime type, and fills the verbs and commands arrays with the command information.

- bool GetOpenCommand (wxString *command, const MessageParameters &params)

  With the first version of this method, if the true is returned, the string pointed to by command is filled with the command which must be executed (see wxExecute()) in order to open the file of the given type.

- wxString GetOpenCommand (const wxString &filename) const

  With the first version of this method, if the true is returned, the string pointed to by command is filled with the command which must be executed (see wxExecute()) in order to open the file of the given type.

Static Public Member Functions

- static wxString ExpandCommand (const wxString &command, const MessageParameters &params)

  This function is primarily intended for GetOpenCommand and GetPrintCommand usage but may be also used by the application directly if, for example, you want to use some non-default command to open the file.

Private Member Functions

- wxFileType ()

  The default constructor is private because you should never create objects of this type: they are only returned by wxMimeTypesManager methods.

21.248.3 Constructor & Destructor Documentation

wxFileType::wxFileType ( ) [private]

The default constructor is private because you should never create objects of this type: they are only returned by wxMimeTypesManager methods.

wxFileType::wxFileType (const wxFileTypeInfo & *ftInfo*)

Copy ctor.

wxFileType::~wxFileType ( )

The destructor of this class is not virtual, so it should not be derived from.

21.248.4 Member Function Documentation

static wxString wxFileType::ExpandCommand (const wxString & *command,* const MessageParameters & *params*) [static]

This function is primarily intended for GetOpenCommand and GetPrintCommand usage but may be also used by the application directly if, for example, you want to use some non-default command to open the file.

The function replaces all occurrences of:

- s with the full file name
- t with the MIME type
- %{param} with the value of the parameter *param* using the MessageParameters object you pass to it.

If there is no 's' in the command string (and the string is not empty), it is assumed that the command reads the data on stdin and so the effect is the same as " %s" were appended to the string.

Unlike all other functions of this class, there is no error return for this function.

size_t wxFileType::GetAllCommands (wxArrayString * *verbs,* wxArrayString * *commands,* const wxFileType::MessageParameters & *params*) const

Returns the number of commands for this mime type, and fills the verbs and commands arrays with the command information.

bool wxFileType::GetDescription (wxString * *desc*) const

If the function returns true, the string pointed to by *desc* is filled with a brief description for this file type: for example, "text document" for the "text/plain" MIME type.

bool wxFileType::GetExtensions (wxArrayString & *extensions*)

If the function returns true, the array *extensions* is filled with all extensions associated with this file type: for example, it may contain the following two elements for the MIME type "text/html" (notice the absence of the leading dot): "html" and "htm".

Windows: This function is currently not implemented: there is no (efficient) way to retrieve associated extensions from the given MIME type on this platform, so it will only return true if the wxFileType object was created by wxMimeTypesManager::GetFileTypeFromExtension function in the first place.

bool wxFileType::GetIcon (wxIconLocation * *iconLoc*) const

If the function returns true, the `iconLoc` is filled with the location of the icon for this MIME type.

A wxIcon may be created from *iconLoc* later.

Note

Under Unix MIME manager gathers information about icons from GNOME and KDE settings and thus GetIcon's success depends on availability of these desktop environments.

bool wxFileType::GetMimeType (wxString * *mimeType*) const

If the function returns true, the string pointed to by *mimeType* is filled with full MIME type specification for this file type: for example, "text/plain".

bool wxFileType::GetMimeTypes (wxArrayString & *mimeTypes*) const

Same as GetMimeType() but returns array of MIME types.

This array will contain only one item in most cases but sometimes, notably under Unix with KDE, may contain more MIME types. This happens when one file extension is mapped to different MIME types by KDE, mailcap and mime.types.

bool wxFileType::GetOpenCommand (wxString * *command,* const MessageParameters & *params*)

With the first version of this method, if the true is returned, the string pointed to by *command* is filled with the command which must be executed (see wxExecute()) in order to open the file of the given type.

In this case, the name of the file as well as any other parameters is retrieved from MessageParameters() class.

In the second case, only the filename is specified and the command to be used to open this kind of file is returned directly. An empty string is returned to indicate that an error occurred (typically meaning that there is no standard way to open this kind of files).

wxString wxFileType::GetOpenCommand (const wxString & *filename*) const

With the first version of this method, if the true is returned, the string pointed to by *command* is filled with the command which must be executed (see wxExecute()) in order to open the file of the given type.

In this case, the name of the file as well as any other parameters is retrieved from MessageParameters() class.

In the second case, only the filename is specified and the command to be used to open this kind of file is returned directly. An empty string is returned to indicate that an error occurred (typically meaning that there is no standard way to open this kind of files).

bool wxFileType::GetPrintCommand (wxString * *command,* const MessageParameters & *params*) const

If the function returns true, the string pointed to by *command* is filled with the command which must be executed (see wxExecute()) in order to print the file of the given type.

The name of the file is retrieved from the MessageParameters class.

21.249 wxFileTypeInfo Class Reference

`#include <wx/mimetype.h>`

21.249.1 Detailed Description

Container of information about wxFileType.

This class simply stores information associated with the file type. It doesn't do anything on its own and is used only to allow constructing wxFileType from it (instead of specifying all the constituent pieces separately) and also with wxMimeTypesManager::AddFallbacks().

Public Member Functions

- wxFileTypeInfo ()

  Default constructor creates an invalid file type info object.
- wxFileTypeInfo (const wxString &mimeType)

  Constructor specifying just the MIME type name.

- wxFileTypeInfo (const wxString &mimeType, const wxString &openCmd, const wxString &printCmd, const wxString &description, const wxString &extension,...)

  Constructor allowing to specify all the fields at once.
- wxFileTypeInfo (const wxArrayString &sArray)

  Constuctor using an array of string elements corresponding to the parameters of the ctor above in the same order.
- void AddExtension (const wxString &ext)

  Add another extension associated with this file type.
- void SetDescription (const wxString &description)

  Set the file type description.
- void SetOpenCommand (const wxString &command)

  Set the command to be used for opening files of this type.
- void SetPrintCommand (const wxString &command)

  Set the command to be used for printing files of this type.
- void SetShortDesc (const wxString &shortDesc)

  Set the short description for the files of this type.
- void SetIcon (const wxString &iconFile, int iconIndex=0)

  Set the icon information.
- const wxString & GetMimeType () const

  Get the MIME type.
- const wxString & GetOpenCommand () const

  Get the open command.
- const wxString & GetPrintCommand () const

  Get the print command.
- const wxString & GetShortDesc () const

  Get the short description (only used under Win32 so far)
- const wxString & GetDescription () const

  Get the long, user visible description.
- const wxArrayString & GetExtensions () const

  Get the array of all extensions.
- size_t GetExtensionsCount () const

  Get the number of extensions.
- const wxString & GetIconFile () const

  Get the icon filename.
- int GetIconIndex () const

  Get the index of the icon within the icon file.

21.249.2 Constructor & Destructor Documentation

wxFileTypeInfo::wxFileTypeInfo ( )

Default constructor creates an invalid file type info object.

Such invalid/empty object should be used to terminate the list of file types passed to wxMimeTypesManager::AddFallbacks().

wxFileTypeInfo::wxFileTypeInfo (const wxString & *mimeType*)

Constructor specifying just the MIME type name.

Use the various setter methods below to fully initialize the object.

Since

2.9.2

wxFileTypeInfo::wxFileTypeInfo (const wxString & *mimeType,* const wxString & *openCmd,* const wxString & *printCmd,* const wxString & *description,* const wxString & *extension,* ...)

Constructor allowing to specify all the fields at once.

This is a vararg constructor taking an arbitrary number of extensions after the first four required parameters. The list must be terminated by `wxNullPtr`, notice that `NULL` can't be used here in portable code (C++0x `nullptr` can be used as well if your compiler supports it).

wxFileTypeInfo::wxFileTypeInfo (const wxArrayString & *sArray*)

Constuctor using an array of string elements corresponding to the parameters of the ctor above in the same order.

21.249.3 Member Function Documentation

void wxFileTypeInfo::AddExtension (const wxString & *ext*)

Add another extension associated with this file type.

Since

2.9.2

const wxString& wxFileTypeInfo::GetDescription ( ) const

Get the long, user visible description.

const wxArrayString& wxFileTypeInfo::GetExtensions ( ) const

Get the array of all extensions.

size_t wxFileTypeInfo::GetExtensionsCount ( ) const

Get the number of extensions.

const wxString& wxFileTypeInfo::GetIconFile ( ) const

Get the icon filename.

int wxFileTypeInfo::GetIconIndex ( ) const

Get the index of the icon within the icon file.

const wxString& wxFileTypeInfo::GetMimeType ( ) const

Get the MIME type.

const wxString& wxFileTypeInfo::GetOpenCommand ( ) const

Get the open command.

const wxString& wxFileTypeInfo::GetPrintCommand ( ) const

Get the print command.

const wxString& wxFileTypeInfo::GetShortDesc ( ) const

Get the short description (only used under Win32 so far)

void wxFileTypeInfo::SetDescription (const wxString & *description*)

Set the file type description.

Since

2.9.2

void wxFileTypeInfo::SetIcon (const wxString & *iconFile,* int *iconIndex* = 0)

Set the icon information.

void wxFileTypeInfo::SetOpenCommand (const wxString & *command*)

Set the command to be used for opening files of this type.

Since

2.9.2

void wxFileTypeInfo::SetPrintCommand (const wxString & *command*)

Set the command to be used for printing files of this type.

Since

2.9.2

void wxFileTypeInfo::SetShortDesc (const wxString & *shortDesc*)

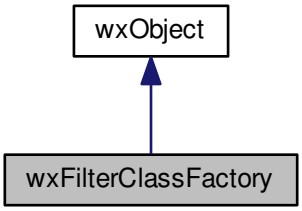
Set the short description for the files of this type.

This is only used under MSW for some of the registry keys used for the file type registration.

21.250 wxFilterClassFactory Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxFilterClassFactory:



21.250.1 Detailed Description

Allows the creation of filter streams to handle compression formats such as gzip and bzip2.

For example, given a filename you can search for a factory that will handle it and create a stream to decompress it:

```
factory = wxFilterClassFactory::Find(filename,
      wxSTREAM_FILEEXT);
if (factory)
    stream = factory->NewStream(new wxFFileInputStream(filename));
```

wxFilterClassFactory::Find can also search for a factory by MIME type, HTTP encoding or by wxFileSystem protocol. The available factories can be enumerated using wxFilterClassFactory::GetFirst() and wxFilterClassFactory::GetNext().

Library: wxBase

Category: Streams

See also

wxFilterInputStream, wxFilterOutputStream, wxArchiveClassFactory, Archive Formats

- const wxFilterClassFactory * GetNext () const

  GetFirst and GetNext can be used to enumerate the available factories.
- static const wxFilterClassFactory * GetFirst ()

  GetFirst and GetNext can be used to enumerate the available factories.

Public Member Functions

- bool CanHandle (const wxString &protocol, wxStreamProtocolType type=wxSTREAM_PROTOCOL) const

  Returns true if this factory can handle the given protocol, MIME type, HTTP encoding or file extension.
- wxString GetProtocol () const

  Returns the wxFileSystem protocol supported by this factory.
- virtual const wxChar *const * GetProtocols (wxStreamProtocolType type=wxSTREAM_PROTOCOL) const =0

  Returns the protocols, MIME types, HTTP encodings or file extensions supported by this factory, as an array of null terminated strings.

- wxString PopExtension (const wxString &location) const

  Remove the file extension of location if it is one of the file extensions handled by this factory.
- void PushFront ()

  Adds this class factory to the list returned by GetFirst()/GetNext().
- void Remove ()

  Removes this class factory from the list returned by GetFirst()/GetNext().

- virtual wxFilterInputStream * NewStream (wxInputStream &stream) const =0

  Create a new input or output stream to decompress or compress a given stream.
- virtual wxFilterOutputStream * NewStream (wxOutputStream &stream) const =0

  Create a new input or output stream to decompress or compress a given stream.
- virtual wxFilterInputStream * NewStream (wxInputStream *stream) const =0

  Create a new input or output stream to decompress or compress a given stream.
- virtual wxFilterOutputStream * NewStream (wxOutputStream *stream) const =0

  Create a new input or output stream to decompress or compress a given stream.

Static Public Member Functions

- static const wxFilterClassFactory * Find (const wxString &protocol, wxStreamProtocolType type=wxSTREAM_PROTOCOL)

  A static member that finds a factory that can handle a given protocol, MIME type, HTTP encoding or file extension.

Additional Inherited Members

21.250.2 Member Function Documentation

bool wxFilterClassFactory::CanHandle (const wxString & *protocol,* wxStreamProtocolType *type* = wxSTREAM_PROTOCOL) const

Returns true if this factory can handle the given protocol, MIME type, HTTP encoding or file extension.

When using `wxSTREAM_FILEEXT` for the second parameter, the first parameter can be a complete filename rather than just an extension.

static const wxFilterClassFactory* wxFilterClassFactory::Find (const wxString & *protocol,* wxStreamProtocolType *type* = wxSTREAM_PROTOCOL) [static]

A static member that finds a factory that can handle a given protocol, MIME type, HTTP encoding or file extension.

Returns a pointer to the class factory if found, or NULL otherwise. It does not give away ownership of the factory.

When using `wxSTREAM_FILEEXT` for the second parameter, the first parameter can be a complete filename rather than just an extension.

static const wxFilterClassFactory* wxFilterClassFactory::GetFirst ( ) [static]

GetFirst and GetNext can be used to enumerate the available factories.

For example, to list them:

```
wxString list;
const wxFilterClassFactory *factory =
      wxFilterClassFactory::GetFirst();

while (factory) {
    list << factory->GetProtocol() << wxT("\n");
    factory = factory->GetNext();
}
```

GetFirst()/GetNext() return a pointer to a factory or NULL if no more are available. They do not give away ownership of the factory.

const wxFilterClassFactory* wxFilterClassFactory::GetNext ( ) const

GetFirst and GetNext can be used to enumerate the available factories.

For example, to list them:

```
wxString list;
const wxFilterClassFactory *factory =
      wxFilterClassFactory::GetFirst();

while (factory) {
    list << factory->GetProtocol() << wxT("\n");
    factory = factory->GetNext();
}
```

GetFirst()/GetNext() return a pointer to a factory or NULL if no more are available. They do not give away ownership of the factory.

wxString wxFilterClassFactory::GetProtocol ( ) const

Returns the wxFileSystem protocol supported by this factory.

Equivalent to

```
wxString(*GetProtocols())
```

.

virtual const wxChar* const* wxFilterClassFactory::GetProtocols (wxStreamProtocolType *type* = wxSTREAM_PROTOCOL) const [pure virtual]

Returns the protocols, MIME types, HTTP encodings or file extensions supported by this factory, as an array of null terminated strings.

It does not give away ownership of the array or strings.

For example, to list the file extensions a factory supports:

```
wxString list;
const wxChar *const *p;

for (p = factory->GetProtocols(wxSTREAM_FILEEXT); *p; p++)
    list << *p << wxT("\n");
```

virtual wxFilterInputStream* wxFilterClassFactory::NewStream (wxInputStream & *stream*) const [pure virtual]

Create a new input or output stream to decompress or compress a given stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

virtual wxFilterOutputStream* wxFilterClassFactory::NewStream (wxOutputStream & *stream*) const [pure virtual]

Create a new input or output stream to decompress or compress a given stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

virtual wxFilterInputStream* wxFilterClassFactory::NewStream (wxInputStream * *stream*) const [pure virtual]

Create a new input or output stream to decompress or compress a given stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

virtual wxFilterOutputStream* wxFilterClassFactory::NewStream (wxOutputStream * *stream*) const [pure virtual]

Create a new input or output stream to decompress or compress a given stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

wxString wxFilterClassFactory::PopExtension (const wxString & *location*) const

Remove the file extension of *location* if it is one of the file extensions handled by this factory.

void wxFilterClassFactory::PushFront ( )

Adds this class factory to the list returned by GetFirst()/GetNext().

It is not necessary to do this to use the filter streams. It is usually used when implementing streams, typically the implementation will add a static instance of its factory class.

It can also be used to change the order of a factory already in the list, bringing it to the front. This isn't a thread safe operation so can't be done when other threads are running that will be using the list.

The list does not take ownership of the factory.

void wxFilterClassFactory::Remove ( )

Removes this class factory from the list returned by GetFirst()/GetNext().

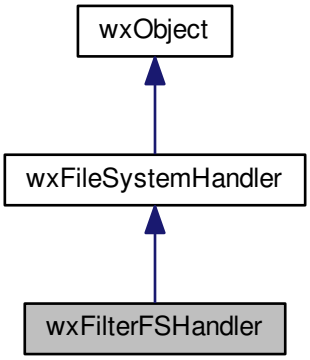
Removing from the list isn't a thread safe operation so can't be done when other threads are running that will be using the list.

The list does not own the factories, so removing a factory does not delete it.

21.251 wxFilterFSHandler Class Reference

`#include <wx/fs_filter.h>`

Inheritance diagram for wxFilterFSHandler:



21.251.1 Detailed Description

Filter file system handler.

Public Member Functions

- wxFilterFSHandler ()
- virtual ~wxFilterFSHandler ()

Additional Inherited Members

21.251.2 Constructor & Destructor Documentation

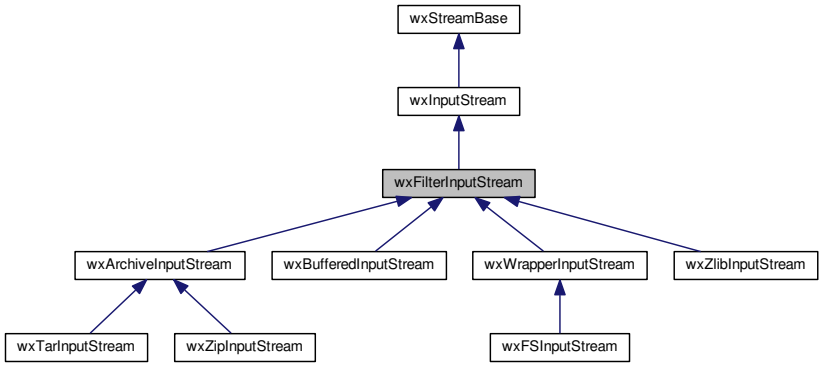
wxFilterFSHandler::wxFilterFSHandler ( )

virtual wxFilterFSHandler::~wxFilterFSHandler ( ) [virtual]

21.252 wxFilterInputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxFilterInputStream:



21.252.1 Detailed Description

A filter stream has the capability of a normal stream but it can be placed on top of another stream.

So, for example, it can uncompress or decrypt the data which are read from another stream and pass it to the requester.

Note

The interface of this class is the same as that of wxInputStream. Only a constructor differs and it is documented below.

Library: wxBase

Category: Streams

See also

wxFilterClassFactory, wxFilterOutputStream

Public Member Functions

- wxFilterInputStream (wxInputStream &stream)
  Initializes a "filter" stream.
- wxFilterInputStream (wxInputStream *stream)
  Initializes a "filter" stream.

Additional Inherited Members

21.252.2 Constructor & Destructor Documentation

wxFilterInputStream::wxFilterInputStream (wxInputStream & *stream*)

Initializes a "filter" stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

wxFilterInputStream::wxFilterInputStream (wxInputStream * *stream*)

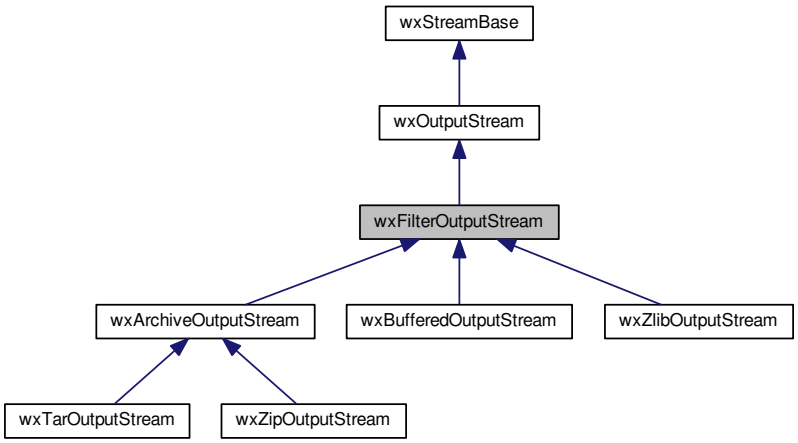
Initializes a "filter" stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

21.253 wxFilterOutputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxFilterOutputStream:



21.253.1 Detailed Description

A filter stream has the capability of a normal stream but it can be placed on top of another stream.

So, for example, it can compress, encrypt the data which are passed to it and write them to another stream.

Note

The use of this class is exactly the same as of wxOutputStream. Only a constructor differs and it is documented below.

Library: wxBase

Category: Streams

See also

wxFilterClassFactory, wxFilterInputStream

Public Member Functions

- wxFilterOutputStream (wxOutputStream &stream)

Initializes a "filter" stream.

- wxFilterOutputStream (wxOutputStream ∗stream)

  Initializes a "filter" stream.

Additional Inherited Members

21.253.2 Constructor & Destructor Documentation

wxFilterOutputStream::wxFilterOutputStream (wxOutputStream & *stream*)

Initializes a "filter" stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

wxFilterOutputStream::wxFilterOutputStream (wxOutputStream ∗ *stream*)

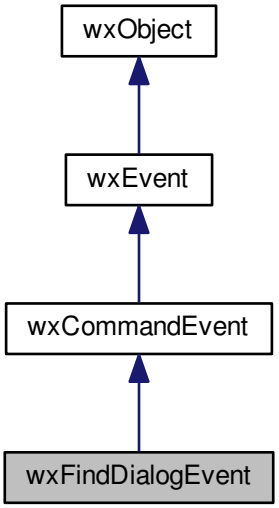
Initializes a "filter" stream.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

21.254 wxFindDialogEvent Class Reference

`#include <wx/fdrepdlg.h>`

Inheritance diagram for wxFindDialogEvent:



21.254.1 Detailed Description

wxFindReplaceDialog events.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFindDialogEvent& event)

Event macros:

- EVT_FIND(id, func): Find button was pressed in the dialog.
- EVT_FIND_NEXT(id, func): Find next button was pressed in the dialog.
- EVT_FIND_REPLACE(id, func): Replace button was pressed in the dialog.
- EVT_FIND_REPLACE_ALL(id, func): Replace all button was pressed in the dialog.
- EVT_FIND_CLOSE(id, func): The dialog is being destroyed, any pointers to it cannot be used any longer.

Library: wxCore

Category: Events

Public Member Functions

- wxFindDialogEvent (wxEventType commandType=wxEVT_NULL, int id=0)

  Constructor used by wxWidgets only.
- wxFindReplaceDialog * GetDialog () const

  Return the pointer to the dialog which generated this event.
- wxString GetFindString () const

  Return the string to find (never empty).
- int GetFlags () const

  Get the currently selected flags: this is the combination of the wxFindReplaceFlags enumeration values.
- const wxString & GetReplaceString () const

  Return the string to replace the search string with (only for replace and replace all events).

Additional Inherited Members

21.254.2 Constructor & Destructor Documentation

wxFindDialogEvent::wxFindDialogEvent (wxEventType *commandType* = wxEVT_NULL, int *id* = 0)

Constructor used by wxWidgets only.

21.254.3 Member Function Documentation

wxFindReplaceDialog* wxFindDialogEvent::GetDialog ( ) const

Return the pointer to the dialog which generated this event.

wxString wxFindDialogEvent::GetFindString ( ) const

Return the string to find (never empty).

int wxFindDialogEvent::GetFlags ( ) const

Get the currently selected flags: this is the combination of the wxFindReplaceFlags enumeration values.

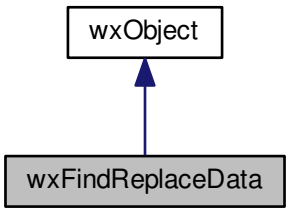
const wxString& wxFindDialogEvent::GetReplaceString ( ) const

Return the string to replace the search string with (only for replace and replace all events).

21.255 wxFindReplaceData Class Reference

`#include <wx/fdrepdlg.h>`

Inheritance diagram for wxFindReplaceData:



21.255.1 Detailed Description

wxFindReplaceData holds the data for wxFindReplaceDialog.

It is used to initialize the dialog with the default values and will keep the last values from the dialog when it is closed. It is also updated each time a wxFindDialogEvent is generated so instead of using the wxFindDialogEvent methods you can also directly query this object.

Note that all `SetXXX()` methods may only be called before showing the dialog and calling them has no effect later.

Library: wxCore

Category: Common Dialogs, Data Structures

Public Member Functions

- wxFindReplaceData (wxUint32 flags=0)

  Constructor initializes the flags to default value (0).
- const wxString & GetFindString () const

  Get the string to find.
- int GetFlags () const

  Get the combination of `wxFindReplaceFlags` values.
- const wxString & GetReplaceString () const

Get the replacement string.

- void SetFindString (const wxString &str)

  Set the string to find (used as initial value by the dialog).

- void SetFlags (wxUint32 flags)

  Set the flags to use to initialize the controls of the dialog.

- void SetReplaceString (const wxString &str)

  Set the replacement string (used as initial value by the dialog).

Additional Inherited Members

21.255.2 Constructor & Destructor Documentation

wxFindReplaceData::wxFindReplaceData (wxUint32 *flags* = `0`)

Constructor initializes the flags to default value (0).

21.255.3 Member Function Documentation

const wxString& wxFindReplaceData::GetFindString ( ) const

Get the string to find.

int wxFindReplaceData::GetFlags ( ) const

Get the combination of `wxFindReplaceFlags` values.

const wxString& wxFindReplaceData::GetReplaceString ( ) const

Get the replacement string.

void wxFindReplaceData::SetFindString (const wxString & *str*)

Set the string to find (used as initial value by the dialog).

void wxFindReplaceData::SetFlags (wxUint32 *flags*)

Set the flags to use to initialize the controls of the dialog.

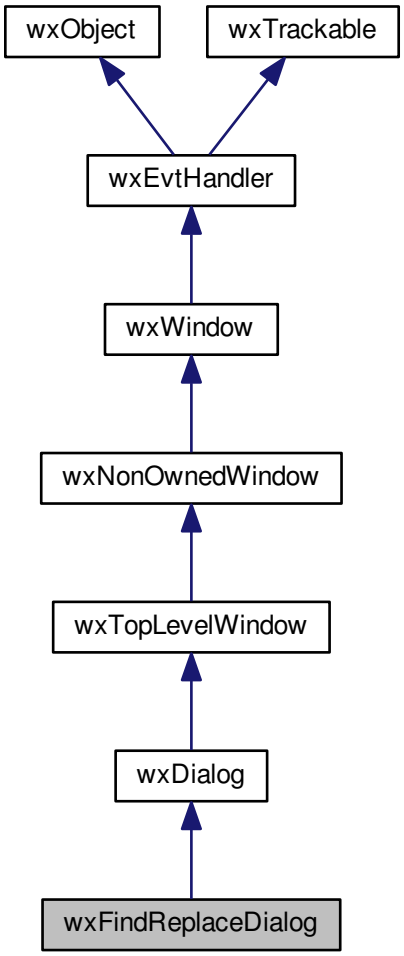
void wxFindReplaceData::SetReplaceString (const wxString & *str*)

Set the replacement string (used as initial value by the dialog).

21.256 wxFindReplaceDialog Class Reference

`#include <wx/fdrepdlg.h>`

Inheritance diagram for wxFindReplaceDialog:



21.256.1 Detailed Description

wxFindReplaceDialog is a standard modeless dialog which is used to allow the user to search for some text (and possibly replace it with something else).

The actual searching is supposed to be done in the owner window which is the parent of this dialog. Note that it means that unlike for the other standard dialogs this one **must** have a parent window. Also note that there is no way to use this dialog in a modal way; it is always, by design and implementation, modeless.

Please see the Dialogs Sample sample for an example of using it.

Library: wxCore

Category: Common Dialogs

Public Member Functions

- wxFindReplaceDialog ()
- wxFindReplaceDialog (wxWindow *parent, wxFindReplaceData *data, const wxString &title, int style=0)

  After using default constructor Create() must be called.
- virtual ~wxFindReplaceDialog ()

  Destructor.
- bool Create (wxWindow *parent, wxFindReplaceData *data, const wxString &title, int style=0)

  Creates the dialog; use wxWindow::Show to show it on screen.
- const wxFindReplaceData * GetData () const

  Get the wxFindReplaceData object used by this dialog.

Additional Inherited Members

21.256.2 Constructor & Destructor Documentation

wxFindReplaceDialog::wxFindReplaceDialog ( )

wxFindReplaceDialog::wxFindReplaceDialog (wxWindow * *parent*, wxFindReplaceData * *data*, const wxString & *title*, int *style* = `0`)

After using default constructor Create() must be called.

The *parent* and *data* parameters must be non-NULL.

virtual wxFindReplaceDialog::~wxFindReplaceDialog ( ) `[virtual]`

Destructor.

21.256.3 Member Function Documentation

bool wxFindReplaceDialog::Create (wxWindow * *parent*, wxFindReplaceData * *data*, const wxString & *title*, int *style* = `0`)

Creates the dialog; use wxWindow::Show to show it on screen.

The *parent* and *data* parameters must be non-NULL.

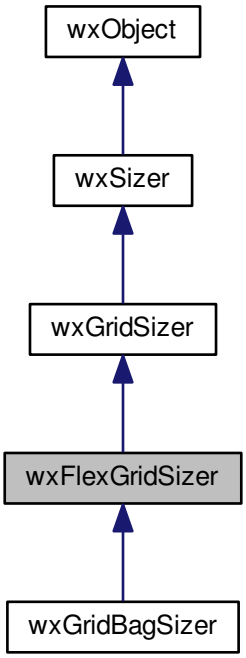
const wxFindReplaceData* wxFindReplaceDialog::GetData ( ) const

Get the wxFindReplaceData object used by this dialog.

21.257 wxFlexGridSizer Class Reference

`#include <wx/sizer.h>`

Inheritance diagram for wxFlexGridSizer:



21.257.1 Detailed Description

A flex grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the wxGridSizer.

Since wxWidgets 2.5.0, wxFlexGridSizer can also size items equally in one direction but unequally ("flexibly") in the other. If the sizer is only flexible in one direction (this can be changed using wxFlexGridSizer::SetFlexibleDirection), it needs to be decided how the sizer should grow in the other ("non-flexible") direction in order to fill the available space. The wxFlexGridSizer::SetNonFlexibleGrowMode() method serves this purpose.

Library: wxCore

Category: Window Layout

See also

wxSizer, Sizers Overview

Public Member Functions

- void AddGrowableCol (size_t idx, int proportion=0)

  Specifies that column idx (starting from zero) should be grown if there is extra space available to the sizer.
- void AddGrowableRow (size_t idx, int proportion=0)

Specifies that row idx (starting from zero) should be grown if there is extra space available to the sizer.

- int GetFlexibleDirection () const

  Returns a wxOrientation value that specifies whether the sizer flexibly resizes its columns, rows, or both (default).

- wxFlexSizerGrowMode GetNonFlexibleGrowMode () const

  Returns the value that specifies how the sizer grows in the "non-flexible" direction if there is one.

- bool IsColGrowable (size_t idx)

  Returns true if column idx is growable.

- bool IsRowGrowable (size_t idx)

  Returns true if row idx is growable.

- void RemoveGrowableCol (size_t idx)

  Specifies that the idx column index is no longer growable.

- void RemoveGrowableRow (size_t idx)

  Specifies that the idx row index is no longer growable.

- void SetFlexibleDirection (int direction)

  Specifies whether the sizer should flexibly resize its columns, rows, or both.

- void SetNonFlexibleGrowMode (wxFlexSizerGrowMode mode)

  Specifies how the sizer should grow in the non-flexible direction if there is one (so SetFlexibleDirection() must have been called previously).

- const wxArrayInt & GetRowHeights () const

  Returns a read-only array containing the heights of the rows in the sizer.

- const wxArrayInt & GetColWidths () const

  Returns a read-only array containing the widths of the columns in the sizer.

- virtual void RecalcSizes ()

  This method is abstract and has to be overwritten by any derived class.

- virtual wxSize CalcMin ()

  This method is abstract and has to be overwritten by any derived class.

- wxFlexGridSizer (int cols, int vgap, int hgap)

  wxFlexGridSizer constructors.

- wxFlexGridSizer (int cols, const wxSize &gap=wxSize(0, 0))

  wxFlexGridSizer constructors.

- wxFlexGridSizer (int rows, int cols, int vgap, int hgap)

  wxFlexGridSizer constructors.

- wxFlexGridSizer (int rows, int cols, const wxSize &gap)

  wxFlexGridSizer constructors.

Additional Inherited Members

21.257.2 Constructor & Destructor Documentation

wxFlexGridSizer::wxFlexGridSizer (int *cols,* int *vgap,* int *hgap*)

wxFlexGridSizer constructors.

Please see wxGridSizer::wxGridSizer documentation.

Since

2.9.1 (except for the four argument overload)

wxFlexGridSizer::wxFlexGridSizer (int *cols*, const wxSize & *gap* = wxSize(0, 0))

wxFlexGridSizer constructors.

Please see wxGridSizer::wxGridSizer documentation.

Since

2.9.1 (except for the four argument overload)

wxFlexGridSizer::wxFlexGridSizer (int *rows*, int *cols*, int *vgap*, int *hgap*)

wxFlexGridSizer constructors.

Please see wxGridSizer::wxGridSizer documentation.

Since

2.9.1 (except for the four argument overload)

wxFlexGridSizer::wxFlexGridSizer (int *rows*, int *cols*, const wxSize & *gap*)

wxFlexGridSizer constructors.

Please see wxGridSizer::wxGridSizer documentation.

Since

2.9.1 (except for the four argument overload)

21.257.3 Member Function Documentation

void wxFlexGridSizer::AddGrowableCol (size_t *idx*, int *proportion* = 0)

Specifies that column *idx* (starting from zero) should be grown if there is extra space available to the sizer.

The *proportion* parameter has the same meaning as the stretch factor for the sizers (see wxBoxSizer) except that if all proportions are 0, then all columns are resized equally (instead of not being resized at all).

Notice that the column must not be already growable, if you need to change the proportion you must call RemoveGrowableCol() first and then make it growable (with a different proportion) again. You can use IsColGrowable() to check whether a column is already growable.

void wxFlexGridSizer::AddGrowableRow (size_t *idx*, int *proportion* = 0)

Specifies that row idx (starting from zero) should be grown if there is extra space available to the sizer.

This is identical to AddGrowableCol() except that it works with rows and not columns.

virtual wxSize wxFlexGridSizer::CalcMin ( ) [virtual]

This method is abstract and has to be overwritten by any derived class.

Here, the sizer will do the actual calculation of its children's minimal sizes.

Reimplemented from wxGridSizer.

Reimplemented in wxGridBagSizer.

const wxArrayInt& wxFlexGridSizer::GetColWidths ( ) const

Returns a read-only array containing the widths of the columns in the sizer.

int wxFlexGridSizer::GetFlexibleDirection ( ) const

Returns a wxOrientation value that specifies whether the sizer flexibly resizes its columns, rows, or both (default).

Returns

One of the following values:

- wxVERTICAL: Rows are flexibly sized.
- wxHORIZONTAL: Columns are flexibly sized.
- wxBOTH: Both rows and columns are flexibly sized (this is the default value).

See also

SetFlexibleDirection()

wxFlexSizerGrowMode wxFlexGridSizer::GetNonFlexibleGrowMode ( ) const

Returns the value that specifies how the sizer grows in the "non-flexible" direction if there is one.

The behaviour of the elements in the flexible direction (i.e. both rows and columns by default, or rows only if GetFlexibleDirection() is `wxVERTICAL` or columns only if it is `wxHORIZONTAL`) is always governed by their proportion as specified in the call to AddGrowableRow() or AddGrowableCol(). What happens in the other direction depends on the value of returned by this function as described below.

Returns

One of the following values:

- wxFLEX_GROWMODE_NONE: Sizer doesn't grow its elements at all in the non-flexible direction.
- wxFLEX_GROWMODE_SPECIFIED: Sizer honors growable columns/rows set with AddGrowableCol() and AddGrowableRow() in the non-flexible direction as well. In this case equal sizing applies to minimum sizes of columns or rows (this is the default value).
- wxFLEX_GROWMODE_ALL: Sizer equally stretches all columns or rows in the non-flexible direction, independently of the proportions applied in the flexible direction.

See also

SetFlexibleDirection(), SetNonFlexibleGrowMode()

const wxArrayInt& wxFlexGridSizer::GetRowHeights ( ) const

Returns a read-only array containing the heights of the rows in the sizer.

bool wxFlexGridSizer::IsColGrowable (size_t *idx*)

Returns true if column *idx* is growable.

Since

2.9.0

bool wxFlexGridSizer::IsRowGrowable (size_t *idx*)

Returns true if row *idx* is growable.

Since

2.9.0

virtual void wxFlexGridSizer::RecalcSizes ( ) [virtual]

This method is abstract and has to be overwritten by any derived class.

Here, the sizer will do the actual calculation of its children's positions and sizes.

Reimplemented from wxGridSizer.

Reimplemented in wxGridBagSizer.

void wxFlexGridSizer::RemoveGrowableCol (size_t *idx*)

Specifies that the *idx* column index is no longer growable.

void wxFlexGridSizer::RemoveGrowableRow (size_t *idx*)

Specifies that the *idx* row index is no longer growable.

void wxFlexGridSizer::SetFlexibleDirection (int *direction*)

Specifies whether the sizer should flexibly resize its columns, rows, or both.

Argument *direction* can be `wxVERTICAL`, `wxHORIZONTAL` or `wxBOTH` (which is the default value). Any other value is ignored.

See GetFlexibleDirection() for the explanation of these values. Note that this method does not trigger relayout.

void wxFlexGridSizer::SetNonFlexibleGrowMode (wxFlexSizerGrowMode *mode*)

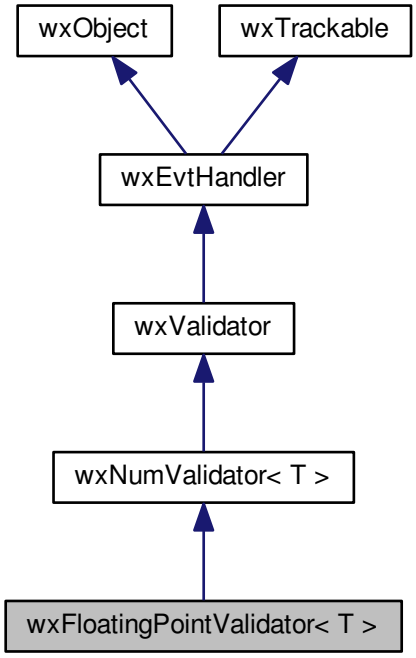
Specifies how the sizer should grow in the non-flexible direction if there is one (so SetFlexibleDirection() must have been called previously).

Argument *mode* can be one of those documented in GetNonFlexibleGrowMode(), please see there for their explanation. Note that this method does not trigger relayout.

21.258 wxFloatingPointValidator< T > Class Template Reference

`#include <wx/valnum.h>`

Inheritance diagram for wxFloatingPointValidator< T >:



21.258.1 Detailed Description

template<typename T>class wxFloatingPointValidator< T >

Validator for text entries used for floating point numbers entry.

This validator can be used with wxTextCtrl or wxComboBox (and potentially any other class implementing wxTextEntry interface) to check that only valid floating point values can be entered into them. Currently only fixed format is supported on input, i.e. scientific format with mantissa and exponent is not supported.

This template class can be instantiated for either `float` or `double`, `long double` values are not currently supported.

Similarly to wxIntegerValidator<>, the range for the accepted values is by default set appropriately for the type. Additionally, this validator allows to specify the maximum number of digits that can be entered after the decimal separator. By default this is also set appropriately for the type used, e.g. 6 for `float` and 15 for `double` on a typical IEEE-754-based implementation. As with the range, the precision can be restricted after the validator creation if necessary.

When the validator displays numbers with decimal or thousands separators, the characters used for the separators (usually "." or ",") depend on the locale set with wxLocale (note that you shouldn't change locale with setlocale() as this can result in a mismatch between the separators used by wxLocale and the one used by the run-time library).

A simple example of using this class:

```
class MyDialog : public wxDialog
{
public:
    MyDialog()
    {
        ...
```

```
        // Allow floating point numbers from 0 to 100 with 2 decimal
        // digits only and handle empty string as 0 by default.
        wxFloatingPointValidator<float>
            val(2, &m_value, wxNUM_VAL_ZERO_AS_BLANK);
        val.SetRange(0, 100);

        // Associate it with the text control:
        new wxTextCtrl(this, ..., val);
    }

private:
    float m_value;
};
```

For more information, please see wxValidator Overview.

Library: wxCore

Category: Validators

See also

wxValidator Overview, wxValidator, wxGenericValidator, wxTextValidator, wxMakeIntegerValidator()

Since

2.9.2

Public Types

- typedef T ValueType

  Type of the values this validator is used with.

Public Member Functions

- wxFloatingPointValidator (ValueType *value=NULL, int style=wxNUM_VAL_DEFAULT)

  Constructor for validator using the default precision.
- wxFloatingPointValidator (int precision, ValueType *value=NULL, int style=wxNUM_VAL_DEFAULT)

  Constructor for validator specifying the precision.
- void SetPrecision (unsigned precision)

  Set precision.

Additional Inherited Members

21.258.2 Member Typedef Documentation

template<typename T> typedef T wxFloatingPointValidator< T >::ValueType

Type of the values this validator is used with.

21.258.3 Constructor & Destructor Documentation

template<typename T> wxFloatingPointValidator< T >::wxFloatingPointValidator (ValueType * *value* = NULL, int *style* = wxNUM_VAL_DEFAULT)

Constructor for validator using the default precision.

Parameters

value	A pointer to the variable associated with the validator. If non NULL, this variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).
style	A combination of wxNumValidatorStyle enum values.

template<typename T> wxFloatingPointValidator< T >::wxFloatingPointValidator (int *precision,* ValueType * *value* = NULL, int *style* = wxNUM_VAL_DEFAULT)

Constructor for validator specifying the precision.

Parameters

value	A pointer to the variable associated with the validator. If non NULL, this variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).
style	A combination of wxNumValidatorStyle enum values.
precision	The number of decimal digits after the decimal separator to show and accept.

21.258.4 Member Function Documentation

template<typename T> void wxFloatingPointValidator< T >::SetPrecision (unsigned *precision*)

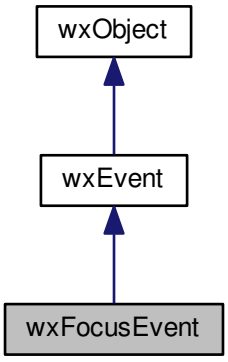
Set precision.

Precision is the number of digits shown (and accepted on input) after the decimal point. By default this is set to the maximal precision supported by the type handled by the validator in its constructor.

21.259 wxFocusEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxFocusEvent:



21.259.1 Detailed Description

A focus event is sent when a window's focus changes.

The window losing focus receives a "kill focus" event while the window gaining it gets a "set focus" one.

Notice that the set focus event happens both when the user gives focus to the window (whether using the mouse or keyboard) and when it is done from the program itself using wxWindow::SetFocus.

The focus event handlers should almost invariably call wxEvent::Skip() on their event argument to allow the default handling to take place. Failure to do this may result in incorrect behaviour of the native controls. Also note that wxE↩ VT_KILL_FOCUS handler must not call wxWindow::SetFocus() as this, again, is not supported by all native controls. If you need to do this, consider using the Delayed Action Mechanism described in wxIdleEvent documentation.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFocusEvent& event)

Event macros:

- EVT_SET_FOCUS(func): Process a `wxEVT_SET_FOCUS` event.
- EVT_KILL_FOCUS(func): Process a `wxEVT_KILL_FOCUS` event.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxFocusEvent (wxEventType eventType=wxEVT_NULL, int id=0)

  Constructor.
- wxWindow * GetWindow () const

  Returns the window associated with this event, that is the window which had the focus before for the `wxEVT_SET↩ _FOCUS` event and the window which is going to receive focus for the `wxEVT_KILL_FOCUS` one.
- void SetWindow (wxWindow *win)

Additional Inherited Members

21.259.2 Constructor & Destructor Documentation

wxFocusEvent::wxFocusEvent (wxEventType *eventType* = wxEVT_NULL, int *id* = 0)

Constructor.

21.259.3 Member Function Documentation

wxWindow* wxFocusEvent::GetWindow ( ) const

Returns the window associated with this event, that is the window which had the focus before for the `wxEVT_SE↩T_FOCUS` event and the window which is going to receive focus for the `wxEVT_KILL_FOCUS` one.

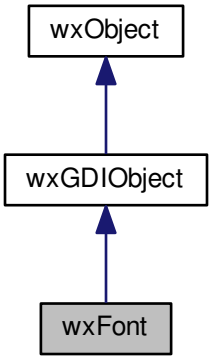
Warning: the window pointer may be NULL!

void wxFocusEvent::SetWindow (wxWindow * *win*)

21.260 wxFont Class Reference

`#include <wx/font.h>`

Inheritance diagram for wxFont:



21.260.1 Detailed Description

A font is an object which determines the appearance of text.

Fonts are used for drawing text to a device context, and setting the appearance of a window's text, see wxDC::Set↩Font() and wxWindow::SetFont().

The easiest way to create a custom font is to use wxFontInfo object to specify the font attributes and then use wxFont::wxFont(const wxFontInfo&) constructor. Alternatively, you could start with one of the pre-defined fonts or use wxWindow::GetFont() and modify the font, e.g. by increasing its size using MakeLarger() or changing its weight using MakeBold().

This class uses reference counting and copy-on-write internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

You can retrieve the current system font settings with wxSystemSettings.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullFont, wxNORMAL_FONT, wxSMALL_FONT, wxITALIC_FONT, wxSWISS_F↩ ONT

See also

wxFont Overview, wxDC::SetFont, wxDC::DrawText, wxDC::GetTextExtent, wxFontDialog, wxSystemSettings

Public Member Functions

- wxFont ()

  Default ctor.
- wxFont (const wxFont &font)

  Copy constructor, uses reference counting.
- wxFont (const wxFontInfo &font)

  Creates a font object using the specified font description.
- wxFont (int pointSize, wxFontFamily family, wxFontStyle style, wxFontWeight weight, bool underline=false, const wxString &faceName=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  Creates a font object with the specified attributes and size in points.
- wxFont (const wxSize &pixelSize, wxFontFamily family, wxFontStyle style, wxFontWeight weight, bool underline=false, const wxString &faceName=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_↩ DEFAULT)

  Creates a font object with the specified attributes and size in pixels.
- wxFont (const wxString &nativeInfoString)

  Constructor from font description string.
- wxFont (const wxNativeFontInfo &nativeInfo)

  Construct font from a native font info structure.
- virtual ∼wxFont ()

  Destructor.
- bool operator!= (const wxFont &font) const

  Inequality operator.
- bool operator== (const wxFont &font) const

  Equality operator.
- wxFont & operator= (const wxFont &font)

  Assignment operator, using reference counting.

Getters

- wxFont GetBaseFont () const

  Returns a font with the same face/size as the given one but with normal weight and style and not underlined nor stricken through.
- virtual wxFontEncoding GetEncoding () const

  Returns the encoding of this font.
- virtual wxString GetFaceName () const

  Returns the face name associated with the font, or the empty string if there is no face information.
- virtual wxFontFamily GetFamily () const

  Gets the font family if possible.
- wxString GetNativeFontInfoDesc () const

  Returns the platform-dependent string completely describing this font.
- wxString GetNativeFontInfoUserDesc () const

  Returns a user-friendly string for this font object.
- const wxNativeFontInfo ∗ GetNativeFontInfo () const

  Returns a font with the same face/size as the given one but with normal weight and style and not underlined nor stricken through.
- virtual int GetPointSize () const

  Gets the point size.

- virtual wxSize GetPixelSize () const
  Gets the pixel size.
- virtual wxFontStyle GetStyle () const
  Gets the font style.
- virtual bool GetUnderlined () const
  Returns true if the font is underlined, false otherwise.
- virtual bool GetStrikethrough () const
  Returns true if the font is stricken-through, false otherwise.
- virtual wxFontWeight GetWeight () const
  Gets the font weight.
- virtual bool IsFixedWidth () const
  Returns true if the font is a fixed width (or monospaced) font, false if it is a proportional one or font is invalid.
- virtual bool IsOk () const
  Returns true if this object is a valid font, false otherwise.

Similar fonts creation

The functions in this section either modify the font in place or create a new font similar to the given one but with its weight, style or size changed.

- wxFont Bold () const
  Returns a bold version of this font.
- wxFont Italic () const
  Returns an italic version of this font.
- wxFont Larger () const
  Returns a larger version of this font.
- wxFont Smaller () const
  Returns a smaller version of this font.
- wxFont Underlined () const
  Returns underlined version of this font.
- wxFont Strikethrough () const
  Returns stricken-through version of this font.
- wxFont & MakeBold ()
  Changes this font to be bold.
- wxFont & MakeItalic ()
  Changes this font to be italic.
- wxFont & MakeLarger ()
  Changes this font to be larger.
- wxFont & MakeSmaller ()
  Changes this font to be smaller.
- wxFont & MakeUnderlined ()
  Changes this font to be underlined.
- wxFont & MakeStrikethrough ()
  Changes this font to be stricken-through.
- wxFont & Scale (float x)
  Changes the size of this font.
- wxFont Scaled (float x) const
  Returns a scaled version of this font.

Setters

These functions internally recreate the native font object with the new specified property.

- virtual void SetEncoding (wxFontEncoding encoding)
  Sets the encoding for this font.
- virtual bool SetFaceName (const wxString &faceName)
  Sets the facename for the font.
- virtual void SetFamily (wxFontFamily family)
  Sets the font family.
- bool SetNativeFontInfo (const wxString &info)

Creates the font corresponding to the given native font description string which must have been previously returned by GetNativeFontInfoDesc().

- bool SetNativeFontInfoUserDesc (const wxString &info)

  Creates the font corresponding to the given native font description string and returns true if the creation was successful.
- void SetNativeFontInfo (const wxNativeFontInfo &info)

  Sets the encoding for this font.
- virtual void SetPointSize (int pointSize)

  Sets the point size.
- virtual void SetPixelSize (const wxSize &pixelSize)

  Sets the pixel size.
- virtual void SetStyle (wxFontStyle style)

  Sets the font style.
- void SetSymbolicSize (wxFontSymbolicSize size)

  Sets the font size using a predefined symbolic size name.
- void SetSymbolicSizeRelativeTo (wxFontSymbolicSize size, int base)

  Sets the font size compared to the base font size.
- virtual void SetUnderlined (bool underlined)

  Sets underlining.
- virtual void SetStrikethrough (bool strikethrough)

  Sets strike-through attribute of the font.
- virtual void SetWeight (wxFontWeight weight)

  Sets the font weight.

Static Public Member Functions

- static wxFontEncoding GetDefaultEncoding ()

  Returns the current application's default encoding.
- static void SetDefaultEncoding (wxFontEncoding encoding)

  Sets the default font encoding.

- static wxFont * New (int pointSize, wxFontFamily family, wxFontStyle style, wxFontWeight weight, bool underline=false, const wxString &faceName=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.
- static wxFont * New (int pointSize, wxFontFamily family, int flags=wxFONTFLAG_DEFAULT, const wxString &faceName=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.
- static wxFont * New (const wxSize &pixelSize, wxFontFamily family, wxFontStyle style, wxFontWeight weight, bool underline=false, const wxString &faceName=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.
- static wxFont * New (const wxSize &pixelSize, wxFontFamily family, int flags=wxFONTFLAG_DEFAULT, const wxString &faceName=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.
- static wxFont * New (const wxNativeFontInfo &nativeInfo)

  This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.
- static wxFont * New (const wxString &nativeInfoString)

  This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Additional Inherited Members

21.260.2 Constructor & Destructor Documentation

wxFont::wxFont ( )

Default ctor.

wxFont::wxFont (const wxFont & *font*)

Copy constructor, uses reference counting.

wxFont::wxFont (const wxFontInfo & *font*)

Creates a font object using the specified font description.

This is the preferred way to create font objects as using this ctor results in more readable code and it is also extensible, e.g. it could continue to be used if support for more font attributes is added in the future. For example, this constructor provides the only way of creating fonts with strike-through style.

Example of creating a font using this ctor:

```
wxFont font(wxFontInfo(10).Bold().Underlined());
```

Since

2.9.5

wxFont::wxFont (int *pointSize*, wxFontFamily *family*, wxFontStyle *style*, wxFontWeight *weight*, bool *underline* = `false`, const wxString & *faceName* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT)

Creates a font object with the specified attributes and size in points.

Notice that the use of this constructor is often more verbose and less readable than using wxFont(const wxFontInfo& font), e.g. the example in that constructor documentation would need to be written as:

```
wxFont font(10, wxFONTFAMILY_DEFAULT,
      wxFONTSTYLE_NORMAL,
            wxFONTWEIGHT_BOLD, true);
```

Parameters

pointSize	Size in points. See SetPointSize() for more info.
family	The font family: a generic portable way of referring to fonts without specifying a facename. This parameter must be one of the wxFontFamily enumeration values. If the *faceName* argument is provided, then it overrides the font family.
style	One of `wxFONTSTYLE_NORMAL`, `wxFONTSTYLE_SLANT` and `wxFONTSTYLE_ITALIC`.
weight	Font weight, sometimes also referred to as font boldness. One of the wxFontWeight enumeration values.

underline	The value can be true or false. At present this has an effect on Windows and Motif 2.x only.
faceName	An optional string specifying the face name to be used. If it is an empty string, a default face name will be chosen based on the family.
encoding	An encoding which may be one of the enumeration values of wxFontEncoding. If the specified encoding isn't available, no font is created (see also Font Encodings).

Remarks

If the desired font does not exist, the closest match will be chosen. Under Windows, only scalable TrueType fonts are used.

wxFont::wxFont (const wxSize & *pixelSize*, wxFontFamily *family*, wxFontStyle *style*, wxFontWeight *weight*, bool *underline* = `false`, const wxString & *faceName* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT)

Creates a font object with the specified attributes and size in pixels.

Notice that the use of this constructor is often more verbose and less readable than the use of constructor from wxFontInfo, consider using that constructor instead.

Parameters

pixelSize	Size in pixels. See SetPixelSize() for more info.
family	The font family: a generic portable way of referring to fonts without specifying a facename. This parameter must be one of the wxFontFamily enumeration values. If the *faceName* argument is provided, then it overrides the font family.
style	One of `wxFONTSTYLE_NORMAL`, `wxFONTSTYLE_SLANT` and `wxFONTSTYLE_ITALIC`.
weight	Font weight, sometimes also referred to as font boldness. One of the wxFontWeight enumeration values.
underline	The value can be true or false. At present this has an effect on Windows and Motif 2.x only.
faceName	An optional string specifying the face name to be used. If it is an empty string, a default face name will be chosen based on the family.
encoding	An encoding which may be one of the enumeration values of wxFontEncoding. If the specified encoding isn't available, no font is created (see also Font Encodings).

Remarks

If the desired font does not exist, the closest match will be chosen. Under Windows, only scalable TrueType fonts are used.

wxFont::wxFont (const wxString & *nativeInfoString*)

Constructor from font description string.

This constructor uses SetNativeFontInfo() to initialize the font. If *fontdesc* is invalid the font remains uninitialized, i.e. its IsOk() method will return false.

wxFont::wxFont (const wxNativeFontInfo & *nativeInfo*)

Construct font from a native font info structure.

virtual wxFont::~wxFont ( ) [virtual]

Destructor.

See reference-counted object destruction for more info.

Remarks

Although all remaining fonts are deleted when the application exits, the application should try to clean up all fonts itself. This is because wxWidgets cannot know if a pointer to the font object is stored in an application data structure, and there is a risk of double deletion.

21.260.3 Member Function Documentation

wxFont wxFont::Bold ( ) const

Returns a bold version of this font.

See also

MakeBold()

Since

2.9.1

wxFont wxFont::GetBaseFont ( ) const

Returns a font with the same face/size as the given one but with normal weight and style and not underlined nor stricken through.

Since

3.1.0

static wxFontEncoding wxFont::GetDefaultEncoding ( ) [static]

Returns the current application's default encoding.

See also

Font Encodings, SetDefaultEncoding()

virtual wxFontEncoding wxFont::GetEncoding ( ) const [virtual]

Returns the encoding of this font.

Note that under wxGTK the returned value is always `wxFONTENCODING_UTF8`.

See also

SetEncoding()

virtual wxString wxFont::GetFaceName ( ) const [virtual]

Returns the face name associated with the font, or the empty string if there is no face information.

See also

SetFaceName()

virtual wxFontFamily wxFont::GetFamily ( ) const `[virtual]`

Gets the font family if possible.

As described in wxFontFamily docs the returned value acts as a rough, basic classification of the main font properties (look, spacing).

If the current font face name is not recognized by wxFont or by the underlying system, `wxFONTFAMILY_DEFAULT` is returned.

Note that currently this function is not very precise and so not particularly useful. Font families mostly make sense only for font creation, see SetFamily().

See also

SetFamily()

const wxNativeFontInfo* wxFont::GetNativeFontInfo ( ) const

Returns a font with the same face/size as the given one but with normal weight and style and not underlined nor stricken through.

Since

3.1.0

wxString wxFont::GetNativeFontInfoDesc ( ) const

Returns the platform-dependent string completely describing this font.

Returned string is always non-empty unless the font is invalid (in which case an assert is triggered).

Note that the returned string is not meant to be shown or edited by the user: a typical use of this function is for serializing in string-form a wxFont object.

See also

SetNativeFontInfo(), GetNativeFontInfoUserDesc()

wxString wxFont::GetNativeFontInfoUserDesc ( ) const

Returns a user-friendly string for this font object.

Returned string is always non-empty unless the font is invalid (in which case an assert is triggered).

The string does not encode all wxFont infos under all platforms; e.g. under wxMSW the font family is not present in the returned string.

Some examples of the formats of returned strings (which are platform-dependent) are in SetNativeFontInfoUserDesc().

See also

SetNativeFontInfoUserDesc(), GetNativeFontInfoDesc()

virtual wxSize wxFont::GetPixelSize ( ) const [virtual]

Gets the pixel size.

Note that under wxMSW if you passed to SetPixelSize() (or to the ctor) a wxSize object with a null width value, you'll get a null width in the returned object.

See also

SetPixelSize()

virtual int wxFont::GetPointSize ( ) const [virtual]

Gets the point size.

See also

SetPointSize()

virtual bool wxFont::GetStrikethrough ( ) const [virtual]

Returns true if the font is stricken-through, false otherwise.

See also

SetStrikethrough()

Since

2.9.4

virtual wxFontStyle wxFont::GetStyle ( ) const [virtual]

Gets the font style.

See wxFontStyle for a list of valid styles.

See also

SetStyle()

virtual bool wxFont::GetUnderlined ( ) const [virtual]

Returns true if the font is underlined, false otherwise.

See also

SetUnderlined()

virtual wxFontWeight wxFont::GetWeight ( ) const [virtual]

Gets the font weight.

See wxFontWeight for a list of valid weight identifiers.

See also

SetWeight()

virtual bool wxFont::IsFixedWidth ( ) const `[virtual]`

Returns true if the font is a fixed width (or monospaced) font, false if it is a proportional one or font is invalid.

Note that this function under some platforms is different than just testing for the font family being equal to `wxFONTFAMILY_TELETYPE` because native platform-specific functions are used for the check (resulting in a more accurate return value).

virtual bool wxFont::IsOk ( ) const `[virtual]`

Returns true if this object is a valid font, false otherwise.

wxFont wxFont::Italic ( ) const

Returns an italic version of this font.

See also

MakeItalic()

Since

2.9.1

wxFont wxFont::Larger ( ) const

Returns a larger version of this font.

The font size is multiplied by `1.2`, the factor of `1.2` being inspired by the W3C CSS specification.

See also

MakeLarger(), Smaller(), Scaled()

Since

2.9.1

wxFont& wxFont::MakeBold ( )

Changes this font to be bold.

See also

Bold()

Since

2.9.1

wxFont& wxFont::MakeItalic ( )

Changes this font to be italic.

See also

Italic()

Since

2.9.1

wxFont& wxFont::MakeLarger ( )

Changes this font to be larger.

The font size is multiplied by `1.2`, the factor of `1.2` being inspired by the W3C CSS specification.

See also

Larger(), MakeSmaller(), Scale()

Since

2.9.1

wxFont& wxFont::MakeSmaller ( )

Changes this font to be smaller.

The font size is divided by `1.2`, the factor of `1.2` being inspired by the W3C CSS specification.

See also

Smaller(), MakeLarger(), Scale()

Since

2.9.1

wxFont& wxFont::MakeStrikethrough ( )

Changes this font to be stricken-through.

Currently stricken-through fonts are only supported in wxMSW, wxGTK and OSX.

See also

Strikethrough()

Since

2.9.4

wxFont& wxFont::MakeUnderlined ( )

Changes this font to be underlined.

See also

Underlined()

Since

2.9.2

static wxFont* wxFont::New (int *pointSize*, wxFontFamily *family*, wxFontStyle *style*, wxFontWeight *weight*, bool *underline* = `false`, const wxString & *faceName* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT) `[static]`

This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Their use is discouraged, use wxFont constructor from wxFontInfo instead.

static wxFont* wxFont::New (int *pointSize*, wxFontFamily *family*, int *flags* = wxFONTFLAG_DEFAULT, const wxString & *faceName* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT) `[static]`

This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Their use is discouraged, use wxFont constructor from wxFontInfo instead.

static wxFont* wxFont::New (const wxSize & *pixelSize*, wxFontFamily *family*, wxFontStyle *style*, wxFontWeight *weight*, bool *underline* = `false`, const wxString & *faceName* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT) `[static]`

This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Their use is discouraged, use wxFont constructor from wxFontInfo instead.

static wxFont* wxFont::New (const wxSize & *pixelSize*, wxFontFamily *family*, int *flags* = wxFONTFLAG_DEFAULT, const wxString & *faceName* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT) `[static]`

This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Their use is discouraged, use wxFont constructor from wxFontInfo instead.

static wxFont* wxFont::New (const wxNativeFontInfo & *nativeInfo*) `[static]`

This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Their use is discouraged, use wxFont constructor from wxFontInfo instead.

static wxFont* wxFont::New (const wxString & *nativeInfoString*) `[static]`

This function takes the same parameters as the relative wxFont constructor and returns a new font object allocated on the heap.

Their use is discouraged, use wxFont constructor from wxFontInfo instead.

bool wxFont::operator!= (const wxFont & *font*) const

Inequality operator.

See reference-counted object comparison for more info.

wxFont& wxFont::operator= (const wxFont & *font*)

Assignment operator, using reference counting.

bool wxFont::operator== (const wxFont & *font*) const

Equality operator.

See reference-counted object comparison for more info.

wxFont& wxFont::Scale (float *x*)

Changes the size of this font.

The font size is multiplied by the given factor (which may be less than 1 to create a smaller version of the font).

See also

Scaled(), MakeLarger(), MakeSmaller()

Since

2.9.1

wxFont wxFont::Scaled (float *x*) const

Returns a scaled version of this font.

The font size is multiplied by the given factor (which may be less than 1 to create a smaller version of the font).

See also

Scale(), Larger(), Smaller()

Since

2.9.1

static void wxFont::SetDefaultEncoding (wxFontEncoding *encoding*) `[static]`

Sets the default font encoding.

See also

Font Encodings, GetDefaultEncoding()

virtual void wxFont::SetEncoding (wxFontEncoding *encoding*) [virtual]

Sets the encoding for this font.

Note that under wxGTK this function has no effect (because the underlying Pango library always uses wxFONTENCODING_UTF8).

See also

GetEncoding()

virtual bool wxFont::SetFaceName (const wxString & *faceName*) [virtual]

Sets the facename for the font.

Parameters

faceName	A valid facename, which should be on the end-user's system.

Remarks

To avoid portability problems, don't rely on a specific face, but specify the font family instead (see wxFontFamily and SetFamily()).

Returns

true if the given face name exists; if the face name doesn't exist in the user's system then the font is invalidated (so that IsOk() will return false) and false is returned.

See also

GetFaceName(), SetFamily()

virtual void wxFont::SetFamily (wxFontFamily *family*) [virtual]

Sets the font family.

As described in wxFontFamily docs the given *family* value acts as a rough, basic indication of the main font properties (look, spacing).

Note that changing the font family results in changing the font face name.

Parameters

family	One of the wxFontFamily values.

See also

GetFamily(), SetFaceName()

bool wxFont::SetNativeFontInfo (const wxString & *info*)

Creates the font corresponding to the given native font description string which must have been previously returned by GetNativeFontInfoDesc().

If the string is invalid, font is unchanged. This function is typically used for de-serializing a wxFont object previously saved in a string-form.

Returns

true if the creation was successful.

See also

SetNativeFontInfoUserDesc()

void wxFont::SetNativeFontInfo (const wxNativeFontInfo & *info*)

Sets the encoding for this font.

Note that under wxGTK this function has no effect (because the underlying Pango library always uses `wxFONTENCODING_UTF8`).

See also

GetEncoding()

bool wxFont::SetNativeFontInfoUserDesc (const wxString & *info*)

Creates the font corresponding to the given native font description string and returns true if the creation was successful.

Unlike SetNativeFontInfo(), this function accepts strings which are user-friendly. Examples of accepted string formats are:

platform	generic syntax	example
wxGTK2	`[underlined] [strikethrough] [FACE-NAME] [bold] [oblique|italic] [POINTSIZE]`	Monospace bold 10
wxMSW	`[light|bold] [italic] [FACE-NAME] [POINTSIZE] [ENCODING]`	Tahoma 10 WINDOWS-1252

Todo add an example for wxMac

For more detailed information about the allowed syntaxes you can look at the documentation of the native API used for font-rendering (e.g. `pango_font_description_from_string` under GTK, although notice that it doesn't support the "underlined" and "strikethrough" attributes and so those are handled by wxWidgets itself).

Note that unlike SetNativeFontInfo(), this function doesn't always restore all attributes of the wxFont object under all platforms; e.g. on wxMSW the font family is not restored (because GetNativeFontInfoUserDesc doesn't return it on wxMSW). If you want to serialize/deserialize a font in string form, you should use GetNativeFontInfoDesc() and SetNativeFontInfo() instead.

See also

SetNativeFontInfo()

virtual void wxFont::SetPixelSize (const wxSize & *pixelSize*) [virtual]

Sets the pixel size.

The height parameter of *pixelSize* must be positive while the width parameter may also be zero (to indicate that you're not interested in the width of the characters: a suitable width will be chosen for best rendering).

This feature (specifying the font pixel size) is directly supported only under wxMSW and wxGTK currently; under other platforms a font with the closest size to the given one is found using binary search (this maybe slower).

See also

GetPixelSize()

virtual void wxFont::SetPointSize (int *pointSize*) [virtual]

Sets the point size.

The *point size* is defined as 1/72 of the Anglo-Saxon inch (25.4 mm): it is approximately 0.0139 inch or 352.8 um.

Parameters

pointSize	Size in points.

See also

GetPointSize()

virtual void wxFont::SetStrikethrough (bool *strikethrough*) [virtual]

Sets strike-through attribute of the font.

Currently stricken-through fonts are only supported in wxMSW, wxGTK and OSX.

Parameters

strikethrough	true to add strike-through style, false to remove it.

See also

GetStrikethrough()

Since

2.9.4

virtual void wxFont::SetStyle (wxFontStyle *style*) [virtual]

Sets the font style.

Parameters

style	One of the wxFontStyle enumeration values.

See also

GetStyle()

void wxFont::SetSymbolicSize (wxFontSymbolicSize *size*)

Sets the font size using a predefined symbolic size name.

This function allows to change font size to be (very) large or small compared to the standard font size.

See also

SetSymbolicSizeRelativeTo().

Since

2.9.2

void wxFont::SetSymbolicSizeRelativeTo (wxFontSymbolicSize *size*, int *base*)

Sets the font size compared to the base font size.

This is the same as SetSymbolicSize() except that it uses the given font size as the normal font size instead of the standard font size.

Since

2.9.2

virtual void wxFont::SetUnderlined (bool *underlined*) [virtual]

Sets underlining.

Parameters

underlined	true to underline, false otherwise.

See also

GetUnderlined()

virtual void wxFont::SetWeight (wxFontWeight *weight*) [virtual]

Sets the font weight.

Parameters

weight	One of the wxFontWeight values.

See also

GetWeight()

wxFont wxFont::Smaller ( ) const

Returns a smaller version of this font.

The font size is divided by `1.2`, the factor of `1.2` being inspired by the W3C CSS specification.

See also

MakeSmaller(), Larger(), Scaled()

Since

2.9.1

wxFont wxFont::Strikethrough ( ) const

Returns stricken-through version of this font.

Currently stricken-through fonts are only supported in wxMSW, wxGTK and OSX.

See also

MakeStrikethrough()

Since

2.9.4

wxFont wxFont::Underlined ( ) const

Returns underlined version of this font.

See also

MakeUnderlined()

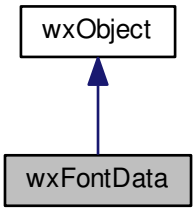
Since

2.9.2

21.261 wxFontData Class Reference

`#include <wx/fontdata.h>`

Inheritance diagram for wxFontData:



21.261.1 Detailed Description

This class holds a variety of information related to font dialogs.

Library: wxCore

Category: Common Dialogs, Data Structures

See also

wxFontDialog Overview, wxFont, wxFontDialog

Public Member Functions

- wxFontData ()

  Constructor.
- void EnableEffects (bool enable)

  Enables or disables "effects" under Windows or generic only.
- bool GetAllowSymbols () const

  Under Windows, returns a flag determining whether symbol fonts can be selected.
- wxFont GetChosenFont () const

  Gets the font chosen by the user if the user pressed OK (wxFontDialog::ShowModal() returned wxID_OK).
- const wxColour & GetColour () const

  Gets the colour associated with the font dialog.
- bool GetEnableEffects () const

  Determines whether "effects" are enabled under Windows.
- wxFont GetInitialFont () const

  Gets the font that will be initially used by the font dialog.
- bool GetShowHelp () const

  Returns true if the Help button will be shown (Windows only).
- void SetAllowSymbols (bool allowSymbols)

  Under Windows, determines whether symbol fonts can be selected.
- void SetChosenFont (const wxFont &font)

  Sets the font that will be returned to the user (for internal use only).
- void SetColour (const wxColour &colour)

  Sets the colour that will be used for the font foreground colour.
- void SetInitialFont (const wxFont &font)

  Sets the font that will be initially used by the font dialog.
- void SetRange (int min, int max)

  Sets the valid range for the font point size (Windows only).
- void SetShowHelp (bool showHelp)

  Determines whether the Help button will be displayed in the font dialog (Windows only).
- wxFontData & operator= (const wxFontData &data)

  Assignment operator for the font data.

Additional Inherited Members

21.261.2 Constructor & Destructor Documentation

wxFontData::wxFontData ( )

Constructor.

Initializes *fontColour* to black, *showHelp* to false, *allowSymbols* to true, *enableEffects* to true, *minSize* to 0 and *maxSize* to 0.

21.261.3 Member Function Documentation

void wxFontData::EnableEffects (bool *enable*)

Enables or disables "effects" under Windows or generic only.

This refers to the controls for manipulating colour, strikeout and underline properties.

The default value is true.

bool wxFontData::GetAllowSymbols ( ) const

Under Windows, returns a flag determining whether symbol fonts can be selected.

Has no effect on other platforms.

The default value is true.

wxFont wxFontData::GetChosenFont ( ) const

Gets the font chosen by the user if the user pressed OK (wxFontDialog::ShowModal() returned wxID_OK).

const wxColour& wxFontData::GetColour ( ) const

Gets the colour associated with the font dialog.

The default value is black.

bool wxFontData::GetEnableEffects ( ) const

Determines whether "effects" are enabled under Windows.

This refers to the controls for manipulating colour, strikeout and underline properties.

The default value is true.

wxFont wxFontData::GetInitialFont ( ) const

Gets the font that will be initially used by the font dialog.

This should have previously been set by the application.

bool wxFontData::GetShowHelp ( ) const

Returns true if the Help button will be shown (Windows only).

The default value is false.

wxFontData& wxFontData::operator= (const wxFontData & *data*)

Assignment operator for the font data.

void wxFontData::SetAllowSymbols (bool *allowSymbols*)

Under Windows, determines whether symbol fonts can be selected.

Has no effect on other platforms.

The default value is true.

void wxFontData::SetChosenFont (const wxFont & *font*)

Sets the font that will be returned to the user (for internal use only).

void wxFontData::SetColour (const wxColour & *colour*)

Sets the colour that will be used for the font foreground colour.
The default colour is black.

void wxFontData::SetInitialFont (const wxFont & *font*)

Sets the font that will be initially used by the font dialog.

void wxFontData::SetRange (int *min,* int *max*)

Sets the valid range for the font point size (Windows only).
The default is 0, 0 (unrestricted range).

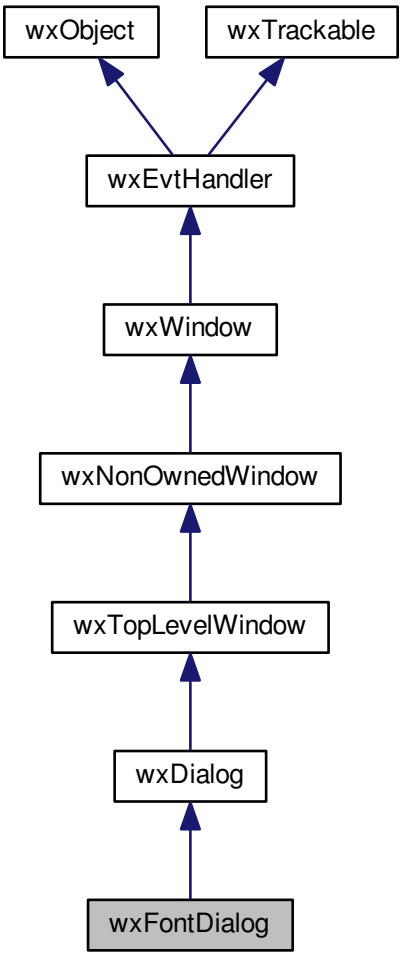
void wxFontData::SetShowHelp (bool *showHelp*)

Determines whether the Help button will be displayed in the font dialog (Windows only).
The default value is false.

21.262 wxFontDialog Class Reference

`#include <wx/fontdlg.h>`

Inheritance diagram for wxFontDialog:



21.262.1 Detailed Description

This class represents the font chooser dialog.

Library: wxCore

Category: Common Dialogs

See also

wxFontDialog Overview, wxFontData, wxGetFontFromUser()

Public Member Functions

- wxFontDialog ()

Default ctor.

- wxFontDialog (wxWindow *parent)

  Constructor with parent window.
- wxFontDialog (wxWindow *parent, const wxFontData &data)

  Constructor.
- bool Create (wxWindow *parent)

  Creates the dialog if the wxFontDialog object had been initialized using the default constructor.
- bool Create (wxWindow *parent, const wxFontData &data)

  Creates the dialog if the wxFontDialog object had been initialized using the default constructor.
- int ShowModal ()

  Shows the dialog, returning `wxID_OK` if the user pressed Ok, and `wxID_CANCEL` otherwise.
- const wxFontData & GetFontData () const

  Returns the font data associated with the font dialog.
- wxFontData & GetFontData ()

  Returns the font data associated with the font dialog.

Additional Inherited Members

21.262.2 Constructor & Destructor Documentation

wxFontDialog::wxFontDialog ( )

Default ctor.

Create() must be called before the dialog can be shown.

wxFontDialog::wxFontDialog (wxWindow * *parent*)

Constructor with parent window.

wxFontDialog::wxFontDialog (wxWindow * *parent,* const wxFontData & *data*)

Constructor.

Pass a parent window, and the font data object to be used to initialize the dialog controls.

21.262.3 Member Function Documentation

bool wxFontDialog::Create (wxWindow * *parent*)

Creates the dialog if the wxFontDialog object had been initialized using the default constructor.

Returns

true on success and false if an error occurred.

bool wxFontDialog::Create (wxWindow * *parent,* const wxFontData & *data*)

Creates the dialog if the wxFontDialog object had been initialized using the default constructor.

Returns

true on success and false if an error occurred.

const wxFontData& wxFontDialog::GetFontData ( ) const

Returns the font data associated with the font dialog.

wxFontData& wxFontDialog::GetFontData ( )

Returns the font data associated with the font dialog.

int wxFontDialog::ShowModal ( ) [virtual]

Shows the dialog, returning `wxID_OK` if the user pressed Ok, and `wxID_CANCEL` otherwise.

If the user cancels the dialog (ShowModal returns `wxID_CANCEL`), no font will be created. If the user presses OK, a new wxFont will be created and stored in the font dialog's wxFontData structure.

See also

GetFontData()

Reimplemented from wxDialog.

21.263 wxFontEnumerator Class Reference

`#include <wx/fontenum.h>`

21.263.1 Detailed Description

wxFontEnumerator enumerates either all available fonts on the system or only the ones with given attributes - either only fixed-width (suited for use in programs such as terminal emulators and the like) or the fonts available in the given encoding).

To do this, you just have to call one of EnumerateXXX() functions - either wxFontEnumerator::EnumerateFacenames() or wxFontEnumerator::EnumerateEncodings() and the corresponding callback (wxFontEnumerator::OnFacename() or wxFontEnumerator::OnFontEncoding()) will be called repeatedly until either all fonts satisfying the specified criteria are exhausted or the callback returns false.

21.263.2 Virtual functions to override

Either OnFacename or OnFontEncoding should be overridden depending on whether you plan to call EnumerateFacenames or EnumerateEncodings. Of course, if you call both of them, you should override both functions.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

Font Encodings, Font Sample, wxFont, wxFontMapper

Public Member Functions

- wxFontEnumerator ()
- virtual ~wxFontEnumerator ()
- virtual bool EnumerateEncodings (const wxString &font=wxEmptyString)

  Call OnFontEncoding() for each encoding supported by the given font - or for each encoding supported by at least some font if font is not specified.
- virtual bool EnumerateFacenames (wxFontEncoding encoding=wxFONTENCODING_SYSTEM, bool fixedWidthOnly=false)

  Call OnFacename() for each font which supports given encoding (only if it is not `wxFONTENCODING_SYSTEM`) and is of fixed width (if fixedWidthOnly is true).
- virtual bool OnFacename (const wxString &font)

  Called by EnumerateFacenames() for each match.
- virtual bool OnFontEncoding (const wxString &font, const wxString &encoding)

  Called by EnumerateEncodings() for each match.

Static Public Member Functions

- static wxArrayString GetEncodings (const wxString &facename=wxEmptyString)

  Return array of strings containing all encodings found by EnumerateEncodings().
- static wxArrayString GetFacenames (wxFontEncoding encoding=wxFONTENCODING_SYSTEM, bool fixedWidthOnly=false)

  Return array of strings containing all facenames found by EnumerateFacenames().
- static bool IsValidFacename (const wxString &facename)

  Returns true if the given string is valid face name, i.e.

21.263.3 Constructor & Destructor Documentation

wxFontEnumerator::wxFontEnumerator ( )

virtual wxFontEnumerator::~wxFontEnumerator ( ) `[virtual]`

21.263.4 Member Function Documentation

virtual bool wxFontEnumerator::EnumerateEncodings (const wxString & *font* = wxEmptyString) `[virtual]`

Call OnFontEncoding() for each encoding supported by the given font - or for each encoding supported by at least some font if *font* is not specified.

virtual bool wxFontEnumerator::EnumerateFacenames (wxFontEncoding *encoding* = wxFONTENCODING_SYSTEM, bool *fixedWidthOnly* = `false`) `[virtual]`

Call OnFacename() for each font which supports given encoding (only if it is not `wxFONTENCODING_SYSTEM`) and is of fixed width (if *fixedWidthOnly* is true).

Calling this function with default arguments will result in enumerating all fonts available on the system.

static wxArrayString wxFontEnumerator::GetEncodings (const wxString & *facename* = wxEmptyString) `[static]`

Return array of strings containing all encodings found by EnumerateEncodings().

static wxArrayString wxFontEnumerator::GetFacenames (wxFontEncoding *encoding* = wxFONTENCODING_SYSTEM, bool *fixedWidthOnly* = `false`) [static]

Return array of strings containing all facenames found by EnumerateFacenames().

static bool wxFontEnumerator::IsValidFacename (const wxString & *facename*) [static]

Returns true if the given string is valid face name, i.e.

it's the face name of an installed font and it can safely be used with wxFont::SetFaceName.

virtual bool wxFontEnumerator::OnFacename (const wxString & *font*) [virtual]

Called by EnumerateFacenames() for each match.

Return true to continue enumeration or false to stop it.

virtual bool wxFontEnumerator::OnFontEncoding (const wxString & *font*, const wxString & *encoding*) [virtual]

Called by EnumerateEncodings() for each match.

Return true to continue enumeration or false to stop it.

21.264 wxFontInfo Class Reference

`#include <wx/font.h>`

21.264.1 Detailed Description

This class is a helper used for wxFont creation using named parameter idiom: it allows to specify various wxFont attributes using the chained calls to its clearly named methods instead of passing them in the fixed order to wxFont constructors.

For example, to create an italic font with the given face name and size you could use:

```
wxFont font(wxFontInfo(12).FaceName("Helvetica").Italic());
```

Notice that all of the methods of this object return a reference to the object itself, allowing the calls to them to be chained as in the example above.

All methods taking boolean parameters can be used to turn the specified font attribute on or off and turn it on by default.

Since

2.9.5

Public Member Functions

- wxFontInfo ()

  Default constructor uses the default font size for the current platform.
- wxFontInfo (int pointSize)

  Constructor setting the font size in points to use.
- wxFontInfo (const wxSize &pixelSize)

  Constructor setting the font size in pixels to use.

- wxFontInfo & Family (wxFontFamily family)

  Set the font family.
- wxFontInfo & FaceName (const wxString &faceName)

  Set the font face name to use.
- wxFontInfo & Bold (bool bold=true)

  Use a bold version of the font.
- wxFontInfo & Light (bool light=true)

  Use a lighter version of the font.
- wxFontInfo & Italic (bool italic=true)

  Use an italic version of the font.
- wxFontInfo & Slant (bool slant=true)

  Use a slanted version of the font.
- wxFontInfo & AntiAliased (bool antiAliased=true)

  Set anti-aliasing flag.
- wxFontInfo & Underlined (bool underlined=true)

  Use an underlined version of the font.
- wxFontInfo & Strikethrough (bool strikethrough=true)

  Use a strike-through version of the font.
- wxFontInfo & Encoding (wxFontEncoding encoding)

  Set the font encoding to use.
- wxFontInfo & AllFlags (int flags)

  Set all the font attributes at once.

21.264.2 Constructor & Destructor Documentation

wxFontInfo::wxFontInfo ( )

Default constructor uses the default font size for the current platform.

wxFontInfo::wxFontInfo (int *pointSize*) `[explicit]`

Constructor setting the font size in points to use.

See also

wxFont::SetPointSize()

wxFontInfo::wxFontInfo (const wxSize & *pixelSize*) `[explicit]`

Constructor setting the font size in pixels to use.

See also

wxFont::SetPixelSize()

21.264.3 Member Function Documentation

wxFontInfo& wxFontInfo::AllFlags (int *flags*)

Set all the font attributes at once.

See wxFontFlag for the various flags that can be used.

wxFontInfo& wxFontInfo::AntiAliased (bool *antiAliased* = `true`)

Set anti-aliasing flag.

Force the use of anti-aliasing on or off.

Currently this is not implemented, i.e. using this method doesn't do anything.

wxFontInfo& wxFontInfo::Bold (bool *bold* = `true`)

Use a bold version of the font.

See also

wxFontWeight, wxFont::SetWeight()

wxFontInfo& wxFontInfo::Encoding (wxFontEncoding *encoding*)

Set the font encoding to use.

This is mostly unneeded in Unicode builds of wxWidgets.

See also

wxFontEncoding, wxFont::SetEncoding()

wxFontInfo& wxFontInfo::FaceName (const wxString & *faceName*)

Set the font face name to use.

Face names are not portable, so prefer to use Family() in portable code.

See also

wxFont::SetFaceName()

wxFontInfo& wxFontInfo::Family (wxFontFamily *family*)

Set the font family.

The family is a generic portable way of referring to fonts without specifying a precise face name. This parameter must be one of the wxFontFamily enumeration values.

If the FaceName() is used, then it overrides the font family.

See also

wxFont::SetFamily()

wxFontInfo& wxFontInfo::Italic (bool *italic* = `true`)

Use an italic version of the font.

See also

wxFontStyle, wxFont::SetStyle()

wxFontInfo& wxFontInfo::Light (bool *light* = `true`)

Use a lighter version of the font.

See also

wxFontWeight, wxFont::SetWeight()

wxFontInfo& wxFontInfo::Slant (bool *slant* = `true`)

Use a slanted version of the font.

See also

wxFontStyle, wxFont::SetStyle()

wxFontInfo& wxFontInfo::Strikethrough (bool *strikethrough* = `true`)

Use a strike-through version of the font.

Currently this is only implemented in wxMSW, wxGTK and OSX.

wxFontInfo& wxFontInfo::Underlined (bool *underlined* = `true`)

Use an underlined version of the font.

21.265 wxFontList Class Reference

`#include <wx/font.h>`

21.265.1 Detailed Description

A font list is a list containing all fonts which have been created.

There is only one instance of this class: wxTheFontList.

Use this object to search for a previously created font of the desired type and create it if not already found.

In some windowing systems, the font may be a scarce resource, so it is best to reuse old resources if possible. When an application finishes, all fonts will be deleted and their resources freed, eliminating the possibility of 'memory leaks'.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxFont

Public Member Functions

- wxFontList ()

  Constructor.
- wxFont * FindOrCreateFont (int point_size, wxFontFamily family, wxFontStyle style, wxFontWeight weight, bool underline=false, const wxString &facename=wxEmptyString, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  Finds a font of the given specification, or creates one and adds it to the list.

21.265.2 Constructor & Destructor Documentation

wxFontList::wxFontList ( )

Constructor.

The application should not construct its own font list: use the object pointer wxTheFontList.

21.265.3 Member Function Documentation

wxFont* wxFontList::FindOrCreateFont (int *point_size,* wxFontFamily *family,* wxFontStyle *style,* wxFontWeight *weight,* bool *underline* = `false`, const wxString & *facename* = wxEmptyString, wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT)

Finds a font of the given specification, or creates one and adds it to the list.

See the wxFont constructor for details of the arguments.

21.266 wxFontMapper Class Reference

```
#include <wx/fontmap.h>
```

21.266.1 Detailed Description

wxFontMapper manages user-definable correspondence between logical font names and the fonts present on the machine.

The default implementations of all functions will ask the user if they are not capable of finding the answer themselves and store the answer in a config file (configurable via SetConfigXXX functions). This behaviour may be disabled by giving the value of false to "interactive" parameter.

However, the functions will always consult the config file to allow the user-defined values override the default logic and there is no way to disable this - which shouldn't be ever needed because if "interactive" was never true, the config file is never created anyhow.

In case everything else fails (i.e. there is no record in config file and "interactive" is false or user denied to choose any replacement), the class queries wxEncodingConverter for "equivalent" encodings (e.g. iso8859-2 and cp1250) and tries them.

21.266.2 Using wxFontMapper in conjunction with wxMBConv classes

If you need to display text in encoding which is not available at host system (see wxFontMapper::IsEncodingAvailable), you may use these two classes to find font in some similar encoding (see wxFontMapper::GetAltForEncoding) and convert the text to this encoding (wxMBConv classes). Following code snippet demonstrates it:

```
if (!wxFontMapper::Get()->IsEncodingAvailable(enc, facename))
{
    wxFontEncoding alternative;
    if (wxFontMapper::Get()->GetAltForEncoding(enc, &alternative,
                                               facename, false))
    {
        wxCSConv convFrom(wxFontMapper::Get()->
      GetEncodingName(enc));
        wxCSConv convTo(wxFontMapper::Get()->
      GetEncodingName(alternative));
        text = wxString(text.mb_str(convFrom), convTo);
    }
    else
        ...failure (or we may try iso8859-1/7bit ASCII)...
}
...display text...
```

Library: wxCore

Category: Application and System configuration

See also

wxEncodingConverter, Writing Non-English Applications

Public Member Functions

- wxFontMapper ()

  Default ctor.
- virtual ~wxFontMapper ()

  Virtual dtor.
- virtual wxFontEncoding CharsetToEncoding (const wxString &charset, bool interactive=true)

  Returns the encoding for the given charset (in the form of RFC 2046) or `wxFONTENCODING_SYSTEM` *if couldn't decode it.*
- virtual bool IsEncodingAvailable (wxFontEncoding encoding, const wxString &facename=wxEmptyString)

  Check whether given encoding is available in given face or not.
- void SetConfigPath (const wxString &prefix)

  Set the root config path to use (should be an absolute path).
- void SetDialogParent (wxWindow *parent)

  The parent window for modal dialogs.
- void SetDialogTitle (const wxString &title)

  The title for the dialogs (note that default is quite reasonable).

- bool GetAltForEncoding (wxFontEncoding encoding, wxNativeEncodingInfo *info, const wxString &facename=wxEmptyString, bool interactive=true)

  Find an alternative for the given encoding (which is supposed to not be available on this system).
- bool GetAltForEncoding (wxFontEncoding encoding, wxFontEncoding *alt_encoding, const wxString &facename=wxEmptyString, bool interactive=true)

  Find an alternative for the given encoding (which is supposed to not be available on this system).

Static Public Member Functions

- static wxFontMapper * Get ()

  Get the current font mapper object.
- static const wxChar ** GetAllEncodingNames (wxFontEncoding encoding)

  Returns the array of all possible names for the given encoding.

- static wxFontEncoding GetEncoding (size_t n)

  Returns the n-th supported encoding.
- static wxString GetEncodingDescription (wxFontEncoding encoding)

  Return user-readable string describing the given encoding.
- static wxFontEncoding GetEncodingFromName (const wxString &encoding)

  Return the encoding corresponding to the given internal name.
- static wxString GetEncodingName (wxFontEncoding encoding)

  Return internal string identifier for the encoding (see also wxFontMapper::GetEncodingDescription).
- static size_t GetSupportedEncodingsCount ()

  Returns the number of the font encodings supported by this class.
- static wxFontMapper * Set (wxFontMapper *mapper)

  Set the current font mapper object and return previous one (may be NULL).

21.266.3 Constructor & Destructor Documentation

wxFontMapper::wxFontMapper ( )

Default ctor.

Note

The preferred way of creating a wxFontMapper instance is to call wxFontMapper::Get().

virtual wxFontMapper::~wxFontMapper ( ) [virtual]

Virtual dtor.

21.266.4 Member Function Documentation

virtual wxFontEncoding wxFontMapper::CharsetToEncoding (const wxString & *charset*, bool *interactive* = true) [virtual]

Returns the encoding for the given charset (in the form of RFC 2046) or wxFONTENCODING_SYSTEM if couldn't decode it.

Be careful when using this function with *interactive* set to true (default value) as the function then may show a dialog box to the user which may lead to unexpected reentrancies and may also take a significantly longer time than a simple function call. For these reasons, it is almost always a bad idea to call this function from the event handlers for repeatedly generated events such as EVT_PAINT.

static wxFontMapper* wxFontMapper::Get ( ) [static]

Get the current font mapper object.

If there is no current object, creates one.

See also

Set()

static const wxChar wxFontMapper::GetAllEncodingNames (wxFontEncoding *encoding*) [static]**

Returns the array of all possible names for the given encoding.

The array is NULL-terminated. IF it isn't empty, the first name in it is the canonical encoding name, i.e. the same string as returned by GetEncodingName().

bool wxFontMapper::GetAltForEncoding (wxFontEncoding *encoding*, wxNativeEncodingInfo * *info*, const wxString & *facename* = wxEmptyString, bool *interactive* = true)

Find an alternative for the given encoding (which is supposed to not be available on this system).

If successful, return true and fill info structure with the parameters required to create the font, otherwise return false.

The first form is for wxWidgets' internal use while the second one is better suitable for general use – it returns wxFontEncoding which can consequently be passed to wxFont constructor.

bool wxFontMapper::GetAltForEncoding (wxFontEncoding *encoding*, wxFontEncoding * *alt_encoding*, const wxString & *facename* = wxEmptyString, bool *interactive* = true)

Find an alternative for the given encoding (which is supposed to not be available on this system).

If successful, return true and fill info structure with the parameters required to create the font, otherwise return false.

The first form is for wxWidgets' internal use while the second one is better suitable for general use – it returns wxFontEncoding which can consequently be passed to wxFont constructor.

static wxFontEncoding wxFontMapper::GetEncoding (size_t *n*) [static]

Returns the *n-th* supported encoding.

Together with GetSupportedEncodingsCount() this method may be used to get all supported encodings.

static wxString wxFontMapper::GetEncodingDescription (wxFontEncoding *encoding*) [static]

Return user-readable string describing the given encoding.

static wxFontEncoding wxFontMapper::GetEncodingFromName (const wxString & *encoding*) [static]

Return the encoding corresponding to the given internal name.

This function is the inverse of GetEncodingName() and is intentionally less general than CharsetToEncoding(), i.e. it doesn't try to make any guesses nor ever asks the user. It is meant just as a way of restoring objects previously serialized using GetEncodingName().

static wxString wxFontMapper::GetEncodingName (wxFontEncoding *encoding*) [static]

Return internal string identifier for the encoding (see also wxFontMapper::GetEncodingDescription).

See also

GetEncodingFromName()

static size_t wxFontMapper::GetSupportedEncodingsCount ( ) [static]

Returns the number of the font encodings supported by this class.

Together with GetEncoding() this method may be used to get all supported encodings.

virtual bool wxFontMapper::IsEncodingAvailable (wxFontEncoding *encoding*, const wxString & *facename* = wxEmptyString) [virtual]

Check whether given encoding is available in given face or not.

If no facename is given, find *any* font in this encoding.

static wxFontMapper* wxFontMapper::Set (wxFontMapper * *mapper*) [static]

Set the current font mapper object and return previous one (may be NULL).

This method is only useful if you want to plug-in an alternative font mapper into wxWidgets.

See also

Get()

void wxFontMapper::SetConfigPath (const wxString & *prefix*)

Set the root config path to use (should be an absolute path).

void wxFontMapper::SetDialogParent (wxWindow * *parent*)

The parent window for modal dialogs.

void wxFontMapper::SetDialogTitle (const wxString & *title*)

The title for the dialogs (note that default is quite reasonable).

21.267 wxFontMetrics Struct Reference

`#include <wx/dc.h>`

21.267.1 Detailed Description

Simple collection of various font metrics.

This object is returned by wxDC::GetFontMetrics().

Since

2.9.2

Library: wxCore

Category: Device Contexts, Graphics Device Interface (GDI)

Public Member Functions

- wxFontMetrics ()

  Constructor initializes all fields to 0.

Public Attributes

- int height

  Total character height.
- int ascent

  Part of the height above the baseline.
- int descent

  Part of the height below the baseline.
- int internalLeading

  Intra-line spacing.
- int externalLeading

  Inter-line spacing.
- int averageWidth

  Average font width, a.k.a. "x-width".

21.267.2 Constructor & Destructor Documentation

wxFontMetrics::wxFontMetrics ( )

Constructor initializes all fields to 0.

21.267.3 Member Data Documentation

int wxFontMetrics::ascent

Part of the height above the baseline.

int wxFontMetrics::averageWidth

Average font width, a.k.a. "x-width".

int wxFontMetrics::descent

Part of the height below the baseline.

int wxFontMetrics::externalLeading

Inter-line spacing.

int wxFontMetrics::height

Total character height.

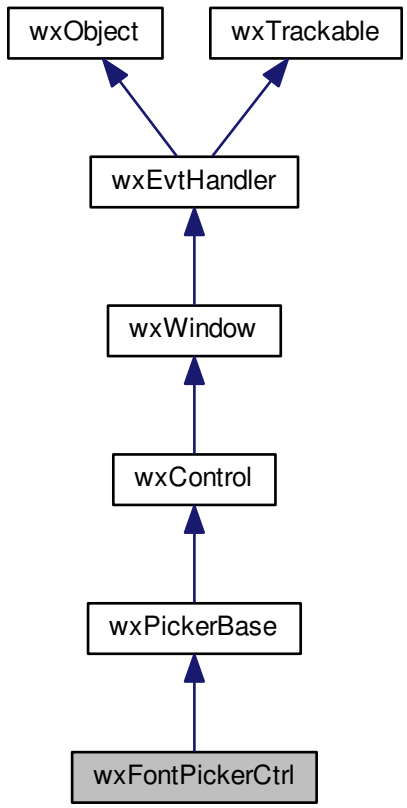
int wxFontMetrics::internalLeading

Intra-line spacing.

21.268 wxFontPickerCtrl Class Reference

`#include <wx/fontpicker.h>`

Inheritance diagram for wxFontPickerCtrl:



21.268.1 Detailed Description

This control allows the user to select a font.

The generic implementation is a button which brings up a wxFontDialog when clicked. Native implementation may differ but this is usually a (small) widget which give access to the font-chooser dialog. It is only available if `wxUSE_FONTPICKERCTRL` is set to 1 (the default).

Styles

This class supports the following styles:

- wxFNTP_DEFAULT_STYLE: The default style: wxFNTP_FONTDESC_AS_LABEL | wxFNTP_USEFONT_FOR_LABEL.
- wxFNTP_USE_TEXTCTRL: Creates a text control to the left of the picker button which is completely managed by the wxFontPickerCtrl and which can be used by the user to specify a font (see SetSelectedFont). The text

control is automatically synchronized with button's value. Use functions defined in wxPickerBase to modify the text control.

- wxFNTP_FONTDESC_AS_LABEL: Keeps the label of the button updated with the fontface name and the font size. E.g. choosing "Times New Roman bold, italic with size 10" from the fontdialog, will update the label (overwriting any previous label) with the "Times New Roman, 10" text.
- wxFNTP_USEFONT_FOR_LABEL: Uses the currently selected font to draw the label of the button.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFontPickerEvent& event)

Event macros for events emitted by this class:

- EVT_FONTPICKER_CHANGED(id, func): The user changed the font selected in the control either using the button or using text control (see wxFNTP_USE_TEXTCTRL; note that in this case the event is fired only if the user's input is valid, i.e. recognizable).

Library: wxCore

Category: Picker Controls

See also

wxFontDialog, wxFontPickerEvent

Public Member Functions

- wxFontPickerCtrl ()
- wxFontPickerCtrl (wxWindow *parent, wxWindowID id, const wxFont &font=wxNullFont, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxFNTP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxFontPickerCtrlNameStr)

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxFont &font=wxNullFont, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxFNTP_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxFontPickerCtrlNameStr)

  Creates this widget with given parameters.
- unsigned int GetMaxPointSize () const

  Returns the maximum point size value allowed for the user-chosen font.
- wxColour GetSelectedColour () const

  Returns the currently selected colour.
- wxFont GetSelectedFont () const

  Returns the currently selected font.
- void SetMaxPointSize (unsigned int max)

  Sets the maximum point size value allowed for the user-chosen font.
- void SetSelectedColour (const wxColour &colour)

  Sets the font colour.
- void SetSelectedFont (const wxFont &font)

  Sets the currently selected font.

Additional Inherited Members

21.268.2 Constructor & Destructor Documentation

wxFontPickerCtrl::wxFontPickerCtrl ( )

wxFontPickerCtrl::wxFontPickerCtrl (wxWindow * *parent*, wxWindowID *id*, const wxFont & *font* = wxNullFont, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxFNTP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxFontPickerCtrlNameStr`)

Initializes the object and calls Create() with all the parameters.

21.268.3 Member Function Documentation

bool wxFontPickerCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxFont & *font* = wxNullFont, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxFNTP_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxFontPickerCtrlNameStr`)

Creates this widget with given parameters.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
font	The initial font shown in the control. If wxNullFont is given, the default font is used.
pos	Initial position.
size	Initial size.
style	The window style, see wxFNTP_* flags.
validator	Validator which can be used for additional date checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

unsigned int wxFontPickerCtrl::GetMaxPointSize ( ) const

Returns the maximum point size value allowed for the user-chosen font.

wxColour wxFontPickerCtrl::GetSelectedColour ( ) const

Returns the currently selected colour.

Note that the colour of the font can only be set by the user under Windows currently, elsewhere this method simply returns the colour previously set by SetSelectedColour() or black if it hadn't been called.

Since

3.1.0

wxFont wxFontPickerCtrl::GetSelectedFont ( ) const

Returns the currently selected font.

Note that this function is completely different from wxWindow::GetFont.

void wxFontPickerCtrl::SetMaxPointSize (unsigned int *max*)

Sets the maximum point size value allowed for the user-chosen font.

The default value is 100. Note that big fonts can require a lot of memory and CPU time both for creation and for rendering; thus, specially because the user has the option to specify the fontsize through a text control (see wxF↩NTP_USE_TEXTCTRL), it's a good idea to put a limit to the maximum font size when huge fonts do not make much sense.

void wxFontPickerCtrl::SetSelectedColour (const wxColour & *colour*)

Sets the font colour.

The font colour is actually only used under Windows currently, but this function is available under all platforms for consistency.

Since

3.1.0

void wxFontPickerCtrl::SetSelectedFont (const wxFont & *font*)

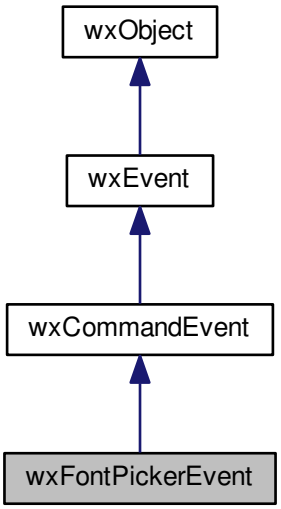
Sets the currently selected font.

Note that this function is completely different from wxWindow::SetFont.

21.269 wxFontPickerEvent Class Reference

`#include <wx/fontpicker.h>`

Inheritance diagram for wxFontPickerEvent:



21.269.1 Detailed Description

This event class is used for the events generated by wxFontPickerCtrl.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxFontPickerEvent& event)

Event macros:

- EVT_FONTPICKER_CHANGED(id, func): Generated whenever the selected font changes.

Library: wxCore

Category: Events

See also

wxFontPickerCtrl

Public Member Functions

- wxFontPickerEvent (wxObject *generator, int id, const wxFont &font)

  The constructor is not normally used by the user code.
- wxFont GetFont () const

  Retrieve the font the user has just selected.
- void SetFont (const wxFont &f)

  Set the font associated with the event.

Additional Inherited Members

21.269.2 Constructor & Destructor Documentation

wxFontPickerEvent::wxFontPickerEvent (wxObject * *generator,* int *id,* const wxFont & *font*)

The constructor is not normally used by the user code.

21.269.3 Member Function Documentation

wxFont wxFontPickerEvent::GetFont ( ) const

Retrieve the font the user has just selected.

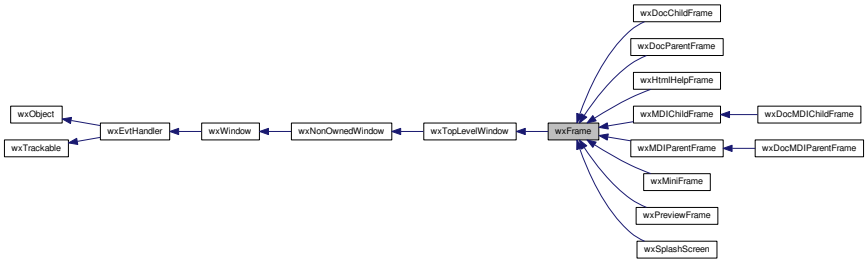
void wxFontPickerEvent::SetFont (const wxFont & *f*)

Set the font associated with the event.

21.270 wxFrame Class Reference

`#include <wx/frame.h>`

Inheritance diagram for wxFrame:



21.270.1 Detailed Description

A frame is a window whose size and position can (usually) be changed by the user.

It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any window that is not a frame or dialog.

A frame that has a status bar and toolbar, created via the CreateStatusBar() and CreateToolBar() functions, manages these windows and adjusts the value returned by GetClientSize() to reflect the remaining size available to application windows.

Remarks

An application should normally define an wxCloseEvent handler for the frame to respond to system close events, for example so that related data and subwindows can be cleaned up.

21.270.2 Default event processing

wxFrame processes the following events:

- `wxEVT_SIZE`: if the frame has exactly one child window, not counting the status and toolbar, this child is resized to take the entire frame client area. If two or more windows are present, they should be laid out explicitly either by manually handling `wxEVT_SIZE` or using sizers;
- `wxEVT_MENU_HIGHLIGHT`: the default implementation displays the help string associated with the selected item in the first pane of the status bar, if there is one.

Styles

This class supports the following styles:

- wxDEFAULT_FRAME_STYLE: Defined as wxMINIMIZE_BOX | wxMAXIMIZE_BOX | wxRESIZE_BORDER | wxSYSTEM_MENU | wxCAPTION | wxCLOSE_BOX | wxCLIP_CHILDREN.
- wxICONIZE: Display the frame iconized (minimized). Windows only.
- wxCAPTION: Puts a caption on the frame. Notice that this flag is required by wxMINIMIZE_BOX, wxMA↩ XIMIZE_BOX and wxCLOSE_BOX on most systems as the corresponding buttons cannot be shown if the window has no title bar at all. I.e. if wxCAPTION is not specified those styles would be simply ignored.

- wxMINIMIZE: Identical to wxICONIZE. Windows only.
- wxMINIMIZE_BOX: Displays a minimize box on the frame.
- wxMAXIMIZE: Displays the frame maximized. Windows and GTK+ only.
- wxMAXIMIZE_BOX: Displays a maximize box on the frame. Notice that under wxGTK wxRESIZE_BORDER must be used as well or this style is ignored.
- wxCLOSE_BOX: Displays a close box on the frame.
- wxSTAY_ON_TOP: Stay on top of all other windows, see also wxFRAME_FLOAT_ON_PARENT.
- wxSYSTEM_MENU: Displays a system menu containing the list of various windows commands in the window title bar. Unlike wxMINIMIZE_BOX, wxMAXIMIZE_BOX and wxCLOSE_BOX styles this style can be used without wxCAPTION, at least under Windows, and makes the system menu available without showing it on screen in this case. However it is recommended to only use it together with wxCAPTION for consistent behaviour under all platforms.
- wxRESIZE_BORDER: Displays a resizable border around the window.
- wxFRAME_TOOL_WINDOW: Causes a frame with a small title bar to be created; the frame does not appear in the taskbar under Windows or GTK+.
- wxFRAME_NO_TASKBAR: Creates an otherwise normal frame but it does not appear in the taskbar under Windows or GTK+ (note that it will minimize to the desktop window under Windows which may seem strange to the users and thus it might be better to use this style only without wxMINIMIZE_BOX style). In wxGTK, the flag is respected only if the window manager supports _NET_WM_STATE_SKIP_TASKBAR hint.
- wxFRAME_FLOAT_ON_PARENT: The frame will always be on top of its parent (unlike wxSTAY_ON_TOP). A frame created with this style must have a non-NULL parent.
- wxFRAME_SHAPED: Windows with this style are allowed to have their shape changed with the SetShape() method.

The default frame style is for normal, resizable frames. To create a frame which cannot be resized by user, you may use the following combination of styles:

```
wxDEFAULT_FRAME_STYLE & ~(wxRESIZE_BORDER |
      wxMAXIMIZE_BOX)
```

See also the Window Styles.

Extra Styles

This class supports the following extra styles:

- wxFRAME_EX_CONTEXTHELP: Under Windows, puts a query button on the caption. When pressed, Windows will go into a context-sensitive help mode and wxWidgets will send a `wxEVT_HELP` event if the user clicked on an application window. Note that this is an extended style and must be set by calling SetExtraStyle before Create is called (two-step construction). You cannot use this style together with wxMAXIMIZE_BOX or wxMINIMIZE_BOX, so you should use wxDEFAULT_FRAME_STYLE ~ (wxMINIMIZE_BOX | wxMAXI↩ MIZE_BOX) for the frames having this style (the dialogs don't have a minimize or a maximize box by default)
- wxFRAME_EX_METAL: On Mac OS X, frames with this style will be shown with a metallic look. This is an extra style.

Events emitted by this class

Event macros for events emitted by this class:

- EVT_CLOSE(func): Process a `wxEVT_CLOSE_WINDOW` event when the frame is being closed by the user or programmatically (see wxWindow::Close). The user may generate this event clicking the close button (typically the 'X' on the top-right of the title bar) if it's present (see the `wxCLOSE_BOX` style). See wxCloseEvent.
- EVT_ICONIZE(func): Process a `wxEVT_ICONIZE` event. See wxIconizeEvent.
- EVT_MENU_OPEN(func): A menu is about to be opened. See wxMenuEvent.
- EVT_MENU_CLOSE(func): A menu has been just closed. See wxMenuEvent.
- EVT_MENU_HIGHLIGHT(id, func): The menu item with the specified id has been highlighted: used to show help prompts in the status bar by wxFrame. See wxMenuEvent.
- EVT_MENU_HIGHLIGHT_ALL(func): A menu item has been highlighted, i.e. the currently selected menu item has changed. See wxMenuEvent.

Library: wxCore

Category: Managed Windows

See also

wxMDIParentFrame, wxMDIChildFrame, wxMiniFrame, wxDialog

Public Member Functions

- wxFrame ()

  Default constructor.
- wxFrame (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor, creating the window.
- virtual ~wxFrame ()

  Destructor.
- void Centre (int direction=wxBOTH)

  Centres the frame on the display.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Used in two-step frame construction.
- virtual wxStatusBar * CreateStatusBar (int number=1, long style=wxSTB_DEFAULT_STYLE, wxWindowID id=0, const wxString &name=wxStatusBarNameStr)

  Creates a status bar at the bottom of the frame.
- virtual wxToolBar * CreateToolBar (long style=wxTB_DEFAULT_STYLE, wxWindowID id=wxID_ANY, const wxString &name=wxToolBarNameStr)

  Creates a toolbar at the top or left of the frame.
- virtual void DoGiveHelp (const wxString &text, bool show)

  Method used to show help string of the selected menu toolbar item.
- virtual wxPoint GetClientAreaOrigin () const

Returns the origin of the frame client area (in client coordinates).

- virtual wxMenuBar * GetMenuBar () const

  Returns a pointer to the menubar currently associated with the frame (if any).
- virtual wxStatusBar * GetStatusBar () const

  Returns a pointer to the status bar currently associated with the frame (if any).
- int GetStatusBarPane () const

  Returns the status bar pane used to display menu and toolbar help.
- virtual wxToolBar * GetToolBar () const

  Returns a pointer to the toolbar currently associated with the frame (if any).
- virtual wxStatusBar * OnCreateStatusBar (int number, long style, wxWindowID id, const wxString &name)

  Virtual function called when a status bar is requested by CreateStatusBar().
- virtual wxToolBar * OnCreateToolBar (long style, wxWindowID id, const wxString &name)

  Virtual function called when a toolbar is requested by CreateToolBar().
- bool ProcessCommand (int id)

  Simulate a menu command.
- virtual void SetMenuBar (wxMenuBar *menuBar)

  Tells the frame to show the given menu bar.
- virtual void SetStatusBar (wxStatusBar *statusBar)

  Associates a status bar with the frame.
- void SetStatusBarPane (int n)

  Set the status bar pane used to display menu and toolbar help.
- virtual void SetStatusText (const wxString &text, int number=0)

  Sets the status bar text and redraws the status bar.
- virtual void SetStatusWidths (int n, const int *widths_field)

  Sets the widths of the fields in the status bar.
- virtual void SetToolBar (wxToolBar *toolBar)

  Associates a toolbar with the frame.
- wxTaskBarButton * MSWGetTaskBarButton ()

  MSW-specific function for accessing the taskbar button under Windows 7 or later.
- void PushStatusText (const wxString &text, int number=0)
- void PopStatusText (int number=0)

Additional Inherited Members

21.270.3 Constructor & Destructor Documentation

wxFrame::wxFrame ( )

Default constructor.

wxFrame::wxFrame (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor, creating the window.

Parameters

parent	The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.
id	The window identifier. It may take a value of -1 to indicate a default value.
title	The caption to be displayed on the frame's title bar.
pos	The window position. The value wxDefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.
size	The window size. The value wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.
style	The window style. See wxFrame class description.
name	The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

For Motif, MWM (the Motif Window Manager) should be running for any window styles to work (otherwise all styles take effect).

See also

Create()

virtual wxFrame::~wxFrame ( ) [virtual]

Destructor.

Destroys all child windows and menu bar if present.

See Window Deletion for more info.

21.270.4 Member Function Documentation

void wxFrame::Centre (int *direction* = wxBOTH)

Centres the frame on the display.

Parameters

direction	The parameter may be wxHORIZONTAL, wxVERTICAL or wxBOTH.

bool wxFrame::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = wxFrameNameStr)

Used in two-step frame construction.

See wxFrame() for further details.

virtual wxStatusBar* wxFrame::CreateStatusBar (int *number* = 1, long *style* = wxSTB_DEFAULT_STYLE, wxWindowID *id* = 0, const wxString & *name* = wxStatusBarNameStr) [virtual]

Creates a status bar at the bottom of the frame.

Parameters

number	The number of fields to create. Specify a value greater than 1 to create a multi-field status bar.
style	The status bar style. See wxStatusBar for a list of valid styles.
id	The status bar window identifier. If -1, an identifier will be chosen by wxWidgets.
name	The status bar window name.

Returns

A pointer to the status bar if it was created successfully, NULL otherwise.

Remarks

The width of the status bar is the whole width of the frame (adjusted automatically when resizing), and the height and text size are chosen by the host windowing system.

See also

SetStatusText(), OnCreateStatusBar(), GetStatusBar()

virtual wxToolBar* wxFrame::CreateToolBar (long *style* = wxTB_DEFAULT_STYLE, wxWindowID *id* = wxID_ANY, const wxString & *name* = wxToolBarNameStr) [virtual]

Creates a toolbar at the top or left of the frame.

Parameters

style	The toolbar style. See wxToolBar for a list of valid styles.
id	The toolbar window identifier. If -1, an identifier will be chosen by wxWidgets.
name	The toolbar window name.

Returns

A pointer to the toolbar if it was created successfully, NULL otherwise.

Remarks

By default, the toolbar is an instance of wxToolBar. To use a different class, override OnCreateToolBar(). When a toolbar has been created with this function, or made known to the frame with wxFrame::SetToolBar(), the frame will manage the toolbar position and adjust the return value from wxWindow::GetClientSize() to reflect the available space for application windows. Under Pocket PC, you should always use this function for creating the toolbar to be managed by the frame, so that wxWidgets can use a combined menubar and toolbar. Where you manage your own toolbars, create a wxToolBar as usual.

See also

CreateStatusBar(), OnCreateToolBar(), SetToolBar(), GetToolBar()

virtual void wxFrame::DoGiveHelp (const wxString & *text*, bool *show*) [virtual]

Method used to show help string of the selected menu toolbar item.

This method is called by the default wxEVT_MENU_HIGHLIGHT event handler and also by wxToolBar to show the optional help string associated with the selected menu or toolbar item. It can be overridden if the default behaviour of showing this string in the frame status bar is not appropriate.

Parameters

text	The help string to show, may be empty. The default implementation simply shows this string in the frame status bar (after remembering its previous text to restore it later).
show	Whether the help should be shown or hidden. The default implementation restores the previously saved status bar text when it is false.

See also

SetStatusBarPane()

virtual wxPoint wxFrame::GetClientAreaOrigin ( ) const `[virtual]`

Returns the origin of the frame client area (in client coordinates).

It may be different from (0, 0) if the frame has a toolbar.

Reimplemented from wxWindow.

virtual wxMenuBar* wxFrame::GetMenuBar ( ) const `[virtual]`

Returns a pointer to the menubar currently associated with the frame (if any).

See also

SetMenuBar(), wxMenuBar, wxMenu

virtual wxStatusBar* wxFrame::GetStatusBar ( ) const `[virtual]`

Returns a pointer to the status bar currently associated with the frame (if any).

See also

CreateStatusBar(), wxStatusBar

int wxFrame::GetStatusBarPane ( ) const

Returns the status bar pane used to display menu and toolbar help.

See also

SetStatusBarPane()

virtual wxToolBar* wxFrame::GetToolBar ( ) const `[virtual]`

Returns a pointer to the toolbar currently associated with the frame (if any).

See also

CreateToolBar(), wxToolBar, SetToolBar()

wxTaskBarButton* wxFrame::MSWGetTaskBarButton ( )

MSW-specific function for accessing the taskbar button under Windows 7 or later.

Returns a wxTaskBarButton pointer representing the taskbar button of the window under Windows 7 or later. The returned wxTaskBarButton may be used, if non-NULL, to access the functionality including thumbnail representations, thumbnail toolbars, notification and status overlays, and progress indicators.

The returned pointer must *not* be deleted, it is owned by the frame and will be only deleted when the frame itself is destroyed.

This function is not available in the other ports by design, any occurrences of it in the portable code must be guarded by

```
#ifdef __WXMSW__
```

preprocessor guards.

Since

3.1.0

virtual wxStatusBar* wxFrame::OnCreateStatusBar (int *number,* long *style,* wxWindowID *id,* const wxString & *name*) [virtual]

Virtual function called when a status bar is requested by CreateStatusBar().

Parameters

number	The number of fields to create.
style	The window style. See wxStatusBar for a list of valid styles.
id	The window identifier. If -1, an identifier will be chosen by wxWidgets.
name	The window name.

Returns

A status bar object.

Remarks

An application can override this function to return a different kind of status bar. The default implementation returns an instance of wxStatusBar.

See also

CreateStatusBar(), wxStatusBar.

virtual wxToolBar* wxFrame::OnCreateToolBar (long *style,* wxWindowID *id,* const wxString & *name*) [virtual]

Virtual function called when a toolbar is requested by CreateToolBar().

Parameters

style	The toolbar style. See wxToolBar for a list of valid styles.

id	The toolbar window identifier. If -1, an identifier will be chosen by wxWidgets.
name	The toolbar window name.

Returns

A toolbar object.

Remarks

An application can override this function to return a different kind of toolbar. The default implementation returns an instance of wxToolBar.

See also

CreateToolBar(), wxToolBar.

void wxFrame::PopStatusText (int *number* = 0)

bool wxFrame::ProcessCommand (int *id*)

Simulate a menu command.

Parameters

id	The identifier for a menu item.

void wxFrame::PushStatusText (const wxString & *text*, int *number* = 0)

virtual void wxFrame::SetMenuBar (wxMenuBar * *menuBar*) [virtual]

Tells the frame to show the given menu bar.

Parameters

menuBar	The menu bar to associate with the frame.

Remarks

If the frame is destroyed, the menu bar and its menus will be destroyed also, so do not delete the menu bar explicitly (except by resetting the frame's menu bar to another frame or NULL). Under Windows, a size event is generated, so be sure to initialize data members properly before calling SetMenuBar(). Note that on some platforms, it is not possible to call this function twice for the same frame object.

See also

GetMenuBar(), wxMenuBar, wxMenu.

virtual void wxFrame::SetStatusBar (wxStatusBar * *statusBar*) [virtual]

Associates a status bar with the frame.

If *statusBar* is NULL, then the status bar, if present, is detached from the frame, but *not* deleted.

See also

CreateStatusBar(), wxStatusBar, GetStatusBar()

void wxFrame::SetStatusBarPane (int *n*)

Set the status bar pane used to display menu and toolbar help.

Using -1 disables help display.

virtual void wxFrame::SetStatusText (const wxString & *text*, int *number* = 0) [virtual]

Sets the status bar text and redraws the status bar.

Parameters

text	The text for the status field.
number	The status field (starting from zero).

Remarks

Use an empty string to clear the status bar.

See also

CreateStatusBar(), wxStatusBar

virtual void wxFrame::SetStatusWidths (int *n*, const int * *widths_field*) [virtual]

Sets the widths of the fields in the status bar.

Parameters

n	The number of fields in the status bar. It must be the same used in CreateStatusBar.
widths_field	Must contain an array of n integers, each of which is a status field width in pixels. A value of -1 indicates that the field is variable width; at least one field must be -1. You should delete this array after calling SetStatusWidths().

Remarks

The widths of the variable fields are calculated from the total width of all fields, minus the sum of widths of the non-variable fields, divided by the number of variable fields.

wxPerl Note: In wxPerl this method takes the field widths as parameters.

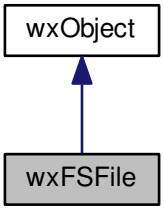
virtual void wxFrame::SetToolBar (wxToolBar * *toolBar*) [virtual]

Associates a toolbar with the frame.

21.271 wxFSFile Class Reference

`#include <wx/filesys.h>`

Inheritance diagram for wxFSFile:



21.271.1 Detailed Description

This class represents a single file opened by wxFileSystem.

It provides more information than wxWidgets' input streams (stream, filename, mime type, anchor).

Note

Any pointer returned by a method of wxFSFile is valid only as long as the wxFSFile object exists. For example a call to GetStream() doesn't *create* the stream but only returns the pointer to it. In other words after 10 calls to GetStream() you will have obtained ten identical pointers.

Library: wxBase

Category: Virtual File System, File Handling

See also

wxFileSystemHandler, wxFileSystem, wxFileSystem Overview

Public Member Functions

- wxFSFile (wxInputStream *stream, const wxString &location, const wxString &mimetype, const wxString &anchor, wxDateTime modif)

  Constructor.
- wxInputStream * DetachStream ()

  Detaches the stream from the wxFSFile object.
- const wxString & GetAnchor () const

  Returns anchor (if present).
- const wxString & GetLocation () const

  Returns full location of the file, including path and protocol.
- const wxString & GetMimeType () const

  Returns the MIME type of the content of this file.
- wxDateTime GetModificationTime () const

  Returns time when this file was modified.
- wxInputStream * GetStream () const

  Returns pointer to the stream.

Additional Inherited Members

21.271.2 Constructor & Destructor Documentation

wxFSFile::wxFSFile (wxInputStream * *stream,* const wxString & *location,* const wxString & *mimetype,* const wxString & *anchor,* wxDateTime *modif*)

Constructor.

You probably won't use it. See the Note for details.

It is seldom used by the application programmer but you will need it if you are writing your own virtual FS. For example you may need something similar to wxMemoryInputStream, but because wxMemoryInputStream doesn't free the memory when destroyed and thus passing a memory stream pointer into wxFSFile constructor would lead to memory leaks, you can write your own class derived from wxFSFile:

```
class wxMyFSFile : public wxFSFile
{
    private:
        void *m_Mem;
    public:
        wxMyFSFile(.....)
        ~wxMyFSFile() {free(m_Mem);}
            // of course dtor is virtual ;-)
};
```

If you are not sure of the meaning of these params, see the description of the GetXXXX() functions.

Parameters

stream	The input stream that will be used to access data
location	The full location (aka filename) of the file
mimetype	MIME type of this file. It may be left empty, in which case the type will be determined from file's extension (location must not be empty in this case).
anchor	Anchor. See GetAnchor() for details.
modif	Modification date and time for this file.

21.271.3 Member Function Documentation

wxInputStream* wxFSFile::DetachStream ( )

Detaches the stream from the wxFSFile object.

That is, the stream obtained with GetStream() will continue its existence after the wxFSFile object is deleted.

You will have to delete the stream yourself.

const wxString& wxFSFile::GetAnchor ( ) const

Returns anchor (if present).

The term of **anchor** can be easily explained using few examples:

```
index.htm#anchor                      // 'anchor' is anchor
index/wx001.htm                       // NO anchor here!
archive/main.zip#zip:index.htm#global // 'global'
archive/main.zip#zip:index.htm        // NO anchor here!
```

Usually an anchor is presented only if the MIME type is 'text/html'. But it may have some meaning with other files; for example myanim.avi#200 may refer to position in animation or reality.wrl::MyView may refer to a predefined view in VRML.

const wxString& wxFSFile::GetLocation ( ) const

Returns full location of the file, including path and protocol.

Examples:

```
http://www.wxwidgets.org
http://www.ms.mff.cuni.cz/~vsla8348/wxhtml/archive.zip#zip:info.txt
file:/home/vasek/index.htm
relative-file.htm
```

const wxString& wxFSFile::GetMimeType ( ) const

Returns the MIME type of the content of this file.

It is either extension-based (see wxMimeTypesManager) or extracted from HTTP protocol Content-Type header.

wxDateTime wxFSFile::GetModificationTime ( ) const

Returns time when this file was modified.

wxInputStream* wxFSFile::GetStream ( ) const

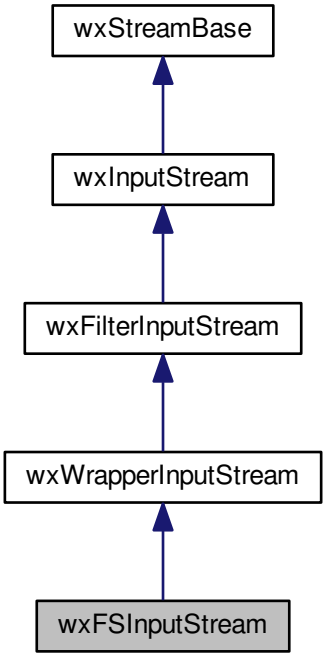
Returns pointer to the stream.

You can use the returned stream to directly access data. You may suppose that the stream provide Seek and GetSize functionality (even in the case of the HTTP protocol which doesn't provide this by default. wxHtml uses local cache to work around this and to speed up the connection).

21.272 wxFSInputStream Class Reference

```
#include <wx/filesys.h>
```

Inheritance diagram for wxFSInputStream:



21.272.1 Detailed Description

Input stream for virtual file stream files.

The stream reads data from wxFSFile obtained from wxFileSystem. It is especially useful to allow using virtual files with other wxWidgets functions and classes working with streams, e.g. for loading images or animations from virtual files and not only physical ones.

Library: wxBase

Category: Streams

See also

wxWrapperInputStream, wxFSFile

Since

2.9.4

Public Member Functions

- wxFileInputStream (const wxString &filename, int flags=0)

  Create a stream associated with the data of the given virtual file system file.

- bool IsOk () const

  Returns true if the stream is initialized and ready.

Additional Inherited Members

21.272.2 Member Function Documentation

bool wxFSInputStream::IsOk ( ) const `[virtual]`

Returns true if the stream is initialized and ready.

Reimplemented from wxStreamBase.

wxFSInputStream::wxFileInputStream (const wxString & *filename*, int *flags* = `0`)

Create a stream associated with the data of the given virtual file system file.

Parameters

filename	The name of the input file passed to wxFileSystem::OpenFile().
flags	Combination of flags from wxFileSystemOpenFlags. wxFS_READ is implied, i.e. it is always added to the flags value.

Use wxStreamBase::IsOk() to verify if the constructor succeeded.

21.273 wxFSVolume Class Reference

`#include <wx/volume.h>`

21.273.1 Detailed Description

wxFSVolume represents a volume (also known as 'drive') in a file system under wxMSW.

Unix ports of wxWidgets do not have the concept of volumes and thus do not implement wxFSVolume.

Availability: only available for the wxMSW port.

Library: wxBase

Category: Miscellaneous

Public Member Functions

- wxFSVolume ()

  Default ctor.
- wxFSVolume (const wxString &name)

  Create the volume object with the given name (which should be one of those returned by GetVolumes()).
- bool Create (const wxString &name)

  Create the volume object with the given name (which should be one of those returned by GetVolumes()).
- bool IsOk () const

  Is this a valid volume?
- wxFSVolumeKind GetKind () const

Returns the kind of this volume.

- int GetFlags () const

  Returns the flags of this volume.

- bool IsWritable () const

  Returns true if this volume is writable.

- wxString GetName () const

  Returns the name of the volume; this is the internal name for the volume used by the operating system.

- wxString GetDisplayName () const

  Returns the name of the volume meant to be shown to the user.

- wxIcon GetIcon (wxFSIconType type) const

  This function is available only when `wxUSE_GUI` is `1`.

Static Public Member Functions

- static wxArrayString GetVolumes (int flagsSet=wxFS_VOL_MOUNTED, int flagsUnset=0)

  Returns an array containing the names of the volumes of this system.

- static void CancelSearch ()

  Stops execution of GetVolumes() called previously (should be called from another thread, of course).

21.273.2 Constructor & Destructor Documentation

wxFSVolume::wxFSVolume ( )

Default ctor.

Use Create() later.

wxFSVolume::wxFSVolume (const wxString & *name*)

Create the volume object with the given *name* (which should be one of those returned by GetVolumes()).

21.273.3 Member Function Documentation

static void wxFSVolume::CancelSearch ( ) [static]

Stops execution of GetVolumes() called previously (should be called from another thread, of course).

bool wxFSVolume::Create (const wxString & *name*)

Create the volume object with the given *name* (which should be one of those returned by GetVolumes()).

wxString wxFSVolume::GetDisplayName ( ) const

Returns the name of the volume meant to be shown to the user.

int wxFSVolume::GetFlags ( ) const

Returns the flags of this volume.

See wxFSVolumeFlags enumeration values.

wxIcon wxFSVolume::GetIcon (wxFSIconType *type*) const

This function is available only when `wxUSE_GUI` is `1`.

Returns the icon used by the native toolkit for the given file system type.

wxFSVolumeKind wxFSVolume::GetKind ( ) const

Returns the kind of this volume.

wxString wxFSVolume::GetName ( ) const

Returns the name of the volume; this is the internal name for the volume used by the operating system.

static wxArrayString wxFSVolume::GetVolumes (int *flagsSet* = wxFS_VOL_MOUNTED, int *flagsUnset* = `0`) `[static]`

Returns an array containing the names of the volumes of this system.

Only the volumes with *flags* such that the expression

```
(flags & flagsSet) == flagsSet && !(flags & flagsUnset)
```

is true, are returned. By default, all mounted ones are returned. See wxFSVolumeFlags enumeration values for a list of valid flags.

This operation may take a while and, even if this function is synchronous, it can be stopped using CancelSearch().

bool wxFSVolume::IsOk ( ) const

Is this a valid volume?

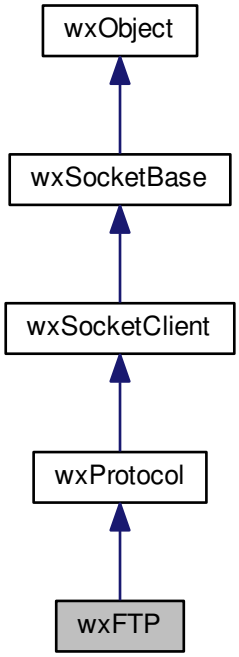
bool wxFSVolume::IsWritable ( ) const

Returns true if this volume is writable.

21.274 wxFTP Class Reference

`#include <wx/protocol/ftp.h>`

Inheritance diagram for wxFTP:



21.274.1 Detailed Description

wxFTP can be used to establish a connection to an FTP server and perform all the usual operations.

Please consult the RFC 959 (http://www.w3.org/Protocols/rfc959/) for more details about the FTP protocol.

wxFTP can thus be used to create a (basic) FTP **client**.

To use a command which doesn't involve file transfer (i.e. directory oriented commands) you just need to call a corresponding member function or use the generic wxFTP::SendCommand() method. However to actually transfer files you just get or give a stream to or from this class and the actual data are read or written using the usual stream methods.

Example of using wxFTP for file downloading:

```
wxFTP ftp;

// if you don't use these lines anonymous login will be used
ftp.SetUser("user");
ftp.SetPassword("password");

if ( !ftp.Connect("ftp.wxwidgets.org") )
{
    wxLogError("Couldn't connect");
    return;
}

ftp.ChDir("/pub/2.8.9");
const char *filename = "wxWidgets-2.8.9.tar.bz2";
int size = ftp.GetFileSize(filename);
if ( size == -1 )
{
    wxLogError("Couldn't get the file size for \"%s\"", filename);
}
```

```
wxInputStream *in = ftp.GetInputStream(filename);
if ( !in )
{
    wxLogError("Couldn't get the file");
}
else
{
    char *data = new char[size];
    if ( !in->Read(data, size) )
    {
        wxLogError("Read error: %d", ftp.GetError());
    }
    else
    {
        // file data is in the buffer
        ...
    }

    delete [] data;
    delete in;
}

// gracefully close the connection to the server
ftp.Close();
```

To upload a file you would do (assuming the connection to the server was opened successfully):

```
wxOutputStream *out = ftp.GetOutputStream("filename");
if ( out )
{
    out->Write(...); // your data
    delete out;
}
```

Library: wxNet

Category: Networking

See also

wxSocketBase

Public Types

- enum TransferMode {
  NONE,
  ASCII,
  BINARY }

  Transfer modes used by wxFTP.

Public Member Functions

- wxFTP ()

  Default constructor.
- virtual ~wxFTP ()

  Destructor will close the connection if connected.

- virtual bool Connect (const wxString &host)

  Connect to the FTP server to default port (21) of the specified host.
- virtual bool Connect (const wxString &host, unsigned short port)

  Connect to the FTP server to any port of the specified host.

Functions for managing the FTP connection

- virtual bool Abort ()

  Aborts the download currently in process, returns true if ok, false if an error occurred.
- virtual bool Close ()

  Gracefully closes the connection with the server.
- bool CheckCommand (const wxString &command, char ret)

  Send the specified command to the FTP server.
- const wxString & GetLastResult ()

  Returns the last command result, i.e.
- char SendCommand (const wxString &command)

  Send the specified command to the FTP server and return the first character of the return code.
- bool SetAscii ()

  Sets the transfer mode to ASCII.
- bool SetBinary ()

  Sets the transfer mode to binary.
- void SetPassive (bool pasv)

  If pasv is true, passive connection to the FTP server is used.
- virtual void SetPassword (const wxString &passwd)

  Sets the password to be sent to the FTP server to be allowed to log in.
- bool SetTransferMode (TransferMode mode)

  Sets the transfer mode to the specified one.
- virtual void SetUser (const wxString &user)

  Sets the user name to be sent to the FTP server to be allowed to log in.

Filesystem commands

- bool ChDir (const wxString &dir)

  Change the current FTP working directory.
- bool MkDir (const wxString &dir)

  Create the specified directory in the current FTP working directory.
- wxString Pwd ()

  Returns the current FTP working directory.
- bool Rename (const wxString &src, const wxString &dst)

  Rename the specified src element to dst.
- bool RmDir (const wxString &dir)

  Remove the specified directory from the current FTP working directory.
- bool RmFile (const wxString &path)

  Delete the file specified by path.
- bool FileExists (const wxString &filename)

  Returns true if the given remote file exists, false otherwise.
- bool GetDirList (wxArrayString &files, const wxString &wildcard=wxEmptyString)

  The GetList() function is quite low-level.
- int GetFileSize (const wxString &filename)

  Returns the file size in bytes or -1 if the file doesn't exist or the size couldn't be determined.
- bool GetFilesList (wxArrayString &files, const wxString &wildcard=wxEmptyString)

  This function returns the computer-parsable list of the files in the current directory (optionally only of the files matching the wildcard, all files by default).

Download and upload functions

- virtual wxInputStream * GetInputStream (const wxString &path)

  Creates a new input stream on the specified path.
- virtual wxOutputStream * GetOutputStream (const wxString &file)

  Initializes an output stream to the specified file.

Additional Inherited Members

21.274.2 Member Enumeration Documentation

enum wxFTP::TransferMode

Transfer modes used by wxFTP.

Enumerator

NONE not set by user explicitly.

ASCII

BINARY

21.274.3 Constructor & Destructor Documentation

wxFTP::wxFTP ( )

Default constructor.

virtual wxFTP::~wxFTP ( ) [virtual]

Destructor will close the connection if connected.

21.274.4 Member Function Documentation

virtual bool wxFTP::Abort ( ) [virtual]

Aborts the download currently in process, returns true if ok, false if an error occurred.

Implements wxProtocol.

bool wxFTP::ChDir (const wxString & *dir*)

Change the current FTP working directory.

Returns true if successful.

bool wxFTP::CheckCommand (const wxString & *command*, char *ret*)

Send the specified *command* to the FTP server.

ret specifies the expected result.

Returns

true if the command has been sent successfully, else false.

virtual bool wxFTP::Close ( ) [virtual]

Gracefully closes the connection with the server.

Reimplemented from wxSocketBase.

virtual bool wxFTP::Connect (const wxString & *host*) [virtual]

Connect to the FTP server to default port (21) of the specified *host*.

virtual bool wxFTP::Connect (const wxString & *host,* unsigned short *port*) [virtual]

Connect to the FTP server to any port of the specified *host*.

By default (*port* = 0), connection is made to default FTP port (21) of the specified *host*.

Since

2.9.1

bool wxFTP::FileExists (const wxString & *filename*)

Returns true if the given remote file exists, false otherwise.

bool wxFTP::GetDirList (wxArrayString & *files,* const wxString & *wildcard* = wxEmptyString)

The GetList() function is quite low-level.

It returns the list of the files in the current directory. The list can be filtered using the *wildcard* string.

If *wildcard* is empty (default), it will return all files in directory. The form of the list can change from one peer system to another. For example, for a UNIX peer system, it will look like this:

```
-r--r--r--   1 guilhem  lavaux      12738 Jan 16 20:17 cmndata.cpp
-r--r--r--   1 guilhem  lavaux      10866 Jan 24 16:41 config.cpp
-rw-rw-rw-   1 guilhem  lavaux      29967 Dec 21 19:17 cwlex_yy.c
-rw-rw-rw-   1 guilhem  lavaux      14342 Jan 22 19:51 cwy_tab.c
-r--r--r--   1 guilhem  lavaux      13890 Jan 29 19:18 date.cpp
-r--r--r--   1 guilhem  lavaux       3989 Feb  8 19:18 datstrm.cpp
```

But on Windows system, it will look like this:

```
winamp~1 exe    520196 02-25-1999  19:28  winamp204.exe
    1 file(s)           520 196 bytes
```

Returns

true if the file list was successfully retrieved, false otherwise.

See also

GetFilesList()

int wxFTP::GetFileSize (const wxString & *filename*)

Returns the file size in bytes or -1 if the file doesn't exist or the size couldn't be determined.

Notice that this size can be approximative size only and shouldn't be used for allocating the buffer in which the remote file is copied, for example.

bool wxFTP::GetFilesList (wxArrayString & *files*, const wxString & *wildcard* = wxEmptyString)

This function returns the computer-parsable list of the files in the current directory (optionally only of the files matching the *wildcard*, all files by default).

This list always has the same format and contains one full (including the directory path) file name per line.

Returns

true if the file list was successfully retrieved, false otherwise.

See also

GetDirList()

virtual wxInputStream* wxFTP::GetInputStream (const wxString & *path*) [virtual]

Creates a new input stream on the specified path.

You can use all but the seek functionality of wxStreamBase. wxStreamBase::Seek() isn't available on all streams. For example, HTTP or FTP streams do not deal with it. Other functions like wxStreamBase::Tell() are not available for this sort of stream, at present.

You will be notified when the EOF is reached by an error.

Returns

Returns NULL if an error occurred (it could be a network failure or the fact that the file doesn't exist).

Implements wxProtocol.

const wxString& wxFTP::GetLastResult ( )

Returns the last command result, i.e.

the full server reply for the last command.

virtual wxOutputStream* wxFTP::GetOutputStream (const wxString & *file*) [virtual]

Initializes an output stream to the specified *file*.

The returned stream has all but the seek functionality of wxStreams. When the user finishes writing data, he has to delete the stream to close it.

Returns

An initialized write-only stream. Returns NULL if an error occurred (it could be a network failure or the fact that the file doesn't exist).

bool wxFTP::MkDir (const wxString & *dir*)

Create the specified directory in the current FTP working directory.

Returns true if successful.

wxString wxFTP::Pwd ( )

Returns the current FTP working directory.

bool wxFTP::Rename (const wxString & *src,* const wxString & *dst*)

Rename the specified *src* element to *dst*.

Returns true if successful.

bool wxFTP::RmDir (const wxString & *dir*)

Remove the specified directory from the current FTP working directory.

Returns true if successful.

bool wxFTP::RmFile (const wxString & *path*)

Delete the file specified by *path*.

Returns true if successful.

char wxFTP::SendCommand (const wxString & *command*)

Send the specified *command* to the FTP server and return the first character of the return code.

bool wxFTP::SetAscii ( )

Sets the transfer mode to ASCII.

It will be used for the next transfer.

bool wxFTP::SetBinary ( )

Sets the transfer mode to binary.

It will be used for the next transfer.

void wxFTP::SetPassive (bool *pasv*)

If *pasv* is true, passive connection to the FTP server is used.

This is the default as it works with practically all firewalls. If the server doesn't support passive mode, you may call this function with false as argument to use an active connection.

virtual void wxFTP::SetPassword (const wxString & *passwd*) `[virtual]`

Sets the password to be sent to the FTP server to be allowed to log in.

Reimplemented from wxProtocol.

bool wxFTP::SetTransferMode (TransferMode *mode*)

Sets the transfer mode to the specified one.

It will be used for the next transfer.

If this function is never called, binary transfer mode is used by default.

virtual void wxFTP::SetUser (const wxString & *user*) `[virtual]`

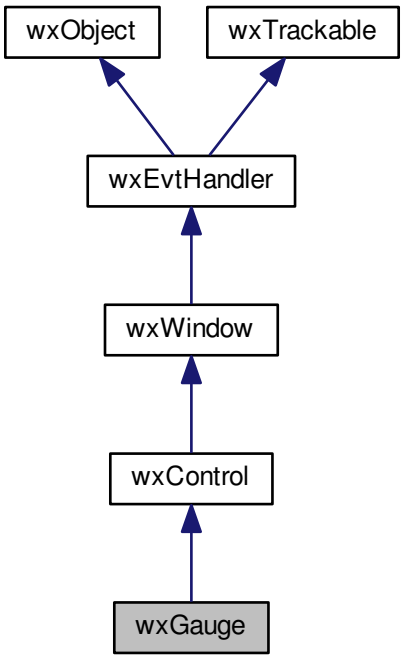
Sets the user name to be sent to the FTP server to be allowed to log in.

Reimplemented from wxProtocol.

21.275 wxGauge Class Reference

`#include <wx/gauge.h>`

Inheritance diagram for wxGauge:



21.275.1 Detailed Description

A gauge is a horizontal or vertical bar which shows a quantity (often time).

wxGauge supports two working modes: determinate and indeterminate progress.

The first is the usual working mode (see SetValue() and SetRange()) while the second can be used when the program is doing some processing but you don't know how much progress is being done. In this case, you can periodically call the Pulse() function to make the progress bar switch to indeterminate mode (graphically it's usually a set of blocks which move or bounce in the bar control).

wxGauge supports dynamic switch between these two work modes.

There are no user commands for the gauge.

Styles

This class supports the following styles:

- wxGA_HORIZONTAL: Creates a horizontal gauge.
- wxGA_VERTICAL: Creates a vertical gauge.
- wxGA_SMOOTH: Creates smooth progress bar with one pixel wide update step (not supported by all platforms).
- wxGA_TEXT: Display the current value in percents in the gauge itself. This style is only supported in wxQt and ignored under the other platforms.

  Since

  3.1.0
- wxGA_PROGRESS: Reflect the value of gauge in the application taskbar button under Windows 7 and later, ignored under the other platforms.

  Since

  3.1.0

Library: wxCore

Category: Controls

See also

wxSlider, wxScrollBar

Public Member Functions

- wxGauge ()

  Default constructor.
- wxGauge (wxWindow *parent, wxWindowID id, int range, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxGA_HORIZONTAL, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxGaugeNameStr)

  Constructor, creating and showing a gauge.
- virtual ~wxGauge ()

  Destructor, destroying the gauge.
- bool Create (wxWindow *parent, wxWindowID id, int range, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxGA_HORIZONTAL, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxGaugeNameStr)

  Creates the gauge for two-step construction.
- int GetBezelFace () const

  Returns the width of the 3D bezel face.
- int GetRange () const

  Returns the maximum position of the gauge.
- int GetShadowWidth () const

  Returns the 3D shadow margin width.
- int GetValue () const

  Returns the current position of the gauge.
- bool IsVertical () const

Returns true if the gauge is vertical (has `wxGA_VERTICAL` style) and false otherwise.

- virtual void Pulse ()

  Switch the gauge to indeterminate mode (if required) and makes the gauge move a bit to indicate the user that some progress has been made.

- void SetBezelFace (int width)

  Sets the 3D bezel face width.

- void SetRange (int range)

  Sets the range (maximum value) of the gauge.

- void SetShadowWidth (int width)

  Sets the 3D shadow width.

- void SetValue (int pos)

  Sets the position of the gauge.

Additional Inherited Members

21.275.2 Constructor & Destructor Documentation

wxGauge::wxGauge ( )

Default constructor.

wxGauge::wxGauge (wxWindow * *parent*, wxWindowID *id*, int *range*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxGA_HORIZONTAL, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxGaugeNameStr`)

Constructor, creating and showing a gauge.

Parameters

parent	Window parent.
id	Window identifier.
range	Integer range (maximum value) of the gauge. See SetRange() for more details about the meaning of this value when using the gauge in indeterminate mode.
pos	Window position.
size	Window size.
style	Gauge style.
validator	Window validator.
name	Window name.

See also

Create()

virtual wxGauge::~wxGauge ( ) `[virtual]`

Destructor, destroying the gauge.

21.275.3 Member Function Documentation

bool wxGauge::Create (wxWindow * *parent*, wxWindowID *id*, int *range*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxGA_HORIZONTAL, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxGaugeNameStr`)

Creates the gauge for two-step construction.

See wxGauge() for further details.

int wxGauge::GetBezelFace ( ) const

Returns the width of the 3D bezel face.

Remarks

This method is not implemented (returns 0) for most platforms.

See also

SetBezelFace()

int wxGauge::GetRange ( ) const

Returns the maximum position of the gauge.

See also

SetRange()

int wxGauge::GetShadowWidth ( ) const

Returns the 3D shadow margin width.

Remarks

This method is not implemented (returns 0) for most platforms.

See also

SetShadowWidth()

int wxGauge::GetValue ( ) const

Returns the current position of the gauge.

See also

SetValue()

bool wxGauge::IsVertical ( ) const

Returns true if the gauge is vertical (has `wxGA_VERTICAL` style) and false otherwise.

virtual void wxGauge::Pulse ( ) [virtual]

Switch the gauge to indeterminate mode (if required) and makes the gauge move a bit to indicate the user that some progress has been made.

Note

After calling this function the value returned by GetValue() is undefined and thus you need to explicitly call SetValue() if you want to restore the determinate mode.

void wxGauge::SetBezelFace (int *width*)

Sets the 3D bezel face width.

Remarks

This method is not implemented (doesn't do anything) for most platforms.

See also

GetBezelFace()

void wxGauge::SetRange (int *range*)

Sets the range (maximum value) of the gauge.

This function makes the gauge switch to determinate mode, if it's not already.

When the gauge is in indeterminate mode, under wxMSW the gauge repeatedly goes from zero to *range* and back; under other ports when in indeterminate mode, the *range* setting is ignored.

See also

GetRange()

void wxGauge::SetShadowWidth (int *width*)

Sets the 3D shadow width.

Remarks

This method is not implemented (doesn't do anything) for most platforms.

void wxGauge::SetValue (int *pos*)

Sets the position of the gauge.

The *pos* must be between 0 and the gauge range as returned by GetRange(), inclusive.

This function makes the gauge switch to determinate mode, if it was in indeterminate mode before.

Parameters

pos	Position for the gauge level.

See also

GetValue()

21.276 wxGBPosition Class Reference

`#include <wx/gbsizer.h>`

21.276.1 Detailed Description

This class represents the position of an item in a virtual grid of rows and columns managed by a wxGridBagSizer.

Library: wxCore

Category: Window Layout

Public Member Functions

- wxGBPosition ()

  Default constructor, setting the row and column to (0,0).
- wxGBPosition (int row, int col)

  Construct a new wxGBPosition, setting the row and column.
- int GetCol () const

  Get the current column value.
- int GetRow () const

  Get the current row value.
- void SetCol (int col)

  Set a new column value.
- void SetRow (int row)

  Set a new row value.
- bool operator!= (const wxGBPosition &p) const

  Compare inequality of two wxGBPositions.
- bool operator== (const wxGBPosition &p) const

  Compare equality of two wxGBPositions.

21.276.2 Constructor & Destructor Documentation

wxGBPosition::wxGBPosition ( )

Default constructor, setting the row and column to (0,0).

wxGBPosition::wxGBPosition (int *row,* int *col*)

Construct a new wxGBPosition, setting the row and column.

21.276.3 Member Function Documentation

int wxGBPosition::GetCol ( ) const

Get the current column value.

int wxGBPosition::GetRow ( ) const

Get the current row value.

bool wxGBPosition::operator!= (const wxGBPosition & *p*) const

Compare inequality of two wxGBPositions.

bool wxGBPosition::operator== (const wxGBPosition & *p*) const

Compare equality of two wxGBPositions.

void wxGBPosition::SetCol (int *col*)

Set a new column value.

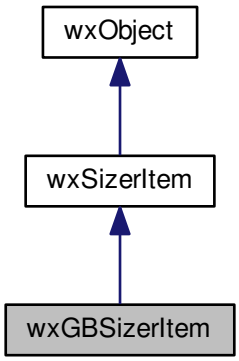
void wxGBPosition::SetRow (int *row*)

Set a new row value.

21.277 wxGBSizerItem Class Reference

`#include <wx/gbsizer.h>`

Inheritance diagram for wxGBSizerItem:



21.277.1 Detailed Description

The wxGBSizerItem class is used by the wxGridBagSizer for tracking the items in the sizer.

It adds grid position and spanning information to the normal wxSizerItem by adding wxGBPosition and wxGBSpan attributes. Most of the time you will not need to use a wxGBSizerItem directly in your code, but there are a couple of cases where it is handy.

Library: wxCore

Category: Window Layout

Public Member Functions

- wxGBSizerItem (int width, int height, const wxGBPosition &pos, const wxGBSpan &span=wxDefaultSpan, int flag=0, int border=0, wxObject ∗userData=NULL)

  Construct a sizer item for tracking a spacer.
- wxGBSizerItem (wxWindow ∗window, const wxGBPosition &pos, const wxGBSpan &span=wxDefaultSpan, int flag=0, int border=0, wxObject ∗userData=NULL)

  Construct a sizer item for tracking a window.
- wxGBSizerItem (wxSizer ∗sizer, const wxGBPosition &pos, const wxGBSpan &span=wxDefaultSpan, int flag=0, int border=0, wxObject ∗userData=NULL)

  Construct a sizer item for tracking a subsizer.
- void GetEndPos (int &row, int &col)

  Get the row and column of the endpoint of this item.
- bool Intersects (const wxGBSizerItem &other)

  Returns true if this item and the other item intersect.
- bool Intersects (const wxGBPosition &pos, const wxGBSpan &span)

  Returns true if the given pos/span would intersect with this item.
- bool SetPos (const wxGBPosition &pos)

  If the item is already a member of a sizer then first ensure that there is no other item that would intersect with this one at the new position, then set the new position.
- bool SetSpan (const wxGBSpan &span)

  If the item is already a member of a sizer then first ensure that there is no other item that would intersect with this one with its new spanning size, then set the new spanning.
- wxGridBagSizer ∗ GetGBSizer () const
- void SetGBSizer (wxGridBagSizer ∗sizer)

- wxGBPosition GetPos () const

  Get the grid position of the item.
- void GetPos (int &row, int &col) const

  Get the grid position of the item.

- wxGBSpan GetSpan () const

  Get the row and column spanning of the item.
- void GetSpan (int &rowspan, int &colspan) const

  Get the row and column spanning of the item.

Additional Inherited Members

21.277.2 Constructor & Destructor Documentation

wxGBSizerItem::wxGBSizerItem (int *width*, int *height*, const wxGBPosition & *pos*, const wxGBSpan & *span* = wxDefaultSpan, int *flag* = 0, int *border* = 0, wxObject ∗ *userData* = NULL)

Construct a sizer item for tracking a spacer.

wxGBSizerItem::wxGBSizerItem (wxWindow ∗ *window*, const wxGBPosition & *pos*, const wxGBSpan & *span* = wxDefaultSpan, int *flag* = 0, int *border* = 0, wxObject ∗ *userData* = NULL)

Construct a sizer item for tracking a window.

wxGBSizerItem::wxGBSizerItem (wxSizer * *sizer,* const wxGBPosition & *pos,* const wxGBSpan & *span =* wxDefaultSpan, int *flag =* 0, int *border =* 0, wxObject * *userData =* NULL)

Construct a sizer item for tracking a subsizer.

21.277.3 Member Function Documentation

void wxGBSizerItem::GetEndPos (int & *row,* int & *col*)

Get the row and column of the endpoint of this item.

wxGridBagSizer* wxGBSizerItem::GetGBSizer ( ) const

wxGBPosition wxGBSizerItem::GetPos ( ) const

Get the grid position of the item.

void wxGBSizerItem::GetPos (int & *row,* int & *col*) const

Get the grid position of the item.

wxGBSpan wxGBSizerItem::GetSpan ( ) const

Get the row and column spanning of the item.

void wxGBSizerItem::GetSpan (int & *rowspan,* int & *colspan*) const

Get the row and column spanning of the item.

bool wxGBSizerItem::Intersects (const wxGBSizerItem & *other*)

Returns true if this item and the *other* item intersect.

bool wxGBSizerItem::Intersects (const wxGBPosition & *pos,* const wxGBSpan & *span*)

Returns true if the given pos/span would intersect with this item.

void wxGBSizerItem::SetGBSizer (wxGridBagSizer * *sizer*)

bool wxGBSizerItem::SetPos (const wxGBPosition & *pos*)

If the item is already a member of a sizer then first ensure that there is no other item that would intersect with this one at the new position, then set the new position.

Returns true if the change is successful and after the next Layout the item will be moved.

bool wxGBSizerItem::SetSpan (const wxGBSpan & *span*)

If the item is already a member of a sizer then first ensure that there is no other item that would intersect with this one with its new spanning size, then set the new spanning.

Returns true if the change is successful and after the next Layout the item will be resized.

21.278 wxGBSpan Class Reference

```
#include <wx/gbsizer.h>
```

21.278.1 Detailed Description

This class is used to hold the row and column spanning attributes of items in a wxGridBagSizer.

Library: wxCore

Category: Window Layout

Public Member Functions

- wxGBSpan ()

  Default constructor, setting the rowspan and colspan to (1,1) meaning that the item occupies one cell in each direction.
- wxGBSpan (int rowspan, int colspan)

  Construct a new wxGBSpan, setting the rowspan and colspan.
- int GetColspan () const

  Get the current colspan value.
- int GetRowspan () const

  Get the current rowspan value.
- void SetColspan (int colspan)

  Set a new colspan value.
- void SetRowspan (int rowspan)

  Set a new rowspan value.
- bool operator!= (const wxGBSpan &o) const

  Compare inequality of two wxGBSpans.
- bool operator== (const wxGBSpan &o) const

  Compare equality of two wxGBSpans.

21.278.2 Constructor & Destructor Documentation

wxGBSpan::wxGBSpan ( )

Default constructor, setting the rowspan and colspan to (1,1) meaning that the item occupies one cell in each direction.

wxGBSpan::wxGBSpan (int *rowspan*, int *colspan*)

Construct a new wxGBSpan, setting the *rowspan* and *colspan*.

21.278.3 Member Function Documentation

int wxGBSpan::GetColspan ( ) const

Get the current colspan value.

int wxGBSpan::GetRowspan ( ) const

Get the current rowspan value.

bool wxGBSpan::operator!= (const wxGBSpan & *o*) const

Compare inequality of two wxGBSpans.

bool wxGBSpan::operator== (const wxGBSpan & *o*) const

Compare equality of two wxGBSpans.

void wxGBSpan::SetColspan (int *colspan*)

Set a new colspan value.

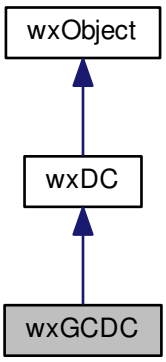
void wxGBSpan::SetRowspan (int *rowspan*)

Set a new rowspan value.

21.279 wxGCDC Class Reference

`#include <wx/dcgraph.h>`

Inheritance diagram for wxGCDC:



21.279.1 Detailed Description

wxGCDC is a device context that draws on a wxGraphicsContext.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxGraphicsContext

Public Member Functions

- wxGCDC (const wxWindowDC &windowDC)

  Constructs a wxGCDC from a wxWindowDC.
- wxGCDC (const wxMemoryDC &memoryDC)

  Constructs a wxGCDC from a wxMemoryDC.
- wxGCDC (const wxPrinterDC &printerDC)

  Constructs a wxGCDC from a wxPrinterDC.
- wxGCDC (wxGraphicsContext *context)

  Construct a wxGCDC from an existing graphics context.
- wxGCDC (const wxEnhMetaFileDC &emfDC)

  Constructs a wxGCDC from a wxEnhMetaFileDC.
- wxGCDC ()
- virtual ~wxGCDC ()
- wxGraphicsContext * GetGraphicsContext () const

  Retrieves associated wxGraphicsContext.
- void SetGraphicsContext (wxGraphicsContext *ctx)

  Set the graphics context to be used for this wxGCDC.

Additional Inherited Members

21.279.2 Constructor & Destructor Documentation

wxGCDC::wxGCDC (const wxWindowDC & *windowDC*)

Constructs a wxGCDC from a wxWindowDC.

wxGCDC::wxGCDC (const wxMemoryDC & *memoryDC*)

Constructs a wxGCDC from a wxMemoryDC.

wxGCDC::wxGCDC (const wxPrinterDC & *printerDC*)

Constructs a wxGCDC from a wxPrinterDC.

wxGCDC::wxGCDC (wxGraphicsContext * *context*)

Construct a wxGCDC from an existing graphics context.

wxGCDC::wxGCDC (const wxEnhMetaFileDC & *emfDC*)

Constructs a wxGCDC from a wxEnhMetaFileDC.

This constructor is only available in wxMSW port and when `wxUSE_ENH_METAFILE` build option is enabled, i.e. when wxEnhMetaFileDC class itself is available.

Since

2.9.3

wxGCDC::wxGCDC ( )

virtual wxGCDC::~wxGCDC ( ) [virtual]

21.279.3 Member Function Documentation

wxGraphicsContext* wxGCDC::GetGraphicsContext ( ) const

Retrieves associated wxGraphicsContext.

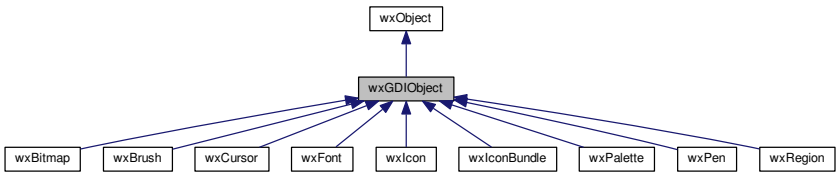
void wxGCDC::SetGraphicsContext (wxGraphicsContext * *ctx*)

Set the graphics context to be used for this wxGCDC.

21.280 wxGDIObject Class Reference

`#include <wx/gdiobj.h>`

Inheritance diagram for wxGDIObject:



21.280.1 Detailed Description

This class allows platforms to implement functionality to optimise GDI objects, such as wxPen, wxBrush and wxFont.

On Windows, the underling GDI objects are a scarce resource and are cleaned up when a usage count goes to zero. On some platforms this class may not have any special functionality.

Since the functionality of this class is platform-specific, it is not documented here in detail.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxPen, wxBrush, wxFont

Public Member Functions

- wxGDIObject ()

  Default constructor.

Additional Inherited Members

21.280.2 Constructor & Destructor Documentation

wxGDIObject::wxGDIObject ( )

Default constructor.

21.281 wxGenericAboutDialog Class Reference

`#include <wx/generic/aboutdlgg.h>`

21.281.1 Detailed Description

This class defines a customizable *About* dialog.

Note that if you don't need customization, you should use the global wxAboutBox() function that is both easier to use and shows the native dialog if available.

To use this class, you need to derive your own class from it and override the virtual method DoAddCustomControls().

To instantiate an object from your wxGenericAboutDialog-based class, you can use either the default constructor followed by a call to Create(), or directly using the alternate constructor. In either case, you have to prepare a wxAboutDialogInfo containing standard informations to display in an about-box.

Example of usage, MyAboutDlg being a class derived from wxGenericAboutDialog:

```
void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxAboutDialogInfo aboutInfo;

    aboutInfo.SetName("MyApp");
    aboutInfo.SetVersion(MY_APP_VERSION_STRING);
    aboutInfo.SetDescription(_("My wxWidgets-based application!"));
    aboutInfo.SetCopyright("(C) 1992-2012");
    aboutInfo.SetWebSite("http://myapp.org");
    aboutInfo.AddDeveloper("My Self");

    MyAboutDlg dlgAbout(aboutInfo, this);
    dlgAbout.ShowModal();
}
```

Library: wxAdvanced

Category: Common Dialogs

See also

wxAboutDialogInfo

Public Member Functions

- wxGenericAboutDialog ()

Default constructor, Create() must be called later.

- wxGenericAboutDialog (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  Creates the dialog and initializes it with the given information.

- bool Create (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  Initializes the dialog created using the default constructor.

Protected Member Functions

- virtual void DoAddCustomControls ()

  This virtual method may be overridden to add more controls to the dialog.

- void AddControl (wxWindow *win, const wxSizerFlags &flags)

  Add arbitrary control to the sizer content with the specified flags.

- void AddControl (wxWindow *win)

  Add arbitrary control to the sizer content and centre it.

- void AddText (const wxString &text)

  Add the given (not empty) text to the sizer content.

- void AddCollapsiblePane (const wxString &title, const wxString &text)

  Add a wxCollapsiblePane containing the given text.

21.281.2 Constructor & Destructor Documentation

wxGenericAboutDialog::wxGenericAboutDialog ( )

Default constructor, Create() must be called later.

wxGenericAboutDialog::wxGenericAboutDialog (const wxAboutDialogInfo & *info*, wxWindow * *parent* = NULL)

Creates the dialog and initializes it with the given information.

21.281.3 Member Function Documentation

void wxGenericAboutDialog::AddCollapsiblePane (const wxString & *title*, const wxString & *text*) [protected]

Add a wxCollapsiblePane containing the given text.

void wxGenericAboutDialog::AddControl (wxWindow * *win*, const wxSizerFlags & *flags*) [protected]

Add arbitrary control to the sizer content with the specified flags.

For example, here is how to add an expandable line with a border of 3 pixels, then a line of text:

```
AddControl(new wxStaticLine(this), wxSizerFlags().Expand().Border(
      wxALL, 3));

AddText(_("This line is just an example of custom text."));
```

void wxGenericAboutDialog::AddControl (wxWindow * *win*) [protected]

Add arbitrary control to the sizer content and centre it.

void wxGenericAboutDialog::AddText (const wxString & *text*) `[protected]`

Add the given (not empty) text to the sizer content.

bool wxGenericAboutDialog::Create (const wxAboutDialogInfo & *info*, wxWindow * *parent* = NULL)

Initializes the dialog created using the default constructor.

virtual void wxGenericAboutDialog::DoAddCustomControls ( ) `[inline], [protected], [virtual]`

This virtual method may be overridden to add more controls to the dialog.

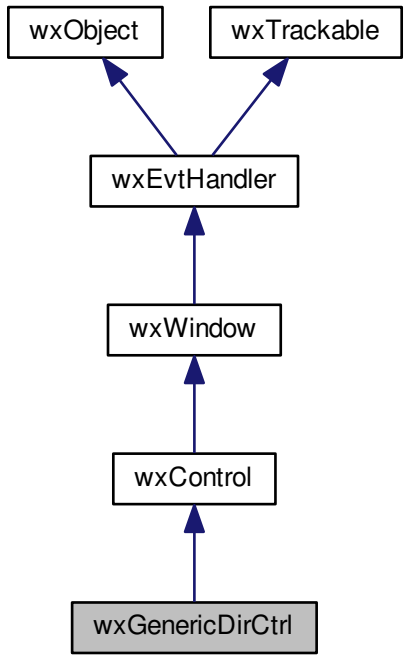
Use the protected AddControl(), AddText() and AddCollapsiblePane() methods to add custom controls.

This method is called during the dialog creation and you don't need to call it, only to override it.

21.282 wxGenericDirCtrl Class Reference

`#include <wx/dirctrl.h>`

Inheritance diagram for wxGenericDirCtrl:



21.282.1 Detailed Description

This control can be used to place a directory listing (with optional files) on an arbitrary window.

The control contains a wxTreeCtrl window representing the directory hierarchy, and optionally, a wxChoice window containing a list of filters.

Styles

This class supports the following styles:

- wxDIRCTRL_DIR_ONLY: Only show directories, and not files.
- wxDIRCTRL_3D_INTERNAL: Use 3D borders for internal controls. This is the default.
- wxDIRCTRL_SELECT_FIRST: When setting the default path, select the first file in the directory.
- wxDIRCTRL_SHOW_FILTERS: Show the drop-down filter list.
- wxDIRCTRL_EDIT_LABELS: Allow the folder and file labels to be editable.
- wxDIRCTRL_MULTIPLE: Allows multiple files and folders to be selected.

Library: wxCore

Category: Controls

Events emitted by this class

Event macros for events emitted by this class:

- EVT_DIRCTRL_SELECTIONCHANGED(id, func): Selected directory has changed. Processes a `wxEVT_DIRCTRL_SELECTIONCHANGED` event type. Notice that this event is generated even for the changes done by the program itself and not only those done by the user. Available since wxWidgets 2.9.5.
- EVT_DIRCTRL_FILEACTIVATED(id, func): The user activated a file by double-clicking or pressing Enter. Available since wxWidgets 2.9.5.

Public Member Functions

- wxGenericDirCtrl ()

  Default constructor.
- wxGenericDirCtrl (wxWindow *parent, const wxWindowID id=wxID_ANY, const wxString &dir=wxDirDialogDefaultFolderStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDIRCTRL_DEFAULT_STYLE, const wxString &filter=wxEmptyString, int defaultFilter=0, const wxString &name=wxTreeCtrlNameStr)

  Main constructor.
- virtual ~wxGenericDirCtrl ()

  Destructor.
- virtual bool CollapsePath (const wxString &path)

  Collapse the given path.
- virtual void CollapseTree ()

  Collapses the entire tree.
- bool Create (wxWindow *parent, const wxWindowID id=wxID_ANY, const wxString &dir=wxDirDialogDefaultFolderStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDIRCTRL_DEFAULT_STYLE, const wxString &filter=wxEmptyString, int defaultFilter=0, const wxString &name=wxTreeCtrlNameStr)

Create function for two-step construction.

- virtual bool ExpandPath (const wxString &path)

  Tries to expand as much of the given path as possible, so that the filename or directory is visible in the tree control.
- virtual wxString GetDefaultPath () const

  Gets the default path.
- virtual wxString GetFilePath () const

  Gets selected filename path only (else empty string).
- virtual void GetFilePaths (wxArrayString &paths) const

  Fills the array paths with the currently selected filepaths.
- virtual wxString GetFilter () const

  Returns the filter string.
- virtual int GetFilterIndex () const

  Returns the current filter index (zero-based).
- virtual wxDirFilterListCtrl * GetFilterListCtrl () const

  Returns a pointer to the filter list control (if present).
- virtual wxString GetPath () const

  Gets the currently-selected directory or filename.
- wxString GetPath (wxTreeItemId itemId) const

  Gets the path corresponding to the given tree control item.
- virtual void GetPaths (wxArrayString &paths) const

  Fills the array paths with the selected directories and filenames.
- virtual wxTreeItemId GetRootId ()

  Returns the root id for the tree control.
- virtual wxTreeCtrl * GetTreeCtrl () const

  Returns a pointer to the tree control.
- virtual void Init ()

  Initializes variables.
- virtual void ReCreateTree ()

  Collapse and expand the tree, thus re-creating it from scratch.
- virtual void SetDefaultPath (const wxString &path)

  Sets the default path.
- virtual void SetFilter (const wxString &filter)

  Sets the filter string.
- virtual void SetFilterIndex (int n)

  Sets the current filter index (zero-based).
- virtual void SetPath (const wxString &path)

  Sets the current path.
- virtual void ShowHidden (bool show)
- virtual void SelectPath (const wxString &path, bool select=true)

  Selects the given item.
- virtual void SelectPaths (const wxArrayString &paths)

  Selects only the specified paths, clearing any previous selection.
- virtual void UnselectAll ()

  Removes the selection from all currently selected items.

Additional Inherited Members

21.282.2 Constructor & Destructor Documentation

wxGenericDirCtrl::wxGenericDirCtrl ( )

Default constructor.

wxGenericDirCtrl::wxGenericDirCtrl (wxWindow * *parent*, const wxWindowID *id* = wxID_ANY, const wxString & *dir* = wxDirDialogDefaultFolderStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDIRCTRL_DEFAULT_STYLE, const wxString & *filter* = wxEmptyString, int *defaultFilter* = 0, const wxString & *name* = wxTreeCtrlNameStr)

Main constructor.

Parameters

parent	Parent window.
id	Window identifier.
dir	Initial folder.
pos	Position.
size	Size.
style	Window style. Please see wxGenericDirCtrl for a list of possible styles.
filter	A filter string, using the same syntax as that for wxFileDialog. This may be empty if filters are not being used. Example: `"All files (*.*)|*.*|JPEG files (*.jpg)|*.jpg"`
defaultFilter	The zero-indexed default filter setting.
name	The window name.

virtual wxGenericDirCtrl::~wxGenericDirCtrl ( ) [virtual]

Destructor.

21.282.3 Member Function Documentation

virtual bool wxGenericDirCtrl::CollapsePath (const wxString & *path*) [virtual]

Collapse the given *path*.

virtual void wxGenericDirCtrl::CollapseTree ( ) [virtual]

Collapses the entire tree.

bool wxGenericDirCtrl::Create (wxWindow * *parent*, const wxWindowID *id* = wxID_ANY, const wxString & *dir* = wxDirDialogDefaultFolderStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDIRCTRL_DEFAULT_STYLE, const wxString & *filter* = wxEmptyString, int *defaultFilter* = 0, const wxString & *name* = wxTreeCtrlNameStr)

Create function for two-step construction.

See wxGenericDirCtrl() for details.

virtual bool wxGenericDirCtrl::ExpandPath (const wxString & *path*) [virtual]

Tries to expand as much of the given *path* as possible, so that the filename or directory is visible in the tree control.

virtual wxString wxGenericDirCtrl::GetDefaultPath ( ) const [virtual]

Gets the default path.

virtual wxString wxGenericDirCtrl::GetFilePath ( ) const [virtual]

Gets selected filename path only (else empty string).

This function doesn't count a directory as a selection.

virtual void wxGenericDirCtrl::GetFilePaths (wxArrayString & *paths*) const [virtual]

Fills the array *paths* with the currently selected filepaths.

This function doesn't count a directory as a selection.

virtual wxString wxGenericDirCtrl::GetFilter ( ) const [virtual]

Returns the filter string.

virtual int wxGenericDirCtrl::GetFilterIndex ( ) const [virtual]

Returns the current filter index (zero-based).

virtual wxDirFilterListCtrl* wxGenericDirCtrl::GetFilterListCtrl ( ) const [virtual]

Returns a pointer to the filter list control (if present).

virtual wxString wxGenericDirCtrl::GetPath ( ) const [virtual]

Gets the currently-selected directory or filename.

wxString wxGenericDirCtrl::GetPath (wxTreeItemId *itemId*) const

Gets the path corresponding to the given tree control item.

Since

2.9.5

virtual void wxGenericDirCtrl::GetPaths (wxArrayString & *paths*) const [virtual]

Fills the array *paths* with the selected directories and filenames.

virtual wxTreeItemId wxGenericDirCtrl::GetRootId ( ) [virtual]

Returns the root id for the tree control.

virtual wxTreeCtrl* wxGenericDirCtrl::GetTreeCtrl ( ) const [virtual]

Returns a pointer to the tree control.

virtual void wxGenericDirCtrl::Init ( ) [virtual]

Initializes variables.

virtual void wxGenericDirCtrl::ReCreateTree ( ) [virtual]

Collapse and expand the tree, thus re-creating it from scratch.

May be used to update the displayed directory content.

virtual void wxGenericDirCtrl::SelectPath (const wxString & *path,* bool *select* = true) [virtual]

Selects the given item.

In multiple selection controls, can be also used to deselect a currently selected item if the value of *select* is false. Existing selections are not changed. Only visible items can be (de)selected, otherwise use ExpandPath().

virtual void wxGenericDirCtrl::SelectPaths (const wxArrayString & *paths*) [virtual]

Selects only the specified paths, clearing any previous selection.

Only supported when wxDIRCTRL_MULTIPLE is set.

virtual void wxGenericDirCtrl::SetDefaultPath (const wxString & *path*) [virtual]

Sets the default path.

virtual void wxGenericDirCtrl::SetFilter (const wxString & *filter*) [virtual]

Sets the filter string.

virtual void wxGenericDirCtrl::SetFilterIndex (int *n*) [virtual]

Sets the current filter index (zero-based).

virtual void wxGenericDirCtrl::SetPath (const wxString & *path*) [virtual]

Sets the current path.

virtual void wxGenericDirCtrl::ShowHidden (bool *show*) [virtual]

Parameters

show	If true, hidden folders and files will be displayed by the control. If false, they will not be displayed.

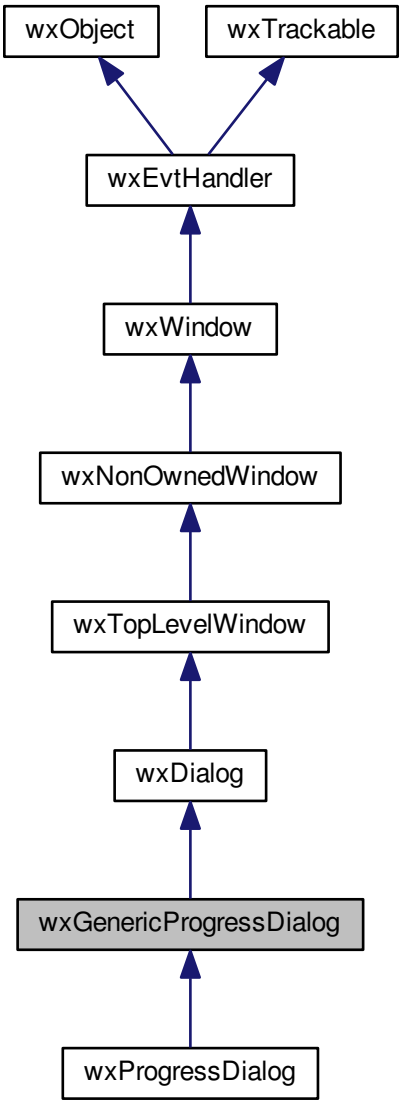
virtual void wxGenericDirCtrl::UnselectAll ( ) [virtual]

Removes the selection from all currently selected items.

21.283 wxGenericProgressDialog Class Reference

```
#include <wx/progdlg.h>
```

Inheritance diagram for wxGenericProgressDialog:



21.283.1 Detailed Description

This class represents a dialog that shows a short message and a progress bar.

Optionally, it can display ABORT and SKIP buttons, and the elapsed, remaining and estimated time for the end of the progress.

This class provides a generic implementation of the progress dialog. If the platform has a native progress dialog available then it will be accessible using the *wxProgressDialog* class, otherwise it will essentially be the same as this class.

Note that you must be aware that wxProgressDialog internally calls wxEventLoopBase::YieldFor with `wxEVT_CATEGORY_UI` and `wxEVT_CATEGORY_USER_INPUT` and this may cause unwanted re-entrancies or the out-of-order processing of pending events (to help preventing the last problem if you're using wxProgressDialog in

a multi-threaded application you should be sure to use wxThreadEvent for your inter-threads communications).

Styles

This class supports the following styles:

- wxPD_APP_MODAL: Make the progress dialog modal. If this flag is not given, it is only "locally" modal - that is the input to the parent window is disabled, but not to the other ones.
- wxPD_AUTO_HIDE: Causes the progress dialog to disappear from screen as soon as the maximum value of the progress meter has been reached. If this style is not present, the dialog will become a modal dialog (see wxDialog::ShowModal) once the maximum value has been reached and wait for the user to dismiss it.
- wxPD_SMOOTH: Causes smooth progress of the gauge control (uses a wxGauge with the `wxGA_SMOOTH` style).
- wxPD_CAN_ABORT: This flag tells the dialog that it should have a "Cancel" button which the user may press. If this happens, the next call to Update() will return false.
- wxPD_CAN_SKIP: This flag tells the dialog that it should have a "Skip" button which the user may press. If this happens, the next call to Update() will return true in its skip parameter.
- wxPD_ELAPSED_TIME: This flag tells the dialog that it should show elapsed time (since creating the dialog).
- wxPD_ESTIMATED_TIME: This flag tells the dialog that it should show estimated time.
- wxPD_REMAINING_TIME: This flag tells the dialog that it should show remaining time.

Library: wxCore

Category: Common Dialogs

Public Member Functions

- wxGenericProgressDialog (const wxString &title, const wxString &message, int maximum=100, wxWindow *parent=NULL, int style=wxPD_AUTO_HIDE|wxPD_APP_MODAL)

  Constructor.
- virtual ~wxGenericProgressDialog ()

  Destructor.
- int GetValue () const

  Returns the last value passed to the Update() function or `wxNOT_FOUND` if the dialog has no progress bar.
- int GetRange () const

  Returns the maximum value of the progress meter, as passed to the constructor or `wxNOT_FOUND` if the dialog has no progress bar.
- wxString GetMessage () const

  Returns the last message passed to the Update() function; if you always passed wxEmptyString to Update() then the message set through the constructor is returned.
- virtual bool Pulse (const wxString &newmsg=wxEmptyString, bool *skip=NULL)

  Like Update() but makes the gauge control run in indeterminate mode.
- void Resume ()

  Can be used to continue with the dialog, after the user had clicked the "Abort" button.
- void SetRange (int maximum)

  Changes the maximum value of the progress meter given in the constructor.
- bool WasCancelled () const

Returns true if the "Cancel" button was pressed.

- bool WasSkipped () const

  Returns true if the "Skip" button was pressed.

- virtual bool Update (int value, const wxString &newmsg=wxEmptyString, bool *skip=NULL)

  Updates the dialog, setting the progress bar to the new value and updating the message if new one is specified.

Additional Inherited Members

21.283.2 Constructor & Destructor Documentation

wxGenericProgressDialog::wxGenericProgressDialog (const wxString & *title*, const wxString & *message*, int *maximum* = 100, wxWindow * *parent* = NULL, int *style* = wxPD_AUTO_HIDE|wxPD_APP_MODAL)

Constructor.

Creates the dialog, displays it and disables user input for other windows, or, if wxPD_APP_MODAL flag is not given, for its parent window only.

Parameters

title	Dialog title to show in titlebar.
message	Message displayed above the progress bar.
maximum	Maximum value for the progress bar. In the generic implementation the progress bar is constructed only if this value is greater than zero.
parent	Parent window.
style	The dialog style. See wxProgressDialog.

virtual wxGenericProgressDialog::~wxGenericProgressDialog ( ) [virtual]

Destructor.

Deletes the dialog and enables all top level windows.

21.283.3 Member Function Documentation

wxString wxGenericProgressDialog::GetMessage ( ) const

Returns the last message passed to the Update() function; if you always passed wxEmptyString to Update() then the message set through the constructor is returned.

Since

2.9.0

int wxGenericProgressDialog::GetRange ( ) const

Returns the maximum value of the progress meter, as passed to the constructor or wxNOT_FOUND if the dialog has no progress bar.

Since

2.9.0

int wxGenericProgressDialog::GetValue ( ) const

Returns the last value passed to the Update() function or `wxNOT_FOUND` if the dialog has no progress bar.

Since

2.9.0

virtual bool wxGenericProgressDialog::Pulse (const wxString & *newmsg* = wxEmptyString, bool * *skip* = `NULL`) `[virtual]`

Like Update() but makes the gauge control run in indeterminate mode.

In indeterminate mode the remaining and the estimated time labels (if present) are set to "Unknown" or to *newmsg* (if it's non-empty). Each call to this function moves the progress bar a bit to indicate that some progress was done.

See also

wxGauge::Pulse(), Update()

void wxGenericProgressDialog::Resume ( )

Can be used to continue with the dialog, after the user had clicked the "Abort" button.

void wxGenericProgressDialog::SetRange (int *maximum*)

Changes the maximum value of the progress meter given in the constructor.

This function can only be called (with a positive value) if the value passed in the constructor was positive.

Since

2.9.1

virtual bool wxGenericProgressDialog::Update (int *value*, const wxString & *newmsg* = wxEmptyString, bool * *skip* = `NULL`) `[virtual]`

Updates the dialog, setting the progress bar to the new value and updating the message if new one is specified.

Returns true unless the "Cancel" button has been pressed.

If false is returned, the application can either immediately destroy the dialog or ask the user for the confirmation and if the abort is not confirmed the dialog may be resumed with Resume() function.

If *value* is the maximum value for the dialog, the behaviour of the function depends on whether `wxPD_AUTO_HIDE` was used when the dialog was created. If it was, the dialog is hidden and the function returns immediately. If it was not, the dialog becomes a modal dialog and waits for the user to dismiss it, meaning that this function does not return until this happens.

Notice that you may want to call Fit() to change the dialog size to conform to the length of the new message if desired. The dialog does not do this automatically.

Parameters

value	The new value of the progress meter. It should be less than or equal to the maximum value given to the constructor.
newmsg	The new messages for the progress dialog text, if it is empty (which is the default) the message is not changed.
skip	If "Skip" button was pressed since last Update() call, this is set to true.

bool wxGenericProgressDialog::WasCancelled ( ) const

Returns true if the "Cancel" button was pressed.

Normally a Cancel button press is indicated by Update() returning false but sometimes it may be more convenient to check if the dialog was cancelled from elsewhere in the code and this function allows to do it.

It always returns false if the Cancel button is not shown at all.

Since

2.9.1

bool wxGenericProgressDialog::WasSkipped ( ) const

Returns true if the "Skip" button was pressed.

This is similar to WasCancelled() but returns true if the "Skip" button was pressed, not the "Cancel" one.

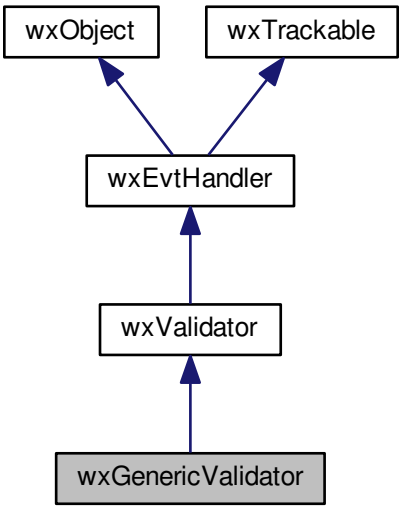
Since

2.9.1

21.284 wxGenericValidator Class Reference

`#include <wx/valgen.h>`

Inheritance diagram for wxGenericValidator:



21.284.1 Detailed Description

wxGenericValidator performs data transfer (but not validation or filtering) for many type of controls.

wxGenericValidator supports:

- wxButton, wxRadioButton, wxToggleButton, wxBitmapToggleButton, wxSpinButton
- wxCheckBox, wxRadioBox, wxComboBox, wxListBox, wxCheckListBox
- wxGauge, wxSlider, wxScrollBar, wxChoice, wxStaticText
- wxSpinCtrl, wxTextCtrl

It checks the type of the window and uses an appropriate type for it. For example, wxButton and wxTextCtrl transfer data to and from a wxString variable; wxListBox uses a wxArrayInt; wxCheckBox uses a boolean.

For more information, please see wxValidator Overview.

Library: wxCore

Category: Validators

See also

wxValidator Overview, wxValidator, wxTextValidator, wxIntegerValidator, wxFloatingPointValidator

Public Member Functions

- wxGenericValidator (const wxGenericValidator &validator)

Copy constructor.

- wxGenericValidator (bool ∗valPtr)

  Constructor taking a bool pointer.
- wxGenericValidator (wxString ∗valPtr)

  Constructor taking a wxString pointer.
- wxGenericValidator (int ∗valPtr)

  Constructor taking an integer pointer.
- wxGenericValidator (wxArrayInt ∗valPtr)

  Constructor taking a wxArrayInt pointer.
- wxGenericValidator (wxDateTime ∗valPtr)

  Constructor taking a wxDateTime pointer.
- wxGenericValidator (wxFileName ∗valPtr)

  Constructor taking a wxFileName pointer.
- wxGenericValidator (float ∗valPtr)

  Constructor taking a float pointer.
- wxGenericValidator (double ∗valPtr)

  Constructor taking a double pointer.
- virtual ∼wxGenericValidator ()

  Destructor.
- virtual wxObject ∗ Clone () const

  Clones the generic validator using the copy constructor.
- virtual bool TransferFromWindow ()

  Transfers the value from the window to the appropriate data type.
- virtual bool TransferToWindow ()

  Transfers the value to the window.

Additional Inherited Members

21.284.2 Constructor & Destructor Documentation

wxGenericValidator::wxGenericValidator (const wxGenericValidator & *validator*)

Copy constructor.

Parameters

validator	Validator to copy.

wxGenericValidator::wxGenericValidator (bool ∗ *valPtr*)

Constructor taking a bool pointer.

This will be used for wxCheckBox, wxRadioButton, wxToggleButton and wxBitmapToggleButton.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

wxGenericValidator::wxGenericValidator (wxString ∗ *valPtr*)

Constructor taking a wxString pointer.

This will be used for wxButton, wxComboBox, wxStaticText, wxTextCtrl.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

wxGenericValidator::wxGenericValidator (int * *valPtr*)

Constructor taking an integer pointer.

This will be used for wxChoice, wxGauge, wxScrollBar, wxRadioBox, wxSlider, wxSpinButton and wxSpinCtrl.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

wxGenericValidator::wxGenericValidator (wxArrayInt * *valPtr*)

Constructor taking a wxArrayInt pointer.

This will be used for wxListBox, wxCheckListBox.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

wxGenericValidator::wxGenericValidator (wxDateTime * *valPtr*)

Constructor taking a wxDateTime pointer.

This will be used for wxDatePickerCtrl.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

wxGenericValidator::wxGenericValidator (wxFileName * *valPtr*)

Constructor taking a wxFileName pointer.

This will be used for wxTextCtrl.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

Since

2.9.3

wxGenericValidator::wxGenericValidator (float * *valPtr*)

Constructor taking a float pointer.

This will be used for wxTextCtrl.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

Since

2.9.3

wxGenericValidator::wxGenericValidator (double * *valPtr*)

Constructor taking a double pointer.

This will be used for wxTextCtrl.

Parameters

valPtr	A pointer to a variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

Since

2.9.3

virtual wxGenericValidator::~wxGenericValidator ( ) [virtual]

Destructor.

21.284.3 Member Function Documentation

virtual wxObject* wxGenericValidator::Clone ( ) const [virtual]

Clones the generic validator using the copy constructor.

Reimplemented from wxValidator.

virtual bool wxGenericValidator::TransferFromWindow ( ) [virtual]

Transfers the value from the window to the appropriate data type.

Reimplemented from wxValidator.

virtual bool wxGenericValidator::TransferToWindow ( ) [virtual]

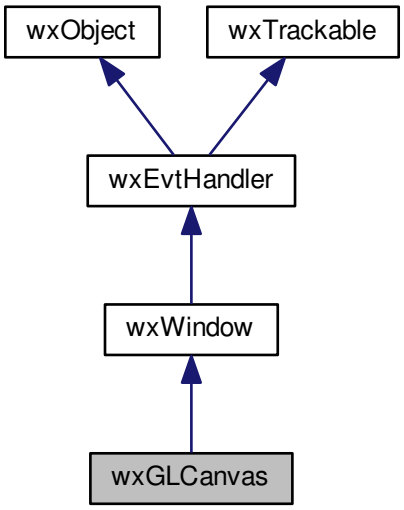
Transfers the value to the window.

Reimplemented from wxValidator.

21.285 wxGLCanvas Class Reference

```
#include <wx/glcanvas.h>
```

Inheritance diagram for wxGLCanvas:



21.285.1 Detailed Description

wxGLCanvas is a class for displaying OpenGL graphics.

It is always used in conjunction with wxGLContext as the context can only be made current (i.e. active for the OpenGL commands) when it is associated to a wxGLCanvas.

More precisely, you first need to create a wxGLCanvas window and then create an instance of a wxGLContext that is initialized with this wxGLCanvas and then later use either SetCurrent() with the instance of the wxGLContext or wxGLContext::SetCurrent() with the instance of the wxGLCanvas (which might be not the same as was used for the creation of the context) to bind the OpenGL state that is represented by the rendering context to the canvas, and then finally call SwapBuffers() to swap the buffers of the OpenGL canvas and thus show your current output.

Notice that versions of wxWidgets previous to 2.9 used to implicitly create a wxGLContext inside wxGLCanvas itself. This is still supported in the current version but is deprecated now and will be removed in the future, please update your code to create the rendering contexts explicitly.

To set up the attributes for the canvas (number of bits for the depth buffer, number of bits for the stencil buffer and so on) you should set up the correct values of the *attribList* parameter. The values that should be set up and their meanings will be described below.

Note

On those platforms which use a configure script (e.g. Linux and Mac OS) OpenGL support is automatically enabled if the relative headers and libraries are found. To switch it on under the other platforms (e.g. Windows), you need to edit the `setup.h` file and set `wxUSE_GLCANVAS` to `1` and then also pass `USE_OPENGL=1` to the make utility. You may also need to add `opengl32.lib` and `glu32.lib` to the list of the libraries your program is linked with.

Library: wxGL

Category: OpenGL

See also

wxGLContext

Public Member Functions

- wxGLCanvas (wxWindow *parent, wxWindowID id=wxID_ANY, const int *attribList=NULL, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name="GL↩ Canvas", const wxPalette &palette=wxNullPalette)

  Creates a window with the given parameters.
- bool SetColour (const wxString &colour)

  Sets the current colour for this window (using `glcolor3f()`), using the wxWidgets colour database to find a named colour.
- bool SetCurrent (const wxGLContext &context) const

  Makes the OpenGL state that is represented by the OpenGL rendering context context current, i.e.
- virtual bool SwapBuffers ()

  Swaps the double-buffer of this window, making the back-buffer the front-buffer and vice versa, so that the output of the previous OpenGL commands is displayed on the window.

Static Public Member Functions

- static bool IsDisplaySupported (const int *attribList)

  Determines if a canvas having the specified attributes is available.
- static bool IsExtensionSupported (const char *extension)

  Returns true if the extension with given name is supported.

Additional Inherited Members

21.285.2 Constructor & Destructor Documentation

wxGLCanvas::wxGLCanvas (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const int * *attribList* = `NULL`, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `"GLCanvas"`, const wxPalette & *palette* = wxNullPalette)

Creates a window with the given parameters.

Notice that you need to create and use a wxGLContext to output to this window.

If *attribList* is not specified, double buffered RGBA mode is used.

Parameters

parent	Pointer to a parent window.
id	Window identifier. If -1, will automatically create an identifier.
pos	Window position. wxDefaultPosition is (-1, -1) which indicates that wxWidgets should generate a default position for the window.
size	Window size. wxDefaultSize is (-1, -1) which indicates that wxWidgets should generate a default size for the window. If no suitable size can be found, the window will be sized to 20x20 pixels so that the window is visible but obviously not correctly sized.
style	Window style.
name	Window name.

attribList	Array of integers. With this parameter you can set the device context attributes associated to this window. This array is zero-terminated: it should be set up using wxGL_FLAGS constants. If a constant should be followed by a value, put it in the next array position. For example, WX_GL_DEPTH_SIZE should be followed by the value that indicates the number of bits for the depth buffer, e.g.:
	`attribList[n++] = WX_GL_DEPTH_SIZE;`
	`attribList[n++] = 32;`
	`attribList[n] = 0; // terminate the list`
	If the attribute list is not specified at all, i.e. if this parameter is NULL, the default attributes including WX_GL_RGBA and WX_GL_DOUBLEBUFFER are used. But notice that if you do specify some attributes you also need to explicitly include these two default attributes in the list if you need them.

palette	Palette for indexed colour (i.e. non WX_GL_RGBA) mode. Ignored under most platforms.

21.285.3 Member Function Documentation

static bool wxGLCanvas::IsDisplaySupported (const int * *attribList*) [static]

Determines if a canvas having the specified attributes is available.

Parameters

attribList	See *attribList* for wxGLCanvas().

Returns

true if attributes are supported.

static bool wxGLCanvas::IsExtensionSupported (const char * *extension*) [static]

Returns true if the extension with given name is supported.

Notice that while this function is implemented for all of GLX, WGL and AGL the extensions names are usually not the same for different platforms and so the code using it still usually uses conditional compilation.

bool wxGLCanvas::SetColour (const wxString & *colour*)

Sets the current colour for this window (using `glcolor3f()`), using the wxWidgets colour database to find a named colour.

bool wxGLCanvas::SetCurrent (const wxGLContext & *context*) const

Makes the OpenGL state that is represented by the OpenGL rendering context *context* current, i.e.

it will be used by all subsequent OpenGL calls.

This is equivalent to wxGLContext::SetCurrent() called with this window as parameter.

Note

This function may only be called when the window is shown on screen, in particular it can't usually be called from the constructor as the window isn't yet shown at this moment.

Returns

false if an error occurred.

virtual bool wxGLCanvas::SwapBuffers ( ) [virtual]

Swaps the double-buffer of this window, making the back-buffer the front-buffer and vice versa, so that the output of the previous OpenGL commands is displayed on the window.

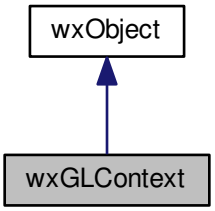
Returns

false if an error occurred.

21.286 wxGLContext Class Reference

`#include <wx/glcanvas.h>`

Inheritance diagram for wxGLContext:



21.286.1 Detailed Description

An instance of a wxGLContext represents the state of an OpenGL state machine and the connection between OpenGL and the system.

The OpenGL state includes everything that can be set with the OpenGL API: colors, rendering variables, display lists, texture objects, etc. Although it is possible to have multiple rendering contexts share display lists in order to save resources, this method is hardly used today any more, because display lists are only a tiny fraction of the overall state.

Therefore, one rendering context is usually used with or bound to multiple output windows in turn, so that the application has access to the complete and identical state while rendering into each window.

Binding (making current) a rendering context with another instance of a wxGLCanvas however works only if the other wxGLCanvas was created with the same attributes as the wxGLCanvas from which the wxGLContext was initialized. (This applies to sharing display lists among contexts analogously.)

Note that some wxGLContext features are extremely platform-specific - its best to check your native platform's glcanvas header (on windows include/wx/msw/glcanvas.h) to see what features your native platform provides.

wxHAS_OPENGL_ES is defined on platforms that only have this implementation available (eg the iPhone) und don't support the full specification.

Library: wxGL

Category: OpenGL

See also

wxGLCanvas

Public Member Functions

- wxGLContext (wxGLCanvas *win, const wxGLContext *other=NULL)

  Constructor.

- virtual bool SetCurrent (const wxGLCanvas &win) const

  Makes the OpenGL state that is represented by this rendering context current with the wxGLCanvas win.

Additional Inherited Members

21.286.2 Constructor & Destructor Documentation

wxGLContext::wxGLContext (wxGLCanvas * *win,* const wxGLContext * *other* = NULL)

Constructor.

Parameters

win	The canvas that is used to initialize this context. This parameter is needed only temporarily, and the caller may do anything with it (e.g. destroy the window) after the constructor returned. It will be possible to bind (make current) this context to any other wxGLCanvas that has been created with equivalent attributes as win.
other	Context to share display lists with or NULL (the default) for no sharing.

21.286.3 Member Function Documentation

virtual bool wxGLContext::SetCurrent (const wxGLCanvas & *win*) const [virtual]

Makes the OpenGL state that is represented by this rendering context current with the wxGLCanvas *win*.

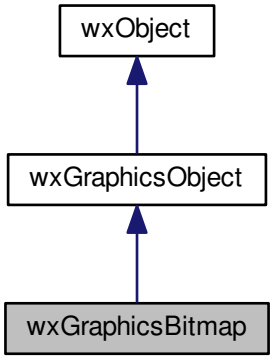
Note

win can be a different wxGLCanvas window than the one that was passed to the constructor of this rendering context. If *RC* is an object of type wxGLContext, the statements *"RC.SetCurrent(win);"* and *"win.SetCurrent(RC);"* are equivalent, see wxGLCanvas::SetCurrent().

21.287 wxGraphicsBitmap Class Reference

`#include <wx/graphics.h>`

Inheritance diagram for wxGraphicsBitmap:



21.287.1 Detailed Description

Represents a bitmap.

The objects of this class are not created directly but only via wxGraphicsContext or wxGraphicsRenderer CreateBitmap(), CreateBitmapFromImage() or CreateSubBitmap() methods. They can subsequently be used with wxGraphicsContext::DrawBitmap(). The only other operation is testing for the bitmap validity which can be performed using IsNull() method inherited from the base class.

Public Member Functions

- wxGraphicsBitmap ()

  Default constructor creates an invalid bitmap.
- wxImage ConvertToImage () const

  Return the contents of this bitmap as wxImage.
- void * GetNativeBitmap () const

  Return the pointer to the native bitmap data.

Additional Inherited Members

21.287.2 Constructor & Destructor Documentation

wxGraphicsBitmap::wxGraphicsBitmap ( ) [inline]

Default constructor creates an invalid bitmap.

21.287.3 Member Function Documentation

wxImage wxGraphicsBitmap::ConvertToImage ( ) const

Return the contents of this bitmap as wxImage.

Using this method is more efficient than converting wxGraphicsBitmap to wxBitmap first and then to wxImage and can be useful if, for example, you want to save wxGraphicsBitmap as a disk file in a format not directly supported by wxBitmap.

Invalid image is returned if the bitmap is invalid.

Since

2.9.3

void* wxGraphicsBitmap::GetNativeBitmap ( ) const

Return the pointer to the native bitmap data.

(CGImageRef for Core Graphics, cairo_surface_t for Cairo, Bitmap* for GDI+.)

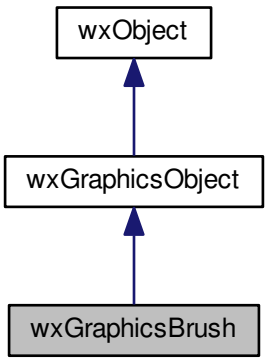
Since

2.9.4

21.288 wxGraphicsBrush Class Reference

`#include <wx/graphics.h>`

Inheritance diagram for wxGraphicsBrush:



21.288.1 Detailed Description

A wxGraphicsBrush is a native representation of a brush.

The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via wxGraphicsContext::CreateBrush() or wxGraphicsRenderer::CreateBrush().

Library: wxCore

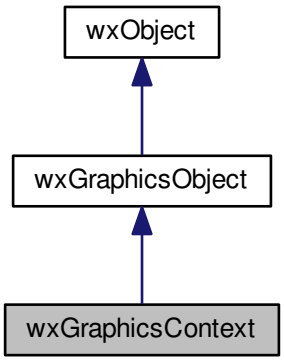
Category: Graphics Device Interface (GDI)

Additional Inherited Members

21.289 wxGraphicsContext Class Reference

`#include <wx/graphics.h>`

Inheritance diagram for wxGraphicsContext:



21.289.1 Detailed Description

A wxGraphicsContext instance is the object that is drawn upon.

It is created by a renderer using wxGraphicsRenderer::CreateContext(). This can be either directly using a renderer instance, or indirectly using the static convenience Create() functions of wxGraphicsContext that always delegate the task to the default renderer.

```
void MyCanvas::OnPaint(wxPaintEvent &event)
{
    // Create paint DC
    wxPaintDC dc(this);

    // Create graphics context from it
    wxGraphicsContext *gc = wxGraphicsContext::Create( dc );

    if (gc)
    {
```

```
        // make a path that contains a circle and some lines
        gc->SetPen( *wxRED_PEN );
        wxGraphicsPath path = gc->CreatePath();
        path.AddCircle( 50.0, 50.0, 50.0 );
        path.MoveToPoint(0.0, 50.0);
        path.AddLineToPoint(100.0, 50.0);
        path.MoveToPoint(50.0, 0.0);
        path.AddLineToPoint(50.0, 100.0 );
        path.CloseSubpath();
        path.AddRectangle(25.0, 25.0, 50.0, 50.0);

        gc->StrokePath(path);

        delete gc;
    }
}
```

Library: wxCore

Category: Graphics Device Interface (GDI), Device Contexts

See also

wxGraphicsRenderer::CreateContext(), wxGCDC, wxDC

Public Member Functions

- virtual void Clip (const wxRegion &region)=0

  Clips drawings to the specified region.
- virtual void Clip (wxDouble x, wxDouble y, wxDouble w, wxDouble h)=0

  Clips drawings to the specified rectangle.
- virtual void ConcatTransform (const wxGraphicsMatrix &matrix)=0

  Concatenates the passed in transform with the current transform of this context.
- virtual wxGraphicsBitmap CreateBitmap (const wxBitmap &bitmap)=0

  Creates wxGraphicsBitmap from an existing wxBitmap.
- virtual wxGraphicsBitmap CreateBitmapFromImage (const wxImage &image)

  Creates wxGraphicsBitmap from an existing wxImage.
- virtual wxGraphicsBitmap CreateSubBitmap (const wxGraphicsBitmap &bitmap, wxDouble x, wxDouble y, wxDouble w, wxDouble h)=0

  Extracts a sub-bitmap from an existing bitmap.
- virtual wxGraphicsBrush CreateBrush (const wxBrush &brush) const

  Creates a native brush from a wxBrush.
- virtual wxGraphicsFont CreateFont (const wxFont &font, const wxColour &col=∗wxBLACK) const

  Creates a native graphics font from a wxFont and a text colour.
- virtual wxGraphicsFont CreateFont (double sizeInPixels, const wxString &facename, int flags=wxFONTFL↩AG_DEFAULT, const wxColour &col=∗wxBLACK) const

  Creates a font object with the specified attributes.
- virtual wxGraphicsMatrix CreateMatrix (wxDouble a=1.0, wxDouble b=0.0, wxDouble c=0.0, wxDouble d=1.0, wxDouble tx=0.0, wxDouble ty=0.0) const

  Creates a native affine transformation matrix from the passed in values.
- wxGraphicsMatrix CreateMatrix (const wxAffineMatrix2DBase &mat) const

  Creates a native affine transformation matrix from the passed generic one.
- wxGraphicsPath CreatePath () const

  Creates a native graphics path which is initially empty.
- virtual wxGraphicsPen CreatePen (const wxPen &pen) const

  Creates a native pen from a wxPen.
- virtual void DrawEllipse (wxDouble x, wxDouble y, wxDouble w, wxDouble h)

Draws an ellipse.

- virtual void DrawIcon (const wxIcon &icon, wxDouble x, wxDouble y, wxDouble w, wxDouble h)=0

  Draws the icon.

- virtual void DrawLines (size_t n, const wxPoint2DDouble *points, wxPolygonFillMode fillStyle=wxODDEVEN_RULE)

  Draws a polygon.

- virtual void DrawPath (const wxGraphicsPath &path, wxPolygonFillMode fillStyle=wxODDEVEN_RULE)

  Draws the path by first filling and then stroking.

- virtual void DrawRectangle (wxDouble x, wxDouble y, wxDouble w, wxDouble h)

  Draws a rectangle.

- virtual void DrawRoundedRectangle (wxDouble x, wxDouble y, wxDouble w, wxDouble h, wxDouble radius)

  Draws a rounded rectangle.

- void DrawText (const wxString &str, wxDouble x, wxDouble y)

  Draws text at the defined position.

- void DrawText (const wxString &str, wxDouble x, wxDouble y, wxDouble angle)

  Draws text at the defined position.

- void DrawText (const wxString &str, wxDouble x, wxDouble y, const wxGraphicsBrush &backgroundBrush)

  Draws text at the defined position.

- void DrawText (const wxString &str, wxDouble x, wxDouble y, wxDouble angle, const wxGraphicsBrush &backgroundBrush)

  Draws text at the defined position.

- virtual void FillPath (const wxGraphicsPath &path, wxPolygonFillMode fillStyle=wxODDEVEN_RULE)=0

  Fills the path with the current brush.

- virtual void * GetNativeContext ()=0

  Returns the native context (CGContextRef for Core Graphics, Graphics pointer for GDIPlus and cairo_t pointer for cairo).

- virtual void GetPartialTextExtents (const wxString &text, wxArrayDouble &widths) const =0

  Fills the widths array with the widths from the beginning of text to the corresponding character of text.

- virtual void GetTextExtent (const wxString &text, wxDouble *width, wxDouble *height, wxDouble *descent, wxDouble *externalLeading) const =0

  Gets the dimensions of the string using the currently selected font.

- virtual wxGraphicsMatrix GetTransform () const =0

  Gets the current transformation matrix of this context.

- virtual void ResetClip ()=0

  Resets the clipping to original shape.

- virtual void Rotate (wxDouble angle)=0

  Rotates the current transformation matrix (in radians).

- virtual void Scale (wxDouble xScale, wxDouble yScale)=0

  Scales the current transformation matrix.

- void SetBrush (const wxBrush &brush)

  Sets the brush for filling paths.

- virtual void SetBrush (const wxGraphicsBrush &brush)

  Sets the brush for filling paths.

- void SetFont (const wxFont &font, const wxColour &colour)

  Sets the font for drawing text.

- virtual void SetFont (const wxGraphicsFont &font)

  Sets the font for drawing text.

- void SetPen (const wxPen &pen)

  Sets the pen used for stroking.

- virtual void SetPen (const wxGraphicsPen &pen)

  Sets the pen used for stroking.

- virtual void SetTransform (const wxGraphicsMatrix &matrix)=0

  Sets the current transformation matrix of this context.
- virtual void StrokeLine (wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2)

  Strokes a single line.
- virtual void StrokeLines (size_t n, const wxPoint2DDouble *beginPoints, const wxPoint2DDouble *endPoints)

  Stroke disconnected lines from begin to end points, fastest method available for this purpose.
- virtual void StrokeLines (size_t n, const wxPoint2DDouble *points)

  Stroke lines connecting all the points.
- virtual void StrokePath (const wxGraphicsPath &path)=0

  Strokes along a path with the current pen.
- virtual void Translate (wxDouble dx, wxDouble dy)=0

  Translates the current transformation matrix.
- virtual void BeginLayer (wxDouble opacity)=0

  Redirects all rendering is done into a fully transparent temporary context.
- virtual void EndLayer ()=0

  Composites back the drawings into the context with the opacity given at the BeginLayer call.
- virtual bool SetAntialiasMode (wxAntialiasMode antialias)=0

  Sets the antialiasing mode, returns true if it supported.
- virtual wxAntialiasMode GetAntialiasMode () const

  Returns the current shape antialiasing mode.
- virtual bool SetInterpolationQuality (wxInterpolationQuality interpolation)=0

  Sets the interpolation quality, returns true if it is supported.
- virtual wxInterpolationQuality GetInterpolationQuality () const

  Returns the current interpolation quality.
- virtual bool SetCompositionMode (wxCompositionMode op)=0

  Sets the compositing operator, returns true if it supported.
- virtual wxCompositionMode GetCompositionMode () const

  Returns the current compositing operator.
- virtual void PushState ()=0

  Push the current state of the context's transformation matrix on a stack.
- virtual void PopState ()=0

  Pops a stored state from the stack and sets the current transformation matrix to that state.
- virtual bool ShouldOffset () const
- virtual void EnableOffset (bool enable=true)
- void DisableOffset ()
- bool OffsetEnabled ()
- virtual bool StartDoc (const wxString &message)

  Begin a new document (relevant only for printing / pdf etc.) If there is a progress dialog, message will be shown.
- virtual void EndDoc ()

  Done with that document (relevant only for printing / pdf etc.)
- virtual void StartPage (wxDouble width=0, wxDouble height=0)

  Opens a new page (relevant only for printing / pdf etc.) with the given size in points.
- virtual void EndPage ()

  Ends the current page (relevant only for printing / pdf etc.)
- virtual void Flush ()

  Make sure that the current content of this context is immediately visible.
- void GetSize (wxDouble *width, wxDouble *height) const

  Returns the size of the graphics context in device coordinates.
- virtual void GetDPI (wxDouble *dpiX, wxDouble *dpiY)

  Returns the resolution of the graphics context in device points per inch.

- wxGraphicsBrush CreateLinearGradientBrush (wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, const wxColour &c1, const wxColour &c2) const

  Creates a native brush with a linear gradient.
- wxGraphicsBrush CreateLinearGradientBrush (wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, const wxGraphicsGradientStops &stops) const

  Creates a native brush with a linear gradient.

- virtual wxGraphicsBrush CreateRadialGradientBrush (wxDouble xo, wxDouble yo, wxDouble xc, wxDouble yc, wxDouble radius, const wxColour &oColor, const wxColour &cColor) const

  Creates a native brush with a radial gradient.
- virtual wxGraphicsBrush CreateRadialGradientBrush (wxDouble xo, wxDouble yo, wxDouble xc, wxDouble yc, wxDouble radius, const wxGraphicsGradientStops &stops)=0

  Creates a native brush with a radial gradient.

- virtual void DrawBitmap (const wxGraphicsBitmap &bmp, wxDouble x, wxDouble y, wxDouble w, wxDouble h)=0

  Draws the bitmap.
- virtual void DrawBitmap (const wxBitmap &bmp, wxDouble x, wxDouble y, wxDouble w, wxDouble h)=0

  Draws the bitmap.

Static Public Member Functions

- static wxGraphicsContext * Create (wxWindow *window)

  Creates a wxGraphicsContext from a wxWindow.
- static wxGraphicsContext * Create (const wxWindowDC &windowDC)

  Creates a wxGraphicsContext from a wxWindowDC.
- static wxGraphicsContext * Create (const wxMemoryDC &memoryDC)

  Creates a wxGraphicsContext from a wxMemoryDC.
- static wxGraphicsContext * Create (const wxPrinterDC &printerDC)

  Creates a wxGraphicsContext from a wxPrinterDC.
- static wxGraphicsContext * Create (const wxEnhMetaFileDC &metaFileDC)

  Creates a wxGraphicsContext from a wxEnhMetaFileDC.
- static wxGraphicsContext * Create (wxImage &image)

  Creates a wxGraphicsContext associated with a wxImage.
- static wxGraphicsContext * Create ()

  Create a lightweight context that can be used only for measuring text.
- static wxGraphicsContext * CreateFromNative (void *context)

  Creates a wxGraphicsContext from a native context.
- static wxGraphicsContext * CreateFromNativeWindow (void *window)

  Creates a wxGraphicsContext from a native window.

Additional Inherited Members

21.289.2 Member Function Documentation

virtual void wxGraphicsContext::BeginLayer (wxDouble *opacity*) `[pure virtual]`

Redirects all rendering is done into a fully transparent temporary context.

virtual void wxGraphicsContext::Clip (const wxRegion & *region*) `[pure virtual]`

Clips drawings to the specified region.

virtual void wxGraphicsContext::Clip (wxDouble *x,* wxDouble *y,* wxDouble *w,* wxDouble *h*) `[pure virtual]`

Clips drawings to the specified rectangle.

virtual void wxGraphicsContext::ConcatTransform (const wxGraphicsMatrix & *matrix*) `[pure virtual]`

Concatenates the passed in transform with the current transform of this context.

static wxGraphicsContext* wxGraphicsContext::Create (wxWindow * *window*) `[static]`

Creates a wxGraphicsContext from a wxWindow.

See also

wxGraphicsRenderer::CreateContext()

static wxGraphicsContext* wxGraphicsContext::Create (const wxWindowDC & *windowDC*) `[static]`

Creates a wxGraphicsContext from a wxWindowDC.

See also

wxGraphicsRenderer::CreateContext()

static wxGraphicsContext* wxGraphicsContext::Create (const wxMemoryDC & *memoryDC*) `[static]`

Creates a wxGraphicsContext from a wxMemoryDC.

See also

wxGraphicsRenderer::CreateContext()

static wxGraphicsContext* wxGraphicsContext::Create (const wxPrinterDC & *printerDC*) `[static]`

Creates a wxGraphicsContext from a wxPrinterDC.

Under GTK+, this will only work when using the GtkPrint printing backend which is available since GTK+ 2.10.

See also

wxGraphicsRenderer::CreateContext(), Printing Under Unix (GTK+)

static wxGraphicsContext* wxGraphicsContext::Create (const wxEnhMetaFileDC & *metaFileDC*) `[static]`

Creates a wxGraphicsContext from a wxEnhMetaFileDC.

This function, as wxEnhMetaFileDC class itself, is only available only under MSW.

See also

wxGraphicsRenderer::CreateContext()

static wxGraphicsContext* wxGraphicsContext::Create (wxImage & *image*) `[static]`

Creates a wxGraphicsContext associated with a wxImage.

The image specifies the size of the context as well as whether alpha is supported (if wxImage::HasAlpha()) or not and the initial contents of the context. The *image* object must have a life time greater than that of the new context as the context copies its contents back to the image when it is destroyed.

Since

2.9.3

static wxGraphicsContext* wxGraphicsContext::Create ( ) `[static]`

Create a lightweight context that can be used only for measuring text.

virtual wxGraphicsBitmap wxGraphicsContext::CreateBitmap (const wxBitmap & *bitmap*) `[pure virtual]`

Creates wxGraphicsBitmap from an existing wxBitmap.

Returns an invalid wxNullGraphicsBitmap on failure.

virtual wxGraphicsBitmap wxGraphicsContext::CreateBitmapFromImage (const wxImage & *image*) `[virtual]`

Creates wxGraphicsBitmap from an existing wxImage.

This method is more efficient than converting wxImage to wxBitmap first and then calling CreateBitmap() but otherwise has the same effect.

Returns an invalid wxNullGraphicsBitmap on failure.

Since

2.9.3

virtual wxGraphicsBrush wxGraphicsContext::CreateBrush (const wxBrush & *brush*) const `[virtual]`

Creates a native brush from a wxBrush.

virtual wxGraphicsFont wxGraphicsContext::CreateFont (const wxFont & *font,* const wxColour & *col* = *wxBLACK) const `[virtual]`

Creates a native graphics font from a wxFont and a text colour.

virtual wxGraphicsFont wxGraphicsContext::CreateFont (double *sizeInPixels,* const wxString & *facename,* int *flags* = wxFONTFLAG_DEFAULT, const wxColour & *col* = *wxBLACK) const `[virtual]`

Creates a font object with the specified attributes.

The use of overload taking wxFont is preferred, see wxGraphicsRenderer::CreateFont() for more details.

Since

2.9.3

static wxGraphicsContext* wxGraphicsContext::CreateFromNative (void * *context*) [static]

Creates a wxGraphicsContext from a native context.

This native context must be a CGContextRef for Core Graphics, a Graphics pointer for GDIPlus, or a cairo_t pointer for cairo.

See also

wxGraphicsRenderer::CreateContextFromNativeContext()

static wxGraphicsContext* wxGraphicsContext::CreateFromNativeWindow (void * *window*) [static]

Creates a wxGraphicsContext from a native window.

See also

wxGraphicsRenderer::CreateContextFromNativeWindow()

wxGraphicsBrush wxGraphicsContext::CreateLinearGradientBrush (wxDouble *x1*, wxDouble *y1*, wxDouble *x2*, wxDouble *y2*, const wxColour & *c1*, const wxColour & *c2*) const

Creates a native brush with a linear gradient.

The brush starts at (*x1*, *y1*) and ends at (*x2*, *y2*). Either just the start and end gradient colours (*c1* and *c2*) or full set of gradient *stops* can be specified.

The version taking wxGraphicsGradientStops is new in wxWidgets 2.9.1.

wxGraphicsBrush wxGraphicsContext::CreateLinearGradientBrush (wxDouble *x1*, wxDouble *y1*, wxDouble *x2*, wxDouble *y2*, const wxGraphicsGradientStops & *stops*) const

Creates a native brush with a linear gradient.

The brush starts at (*x1*, *y1*) and ends at (*x2*, *y2*). Either just the start and end gradient colours (*c1* and *c2*) or full set of gradient *stops* can be specified.

The version taking wxGraphicsGradientStops is new in wxWidgets 2.9.1.

virtual wxGraphicsMatrix wxGraphicsContext::CreateMatrix (wxDouble *a* = 1.0, wxDouble *b* = 0.0, wxDouble *c* = 0.0, wxDouble *d* = 1.0, wxDouble *tx* = 0.0, wxDouble *ty* = 0.0) const [virtual]

Creates a native affine transformation matrix from the passed in values.

The default parameters result in an identity matrix.

wxGraphicsMatrix wxGraphicsContext::CreateMatrix (const wxAffineMatrix2DBase & *mat*) const

Creates a native affine transformation matrix from the passed generic one.

Since

2.9.4

wxGraphicsPath wxGraphicsContext::CreatePath ( ) const

Creates a native graphics path which is initially empty.

virtual wxGraphicsPen wxGraphicsContext::CreatePen (const wxPen & *pen*) const [virtual]

Creates a native pen from a wxPen.

virtual wxGraphicsBrush wxGraphicsContext::CreateRadialGradientBrush (wxDouble *xo*, wxDouble *yo*, wxDouble *xc*, wxDouble *yc*, wxDouble *radius*, const wxColour & *oColor*, const wxColour & *cColor*) const [virtual]

Creates a native brush with a radial gradient.

The brush originates at (*xo*, *yc*) and ends on a circle around (*xc*, *yc*) with the given *radius*.

The gradient may be specified either by its start and end colours *oColor* and *cColor* or by a full set of gradient *stops*.

The version taking wxGraphicsGradientStops is new in wxWidgets 2.9.1.

virtual wxGraphicsBrush wxGraphicsContext::CreateRadialGradientBrush (wxDouble *xo*, wxDouble *yo*, wxDouble *xc*, wxDouble *yc*, wxDouble *radius*, const wxGraphicsGradientStops & *stops*) [pure virtual]

Creates a native brush with a radial gradient.

The brush originates at (*xo*, *yc*) and ends on a circle around (*xc*, *yc*) with the given *radius*.

The gradient may be specified either by its start and end colours *oColor* and *cColor* or by a full set of gradient *stops*.

The version taking wxGraphicsGradientStops is new in wxWidgets 2.9.1.

virtual wxGraphicsBitmap wxGraphicsContext::CreateSubBitmap (const wxGraphicsBitmap & *bitmap*, wxDouble *x*, wxDouble *y*, wxDouble *w*, wxDouble *h*) [pure virtual]

Extracts a sub-bitmap from an existing bitmap.

Currently this function is implemented in the native MSW and OS X versions but not when using Cairo.

void wxGraphicsContext::DisableOffset ( )

virtual void wxGraphicsContext::DrawBitmap (const wxGraphicsBitmap & *bmp*, wxDouble *x*, wxDouble *y*, wxDouble *w*, wxDouble *h*) [pure virtual]

Draws the bitmap.

In case of a mono bitmap, this is treated as a mask and the current brushed is used for filling.

virtual void wxGraphicsContext::DrawBitmap (const wxBitmap & *bmp*, wxDouble *x*, wxDouble *y*, wxDouble *w*, wxDouble *h*) [pure virtual]

Draws the bitmap.

In case of a mono bitmap, this is treated as a mask and the current brushed is used for filling.

virtual void wxGraphicsContext::DrawEllipse (wxDouble *x*, wxDouble *y*, wxDouble *w*, wxDouble *h*) [virtual]

Draws an ellipse.

virtual void wxGraphicsContext::DrawIcon (const wxIcon & *icon*, wxDouble *x*, wxDouble *y*, wxDouble *w*, wxDouble *h*) [pure virtual]

Draws the icon.

virtual void wxGraphicsContext::DrawLines (size_t *n,* const wxPoint2DDouble * *points,* wxPolygonFillMode *fillStyle =* wxODDEVEN_RULE) [virtual]

Draws a polygon.

virtual void wxGraphicsContext::DrawPath (const wxGraphicsPath & *path,* wxPolygonFillMode *fillStyle =* wxODDEVEN_RULE) [virtual]

Draws the path by first filling and then stroking.

virtual void wxGraphicsContext::DrawRectangle (wxDouble *x,* wxDouble *y,* wxDouble *w,* wxDouble *h*) [virtual]

Draws a rectangle.

virtual void wxGraphicsContext::DrawRoundedRectangle (wxDouble *x,* wxDouble *y,* wxDouble *w,* wxDouble *h,* wxDouble *radius*) [virtual]

Draws a rounded rectangle.

void wxGraphicsContext::DrawText (const wxString & *str,* wxDouble *x,* wxDouble *y*)

Draws text at the defined position.

void wxGraphicsContext::DrawText (const wxString & *str,* wxDouble *x,* wxDouble *y,* wxDouble *angle*)

Draws text at the defined position.

Parameters

str	The text to draw.
x	The x coordinate position to draw the text at.
y	The y coordinate position to draw the text at.
angle	The angle relative to the (default) horizontal direction to draw the string.

void wxGraphicsContext::DrawText (const wxString & *str,* wxDouble *x,* wxDouble *y,* const wxGraphicsBrush & *backgroundBrush*)

Draws text at the defined position.

Parameters

str	The text to draw.
x	The x coordinate position to draw the text at.
y	The y coordinate position to draw the text at.
backgroundBrush	Brush to fill the text with.

void wxGraphicsContext::DrawText (const wxString & *str,* wxDouble *x,* wxDouble *y,* wxDouble *angle,* const wxGraphicsBrush & *backgroundBrush*)

Draws text at the defined position.

Parameters

str	The text to draw.
x	The x coordinate position to draw the text at.
y	The y coordinate position to draw the text at.
angle	The angle relative to the (default) horizontal direction to draw the string.
background↩ Brush	Brush to fill the text with.

virtual void wxGraphicsContext::EnableOffset (bool *enable* = `true`) [virtual]

virtual void wxGraphicsContext::EndDoc ( ) [virtual]

Done with that document (relevant only for printing / pdf etc.)

virtual void wxGraphicsContext::EndLayer ( ) [pure virtual]

Composites back the drawings into the context with the opacity given at the BeginLayer call.

virtual void wxGraphicsContext::EndPage ( ) [virtual]

Ends the current page (relevant only for printing / pdf etc.)

virtual void wxGraphicsContext::FillPath (const wxGraphicsPath & *path*, wxPolygonFillMode *fillStyle* = wxODDEVEN_RULE) [pure virtual]

Fills the path with the current brush.

virtual void wxGraphicsContext::Flush ( ) [virtual]

Make sure that the current content of this context is immediately visible.

virtual wxAntialiasMode wxGraphicsContext::GetAntialiasMode ( ) const [virtual]

Returns the current shape antialiasing mode.

virtual wxCompositionMode wxGraphicsContext::GetCompositionMode ( ) const [virtual]

Returns the current compositing operator.

virtual void wxGraphicsContext::GetDPI (wxDouble * *dpiX*, wxDouble * *dpiY*) [virtual]

Returns the resolution of the graphics context in device points per inch.

virtual wxInterpolationQuality wxGraphicsContext::GetInterpolationQuality ( ) const [virtual]

Returns the current interpolation quality.

virtual void* wxGraphicsContext::GetNativeContext ( ) [pure virtual]

Returns the native context (CGContextRef for Core Graphics, Graphics pointer for GDIPlus and cairo_t pointer for cairo).

virtual void wxGraphicsContext::GetPartialTextExtents (const wxString & *text*, wxArrayDouble & *widths*) const [pure virtual]

Fills the *widths* array with the widths from the beginning of *text* to the corresponding character of *text*.

void wxGraphicsContext::GetSize (wxDouble * *width*, wxDouble * *height*) const

Returns the size of the graphics context in device coordinates.

virtual void wxGraphicsContext::GetTextExtent (const wxString & *text*, wxDouble * *width*, wxDouble * *height*, wxDouble * *descent*, wxDouble * *externalLeading*) const [pure virtual]

Gets the dimensions of the string using the currently selected font.

Parameters

text	The text string to measure.
width	Variable to store the total calculated width of the text.
height	Variable to store the total calculated height of the text.
descent	Variable to store the dimension from the baseline of the font to the bottom of the descender.
externalLeading	Any extra vertical space added to the font by the font designer (usually is zero).

virtual wxGraphicsMatrix wxGraphicsContext::GetTransform ( ) const [pure virtual]

Gets the current transformation matrix of this context.

bool wxGraphicsContext::OffsetEnabled ( )

virtual void wxGraphicsContext::PopState ( ) [pure virtual]

Pops a stored state from the stack and sets the current transformation matrix to that state.

See also

wxGraphicsContext::PopState

virtual void wxGraphicsContext::PushState ( ) [pure virtual]

Push the current state of the context's transformation matrix on a stack.

See also

wxGraphicsContext::PopState

virtual void wxGraphicsContext::ResetClip ( ) [pure virtual]

Resets the clipping to original shape.

virtual void wxGraphicsContext::Rotate (wxDouble *angle*) `[pure virtual]`

Rotates the current transformation matrix (in radians).

virtual void wxGraphicsContext::Scale (wxDouble *xScale,* wxDouble *yScale*) `[pure virtual]`

Scales the current transformation matrix.

virtual bool wxGraphicsContext::SetAntialiasMode (wxAntialiasMode *antialias*) `[pure virtual]`

Sets the antialiasing mode, returns true if it supported.

void wxGraphicsContext::SetBrush (const wxBrush & *brush*)

Sets the brush for filling paths.

virtual void wxGraphicsContext::SetBrush (const wxGraphicsBrush & *brush*) `[virtual]`

Sets the brush for filling paths.

virtual bool wxGraphicsContext::SetCompositionMode (wxCompositionMode *op*) `[pure virtual]`

Sets the compositing operator, returns true if it supported.

void wxGraphicsContext::SetFont (const wxFont & *font,* const wxColour & *colour*)

Sets the font for drawing text.

virtual void wxGraphicsContext::SetFont (const wxGraphicsFont & *font*) `[virtual]`

Sets the font for drawing text.

virtual bool wxGraphicsContext::SetInterpolationQuality (wxInterpolationQuality *interpolation*) `[pure virtual]`

Sets the interpolation quality, returns true if it is supported.
Not implemented in Cairo backend currently.

void wxGraphicsContext::SetPen (const wxPen & *pen*)

Sets the pen used for stroking.

virtual void wxGraphicsContext::SetPen (const wxGraphicsPen & *pen*) `[virtual]`

Sets the pen used for stroking.

virtual void wxGraphicsContext::SetTransform (const wxGraphicsMatrix & *matrix*) `[pure virtual]`

Sets the current transformation matrix of this context.

virtual bool wxGraphicsContext::ShouldOffset ( ) const [virtual]

virtual bool wxGraphicsContext::StartDoc (const wxString & *message*) [virtual]

Begin a new document (relevant only for printing / pdf etc.) If there is a progress dialog, message will be shown.

virtual void wxGraphicsContext::StartPage (wxDouble *width* = 0, wxDouble *height* = 0) [virtual]

Opens a new page (relevant only for printing / pdf etc.) with the given size in points.

(If both are null the default page size will be used.)

virtual void wxGraphicsContext::StrokeLine (wxDouble *x1*, wxDouble *y1*, wxDouble *x2*, wxDouble *y2*) [virtual]

Strokes a single line.

virtual void wxGraphicsContext::StrokeLines (size_t *n*, const wxPoint2DDouble * *beginPoints*, const wxPoint2DDouble * *endPoints*) [virtual]

Stroke disconnected lines from begin to end points, fastest method available for this purpose.

virtual void wxGraphicsContext::StrokeLines (size_t *n*, const wxPoint2DDouble * *points*) [virtual]

Stroke lines connecting all the points.

Unlike the other overload of this function, this method draws a single polyline and not a number of disconnected lines.

virtual void wxGraphicsContext::StrokePath (const wxGraphicsPath & *path*) [pure virtual]

Strokes along a path with the current pen.

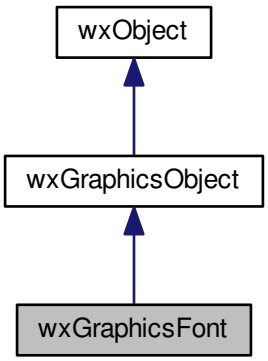
virtual void wxGraphicsContext::Translate (wxDouble *dx*, wxDouble *dy*) [pure virtual]

Translates the current transformation matrix.

21.290 wxGraphicsFont Class Reference

```
#include <wx/graphics.h>
```

Inheritance diagram for wxGraphicsFont:



21.290.1 Detailed Description

A wxGraphicsFont is a native representation of a font.

The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via wxGraphicsContext::CreateFont() or wxGraphicsRenderer::CreateFont().

Library: wxCore

Category: Graphics Device Interface (GDI)

Additional Inherited Members

21.291 wxGraphicsGradientStop Class Reference

```
#include <wx/graphics.h>
```

21.291.1 Detailed Description

Represents a single gradient stop in a collection of gradient stops as represented by wxGraphicsGradientStops.

Library: wxCore

Category: Graphics Device Interface (GDI)

Since

2.9.1

Public Member Functions

- wxGraphicsGradientStop (wxColour col=wxTransparentColour, float pos=0.)

  Creates a stop with the given colour and position.
- const wxColour & GetColour () const

  Return the stop colour.
- void SetColour (const wxColour &col)

  Change the stop colour.
- float GetPosition () const

  Return the stop position.
- void SetPosition (float pos)

  Change the stop position.

21.291.2 Constructor & Destructor Documentation

wxGraphicsGradientStop::wxGraphicsGradientStop (wxColour *col* = wxTransparentColour, float *pos* = 0.)

Creates a stop with the given colour and position.

Parameters

col	The colour of this stop. Note that the alpha component of the colour is honoured thus allowing the background colours to partially show through the gradient.
pos	The stop position, must be in [0, 1] range with 0 being the beginning and 1 the end of the gradient.

21.291.3 Member Function Documentation

const wxColour& wxGraphicsGradientStop::GetColour ( ) const

Return the stop colour.

float wxGraphicsGradientStop::GetPosition ( ) const

Return the stop position.

void wxGraphicsGradientStop::SetColour (const wxColour & *col*)

Change the stop colour.

Parameters

col	The new colour.

void wxGraphicsGradientStop::SetPosition (float *pos*)

Change the stop position.

Parameters

pos	The new position, must always be in [0, 1] range.

21.292 wxGraphicsGradientStops Class Reference

`#include <wx/graphics.h>`

21.292.1 Detailed Description

Represents a collection of wxGraphicGradientStop values for use with CreateLinearGradientBrush and CreateRadialGradientBrush.

The stops are maintained in order of position. If two or more stops are added with the same position then the one(s) added later come later. This can be useful for producing discontinuities in the colour gradient.

Notice that this class is write-once, you can't modify the stops once they had been added.

Library: wxCore

Category: Graphics Device Interface (GDI)

Since

2.9.1

Public Member Functions

- wxGraphicsGradientStops (wxColour startCol=wxTransparentColour, wxColour endCol=wxTransparentColour)

  Initializes the gradient stops with the given boundary colours.
- wxGraphicsGradientStop Item (unsigned n) const

  Returns the stop at the given index.
- size_t GetCount () const

  Returns the number of stops.
- void SetStartColour (wxColour col)

  Set the start colour to col.
- wxColour GetStartColour () const

  Returns the start colour.
- void SetEndColour (wxColour col)

  Set the end colour to col.
- wxColour GetEndColour () const

  Returns the end colour.

- void Add (const wxGraphicsGradientStop &stop)

  Add a new stop.
- void Add (wxColour col, float pos)

  Add a new stop.

21.292.2 Constructor & Destructor Documentation

wxGraphicsGradientStops::wxGraphicsGradientStops (wxColour *startCol* = wxTransparentColour, wxColour *endCol* = wxTransparentColour)

Initializes the gradient stops with the given boundary colours.

Creates a wxGraphicsGradientStops instance with start colour given by *startCol* and end colour given by *endCol*.

21.292.3 Member Function Documentation

void wxGraphicsGradientStops::Add (const wxGraphicsGradientStop & *stop*)

Add a new stop.

void wxGraphicsGradientStops::Add (wxColour *col*, float *pos*)

Add a new stop.

size_t wxGraphicsGradientStops::GetCount ( ) const

Returns the number of stops.

wxColour wxGraphicsGradientStops::GetEndColour ( ) const

Returns the end colour.

wxColour wxGraphicsGradientStops::GetStartColour ( ) const

Returns the start colour.

wxGraphicsGradientStop wxGraphicsGradientStops::Item (unsigned *n*) const

Returns the stop at the given index.

Parameters

n	The index, must be in [0, GetCount()) range.

void wxGraphicsGradientStops::SetEndColour (wxColour *col*)

Set the end colour to *col*.

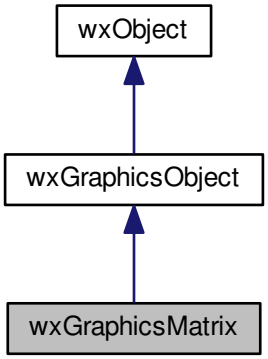
void wxGraphicsGradientStops::SetStartColour (wxColour *col*)

Set the start colour to *col*.

21.293 wxGraphicsMatrix Class Reference

```
#include <wx/graphics.h>
```

Inheritance diagram for wxGraphicsMatrix:



21.293.1 Detailed Description

A wxGraphicsMatrix is a native representation of an affine matrix.

The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via wxGraphicsContext::CreateMatrix() or wxGraphicsRenderer::CreateMatrix().

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- virtual void Concat (const wxGraphicsMatrix *t)

  Concatenates the matrix passed with the current matrix.
- void Concat (const wxGraphicsMatrix &t)

  Concatenates the matrix passed with the current matrix.
- virtual void Get (wxDouble *a=NULL, wxDouble *b=NULL, wxDouble *c=NULL, wxDouble *d=NULL, wxDouble *tx=NULL, wxDouble *ty=NULL) const

  Returns the component values of the matrix via the argument pointers.
- virtual void * GetNativeMatrix () const

  Returns the native representation of the matrix.
- virtual void Invert ()

  Inverts the matrix.
- virtual bool IsEqual (const wxGraphicsMatrix *t) const

  Returns true if the elements of the transformation matrix are equal.
- bool IsEqual (const wxGraphicsMatrix &t) const

  Returns true if the elements of the transformation matrix are equal.
- virtual bool IsIdentity () const

  Return true if this is the identity matrix.

- virtual void Rotate (wxDouble angle)

  Rotates this matrix clockwise (in radians).
- virtual void Scale (wxDouble xScale, wxDouble yScale)

  Scales this matrix.
- virtual void Set (wxDouble a=1.0, wxDouble b=0.0, wxDouble c=0.0, wxDouble d=1.0, wxDouble tx=0.0, wxDouble ty=0.0)

  Sets the matrix to the respective values (default values are the identity matrix).
- virtual void TransformDistance (wxDouble *dx, wxDouble *dy) const

  Applies this matrix to a distance (ie.
- virtual void TransformPoint (wxDouble *x, wxDouble *y) const

  Applies this matrix to a point.
- virtual void Translate (wxDouble dx, wxDouble dy)

  Translates this matrix.

Additional Inherited Members

21.293.2 Member Function Documentation

virtual void wxGraphicsMatrix::Concat (const wxGraphicsMatrix * *t*) [virtual]

Concatenates the matrix passed with the current matrix.

void wxGraphicsMatrix::Concat (const wxGraphicsMatrix & *t*)

Concatenates the matrix passed with the current matrix.

virtual void wxGraphicsMatrix::Get (wxDouble * *a* = NULL, wxDouble * *b* = NULL, wxDouble * *c* = NULL, wxDouble * *d* = NULL, wxDouble * *tx* = NULL, wxDouble * *ty* = NULL) const [virtual]

Returns the component values of the matrix via the argument pointers.

virtual void* wxGraphicsMatrix::GetNativeMatrix ( ) const [virtual]

Returns the native representation of the matrix.

For CoreGraphics this is a CFAffineMatrix pointer, for GDIPlus a Matrix Pointer, and for Cairo a cairo_matrix_t pointer.

virtual void wxGraphicsMatrix::Invert ( ) [virtual]

Inverts the matrix.

virtual bool wxGraphicsMatrix::IsEqual (const wxGraphicsMatrix * *t*) const [virtual]

Returns true if the elements of the transformation matrix are equal.

bool wxGraphicsMatrix::IsEqual (const wxGraphicsMatrix & *t*) const

Returns true if the elements of the transformation matrix are equal.

virtual bool wxGraphicsMatrix::IsIdentity ( ) const [virtual]

Return true if this is the identity matrix.

virtual void wxGraphicsMatrix::Rotate (wxDouble *angle*) [virtual]

Rotates this matrix clockwise (in radians).

Parameters

angle	Rotation angle in radians, clockwise.

virtual void wxGraphicsMatrix::Scale (wxDouble *xScale,* wxDouble *yScale*) [virtual]

Scales this matrix.

virtual void wxGraphicsMatrix::Set (wxDouble *a* = 1.0, wxDouble *b* = 0.0, wxDouble *c* = 0.0, wxDouble *d* = 1.0, wxDouble *tx* = 0.0, wxDouble *ty* = 0.0) [virtual]

Sets the matrix to the respective values (default values are the identity matrix).

virtual void wxGraphicsMatrix::TransformDistance (wxDouble * *dx,* wxDouble * *dy*) const [virtual]

Applies this matrix to a distance (ie.

performs all transforms except translations).

virtual void wxGraphicsMatrix::TransformPoint (wxDouble * *x,* wxDouble * *y*) const [virtual]

Applies this matrix to a point.

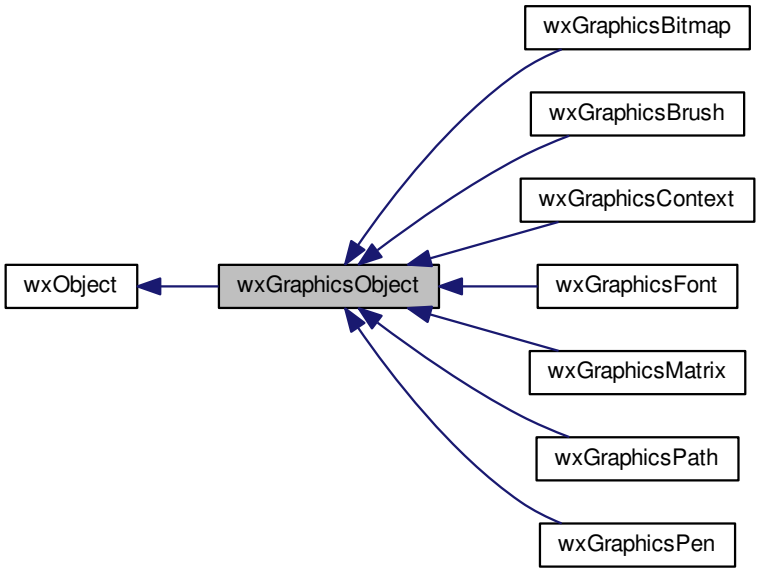
virtual void wxGraphicsMatrix::Translate (wxDouble *dx,* wxDouble *dy*) [virtual]

Translates this matrix.

21.294 wxGraphicsObject Class Reference

`#include <wx/graphics.h>`

Inheritance diagram for wxGraphicsObject:



21.294.1 Detailed Description

This class is the superclass of native graphics objects like pens etc.

It allows reference counting. Not instantiated by user code.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxGraphicsBrush, wxGraphicsPen, wxGraphicsMatrix, wxGraphicsPath

Public Member Functions

- wxGraphicsRenderer * GetRenderer () const

  Returns the renderer that was used to create this instance, or NULL if it has not been initialized yet.
- bool IsNull () const

Additional Inherited Members

21.294.2 Member Function Documentation

wxGraphicsRenderer* wxGraphicsObject::GetRenderer ( ) const

Returns the renderer that was used to create this instance, or NULL if it has not been initialized yet.

bool wxGraphicsObject::IsNull ( ) const

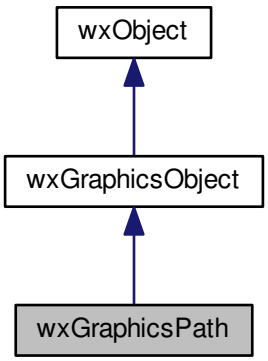
Returns

false if this object is valid, otherwise returns true.

21.295 wxGraphicsPath Class Reference

`#include <wx/graphics.h>`

Inheritance diagram for wxGraphicsPath:



21.295.1 Detailed Description

A wxGraphicsPath is a native representation of a geometric path.

The contents are specific and private to the respective renderer. Instances are reference counted and can therefore be assigned as usual. The only way to get a valid instance is by using wxGraphicsContext::CreatePath() or wxGraphicsRenderer::CreatePath().

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- virtual void AddArcToPoint (wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, wxDouble r)

  Appends a an arc to two tangents connecting (current) to (x1,y1) and (x1,y1) to (x2,y2), also a straight line from (current) to (x1,y1).
- virtual void AddCircle (wxDouble x, wxDouble y, wxDouble r)

  Appends a circle around (x,y) with radius r as a new closed subpath.
- virtual void AddCurveToPoint (wxDouble cx1, wxDouble cy1, wxDouble cx2, wxDouble cy2, wxDouble x, wxDouble y)

  Adds a cubic bezier curve from the current point, using two control points and an end point.

- void AddCurveToPoint (const wxPoint2DDouble &c1, const wxPoint2DDouble &c2, const wxPoint2DDouble &e)

  Adds a cubic bezier curve from the current point, using two control points and an end point.
- virtual void AddEllipse (wxDouble x, wxDouble y, wxDouble w, wxDouble h)

  Appends an ellipse fitting into the passed in rectangle.
- virtual void AddLineToPoint (wxDouble x, wxDouble y)

  Adds a straight line from the current point to (x,y).
- void AddLineToPoint (const wxPoint2DDouble &p)

  Adds a straight line from the current point to p.
- virtual void AddPath (const wxGraphicsPath &path)

  Adds another path.
- virtual void AddQuadCurveToPoint (wxDouble cx, wxDouble cy, wxDouble x, wxDouble y)

  Adds a quadratic bezier curve from the current point, using a control point and an end point.
- virtual void AddRectangle (wxDouble x, wxDouble y, wxDouble w, wxDouble h)

  Appends a rectangle as a new closed subpath.
- virtual void AddRoundedRectangle (wxDouble x, wxDouble y, wxDouble w, wxDouble h, wxDouble radius)

  Appends a rounded rectangle as a new closed subpath.
- virtual void CloseSubpath ()

  Closes the current sub-path.
- bool Contains (const wxPoint2DDouble &c, wxPolygonFillMode fillStyle=wxODDEVEN_RULE) const
- virtual bool Contains (wxDouble x, wxDouble y, wxPolygonFillMode fillStyle=wxODDEVEN_RULE) const
- wxRect2DDouble GetBox () const

  Gets the bounding box enclosing all points (possibly including control points).
- virtual void GetBox (wxDouble *x, wxDouble *y, wxDouble *w, wxDouble *h) const

  Gets the bounding box enclosing all points (possibly including control points).
- virtual void GetCurrentPoint (wxDouble *x, wxDouble *y) const

  Gets the last point of the current path, (0,0) if not yet set.
- wxPoint2DDouble GetCurrentPoint () const

  Gets the last point of the current path, (0,0) if not yet set.
- virtual void * GetNativePath () const

  Returns the native path (CGPathRef for Core Graphics, Path pointer for GDIPlus and a cairo_path_t pointer for cairo).
- virtual void MoveToPoint (wxDouble x, wxDouble y)

  Begins a new subpath at (x,y).
- void MoveToPoint (const wxPoint2DDouble &p)

  Begins a new subpath at p.
- virtual void Transform (const wxGraphicsMatrix &matrix)

  Transforms each point of this path by the matrix.
- virtual void UnGetNativePath (void *p) const

  Gives back the native path returned by GetNativePath() because there might be some deallocations necessary (e.g.

- virtual void AddArc (wxDouble x, wxDouble y, wxDouble r, wxDouble startAngle, wxDouble endAngle, bool clockwise)

  Adds an arc of a circle.
- void AddArc (const wxPoint2DDouble &c, wxDouble r, wxDouble startAngle, wxDouble endAngle, bool clockwise)

  Adds an arc of a circle.

Additional Inherited Members

21.295.2 Member Function Documentation

virtual void wxGraphicsPath::AddArc (wxDouble *x,* wxDouble *y,* wxDouble *r,* wxDouble *startAngle,* wxDouble *endAngle,* bool *clockwise*) [virtual]

Adds an arc of a circle.

The circle is defined by the coordinates of its centre (*x*, *y*) or *c* and its radius *r*. The arc goes from the starting angle *startAngle* to *endAngle* either clockwise or counter-clockwise depending on the value of *clockwise* argument.

The angles are measured in radians but, contrary to the usual mathematical convention, are always *clockwise* from the horizontal axis.

void wxGraphicsPath::AddArc (const wxPoint2DDouble & *c,* wxDouble *r,* wxDouble *startAngle,* wxDouble *endAngle,* bool *clockwise*)

Adds an arc of a circle.

The circle is defined by the coordinates of its centre (*x*, *y*) or *c* and its radius *r*. The arc goes from the starting angle *startAngle* to *endAngle* either clockwise or counter-clockwise depending on the value of *clockwise* argument.

The angles are measured in radians but, contrary to the usual mathematical convention, are always *clockwise* from the horizontal axis.

virtual void wxGraphicsPath::AddArcToPoint (wxDouble *x1,* wxDouble *y1,* wxDouble *x2,* wxDouble *y2,* wxDouble *r*) [virtual]

Appends a an arc to two tangents connecting (current) to (*x1*,*y1*) and (*x1*,*y1*) to (*x2*,*y2*), also a straight line from (current) to (*x1*,*y1*).

virtual void wxGraphicsPath::AddCircle (wxDouble *x,* wxDouble *y,* wxDouble *r*) [virtual]

Appends a circle around (*x*,*y*) with radius *r* as a new closed subpath.

virtual void wxGraphicsPath::AddCurveToPoint (wxDouble *cx1,* wxDouble *cy1,* wxDouble *cx2,* wxDouble *cy2,* wxDouble *x,* wxDouble *y*) [virtual]

Adds a cubic bezier curve from the current point, using two control points and an end point.

void wxGraphicsPath::AddCurveToPoint (const wxPoint2DDouble & *c1,* const wxPoint2DDouble & *c2,* const wxPoint2DDouble & *e*)

Adds a cubic bezier curve from the current point, using two control points and an end point.

virtual void wxGraphicsPath::AddEllipse (wxDouble *x,* wxDouble *y,* wxDouble *w,* wxDouble *h*) [virtual]

Appends an ellipse fitting into the passed in rectangle.

virtual void wxGraphicsPath::AddLineToPoint (wxDouble *x,* wxDouble *y*) [virtual]

Adds a straight line from the current point to (*x*,*y*).

void wxGraphicsPath::AddLineToPoint (const wxPoint2DDouble & *p*)

Adds a straight line from the current point to *p*.

virtual void wxGraphicsPath::AddPath (const wxGraphicsPath & *path*) `[virtual]`

Adds another path.

virtual void wxGraphicsPath::AddQuadCurveToPoint (wxDouble *cx,* wxDouble *cy,* wxDouble *x,* wxDouble *y*) `[virtual]`

Adds a quadratic bezier curve from the current point, using a control point and an end point.

virtual void wxGraphicsPath::AddRectangle (wxDouble *x,* wxDouble *y,* wxDouble *w,* wxDouble *h*) `[virtual]`

Appends a rectangle as a new closed subpath.

virtual void wxGraphicsPath::AddRoundedRectangle (wxDouble *x,* wxDouble *y,* wxDouble *w,* wxDouble *h,* wxDouble *radius*) `[virtual]`

Appends a rounded rectangle as a new closed subpath.

virtual void wxGraphicsPath::CloseSubpath ( ) `[virtual]`

Closes the current sub-path.

bool wxGraphicsPath::Contains (const wxPoint2DDouble & *c,* wxPolygonFillMode *fillStyle* = wxODDEVEN_RULE) const

Returns

true if the point is within the path.

virtual bool wxGraphicsPath::Contains (wxDouble *x,* wxDouble *y,* wxPolygonFillMode *fillStyle* = wxODDEVEN_RULE) const `[virtual]`

Returns

true if the point is within the path.

wxRect2DDouble wxGraphicsPath::GetBox ( ) const

Gets the bounding box enclosing all points (possibly including control points).

virtual void wxGraphicsPath::GetBox (wxDouble * *x,* wxDouble * *y,* wxDouble * *w,* wxDouble * *h*) const `[virtual]`

Gets the bounding box enclosing all points (possibly including control points).

virtual void wxGraphicsPath::GetCurrentPoint (wxDouble * *x*, wxDouble * *y*) const [virtual]

Gets the last point of the current path, (0,0) if not yet set.

wxPoint2DDouble wxGraphicsPath::GetCurrentPoint ( ) const

Gets the last point of the current path, (0,0) if not yet set.

virtual void* wxGraphicsPath::GetNativePath ( ) const [virtual]

Returns the native path (CGPathRef for Core Graphics, Path pointer for GDIPlus and a cairo_path_t pointer for cairo).

virtual void wxGraphicsPath::MoveToPoint (wxDouble *x*, wxDouble *y*) [virtual]

Begins a new subpath at (*x*,*y*).

void wxGraphicsPath::MoveToPoint (const wxPoint2DDouble & *p*)

Begins a new subpath at *p*.

virtual void wxGraphicsPath::Transform (const wxGraphicsMatrix & *matrix*) [virtual]

Transforms each point of this path by the matrix.

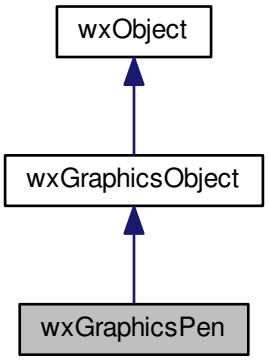
virtual void wxGraphicsPath::UnGetNativePath (void * *p*) const [virtual]

Gives back the native path returned by GetNativePath() because there might be some deallocations necessary (e.g. on cairo the native path returned by GetNativePath() is newly allocated each time).

21.296 wxGraphicsPen Class Reference

`#include <wx/graphics.h>`

Inheritance diagram for wxGraphicsPen:



21.296.1 Detailed Description

A wxGraphicsPen is a native representation of a pen.

The contents are specific and private to the respective renderer. Instances are ref counted and can therefore be assigned as usual. The only way to get a valid instance is via wxGraphicsContext::CreatePen() or wxGraphics↩ Renderer::CreatePen().

Library: wxCore

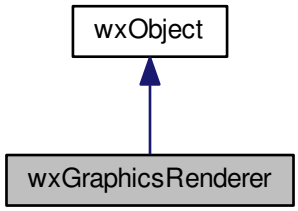
Category: Graphics Device Interface (GDI)

Additional Inherited Members

21.297 wxGraphicsRenderer Class Reference

```
#include <wx/graphics.h>
```

Inheritance diagram for wxGraphicsRenderer:



21.297.1 Detailed Description

A wxGraphicsRenderer is the instance corresponding to the rendering engine used.

There may be multiple instances on a system, if there are different rendering engines present, but there is always only one instance per engine. This instance is pointed back to by all objects created by it (wxGraphicsContext, wxGraphicsPath etc) and can be retrieved through their wxGraphicsObject::GetRenderer() method. Therefore you can create an additional instance of a path etc. by calling wxGraphicsObject::GetRenderer() and then using the appropriate CreateXXX() function of that renderer.

```
wxGraphicsPath *path = // from somewhere
wxGraphicsBrush *brush = path->GetRenderer()->
      CreateBrush( *wxBLACK_BRUSH );
```

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- virtual wxGraphicsBitmap CreateBitmap (const wxBitmap &bitmap)=0

  Creates wxGraphicsBitmap from an existing wxBitmap.
- virtual wxGraphicsBitmap CreateBitmapFromImage (const wxImage &image)=0

  Creates wxGraphicsBitmap from an existing wxImage.
- virtual wxImage CreateImageFromBitmap (const wxGraphicsBitmap &bmp)=0

  Creates a wxImage from a wxGraphicsBitmap.
- virtual wxGraphicsBitmap CreateBitmapFromNativeBitmap (void *bitmap)=0

  Creates wxGraphicsBitmap from a native bitmap handle.
- virtual wxGraphicsContext * CreateContext (wxWindow *window)=0

  Creates a wxGraphicsContext from a wxWindow.
- virtual wxGraphicsContext * CreateContext (const wxWindowDC &windowDC)=0

  Creates a wxGraphicsContext from a wxWindowDC.
- virtual wxGraphicsContext * CreateContext (const wxMemoryDC &memoryDC)=0

  Creates a wxGraphicsContext from a wxMemoryDC.
- virtual wxGraphicsContext * CreateContext (const wxPrinterDC &printerDC)=0

  Creates a wxGraphicsContext from a wxPrinterDC.

- virtual wxGraphicsContext * CreateContext (const wxEnhMetaFileDC &metaFileDC)=0

  Creates a wxGraphicsContext from a wxEnhMetaFileDC.
- wxGraphicsContext * CreateContextFromImage (wxImage &image)

  Creates a wxGraphicsContext associated with a wxImage.
- virtual wxGraphicsBrush CreateBrush (const wxBrush &brush)=0

  Creates a native brush from a wxBrush.
- virtual wxGraphicsContext * CreateContextFromNativeContext (void *context)=0

  Creates a wxGraphicsContext from a native context.
- virtual wxGraphicsContext * CreateContextFromNativeWindow (void *window)=0

  Creates a wxGraphicsContext from a native window.
- virtual wxGraphicsContext * CreateMeasuringContext ()=0

  Creates a wxGraphicsContext that can be used for measuring texts only.
- virtual wxGraphicsFont CreateFont (const wxFont &font, const wxColour &col=*wxBLACK)=0

  Creates a native graphics font from a wxFont and a text colour.
- virtual wxGraphicsFont CreateFont (double sizeInPixels, const wxString &facename, int flags=wxFONTFLAG_DEFAULT, const wxColour &col=*wxBLACK)=0

  Creates a graphics font with the given characteristics.
- virtual wxGraphicsBrush CreateLinearGradientBrush (wxDouble x1, wxDouble y1, wxDouble x2, wxDouble y2, const wxGraphicsGradientStops &stops)=0

  Creates a native brush with a linear gradient.
- virtual wxGraphicsMatrix CreateMatrix (wxDouble a=1.0, wxDouble b=0.0, wxDouble c=0.0, wxDouble d=1.0, wxDouble tx=0.0, wxDouble ty=0.0)=0

  Creates a native affine transformation matrix from the passed in values.
- virtual wxGraphicsPath CreatePath ()=0

  Creates a native graphics path which is initially empty.
- virtual wxGraphicsPen CreatePen (const wxPen &pen)=0

  Creates a native pen from a wxPen.
- virtual wxGraphicsBrush CreateRadialGradientBrush (wxDouble xo, wxDouble yo, wxDouble xc, wxDouble yc, wxDouble radius, const wxGraphicsGradientStops &stops)=0

  Creates a native brush with a radial gradient.
- virtual wxGraphicsBitmap CreateSubBitmap (const wxGraphicsBitmap &bitmap, wxDouble x, wxDouble y, wxDouble w, wxDouble h)=0

  Extracts a sub-bitmap from an existing bitmap.
- virtual wxString GetName () const =0

  Returns the name of the technology used by the renderer.
- virtual void GetVersion (int *major, int *minor=NULL, int *micro=NULL) const =0

  Returns the version major, minor and micro/build of the technology used by the renderer.

Static Public Member Functions

- static wxGraphicsRenderer * GetDefaultRenderer ()

  Returns the default renderer on this platform.
- static wxGraphicsRenderer * GetCairoRenderer ()

Additional Inherited Members

21.297.2 Member Function Documentation

virtual wxGraphicsBitmap wxGraphicsRenderer::CreateBitmap (const wxBitmap & *bitmap*) `[pure virtual]`

Creates wxGraphicsBitmap from an existing wxBitmap.

Returns an invalid wxNullGraphicsBitmap on failure.

virtual wxGraphicsBitmap wxGraphicsRenderer::CreateBitmapFromImage (const wxImage & *image*) `[pure virtual]`

Creates wxGraphicsBitmap from an existing wxImage.

This method is more efficient than converting wxImage to wxBitmap first and then calling CreateBitmap() but otherwise has the same effect.

Returns an invalid wxNullGraphicsBitmap on failure.

Since

2.9.3

virtual wxGraphicsBitmap wxGraphicsRenderer::CreateBitmapFromNativeBitmap (void * *bitmap*) `[pure virtual]`

Creates wxGraphicsBitmap from a native bitmap handle.

bitmap meaning is platform-dependent. Currently it's a GDI+ `Bitmap` pointer under MSW, `CGImage` pointer under OS X or a `cairo_surface_t` pointer when using Cairo under any platform.

Notice that this method takes ownership of *bitmap*, i.e. it will be destroyed when the returned wxGraphicsBitmap is.

virtual wxGraphicsBrush wxGraphicsRenderer::CreateBrush (const wxBrush & *brush*) `[pure virtual]`

Creates a native brush from a wxBrush.

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContext (wxWindow * *window*) `[pure virtual]`

Creates a wxGraphicsContext from a wxWindow.

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContext (const wxWindowDC & *windowDC*) `[pure virtual]`

Creates a wxGraphicsContext from a wxWindowDC.

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContext (const wxMemoryDC & *memoryDC*) `[pure virtual]`

Creates a wxGraphicsContext from a wxMemoryDC.

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContext (const wxPrinterDC & *printerDC*) `[pure virtual]`

Creates a wxGraphicsContext from a wxPrinterDC.

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContext (const wxEnhMetaFileDC & *metaFileDC*) `[pure virtual]`

Creates a wxGraphicsContext from a wxEnhMetaFileDC.

This function, as wxEnhMetaFileDC class itself, is only available only under MSW.

wxGraphicsContext* wxGraphicsRenderer::CreateContextFromImage (wxImage & *image*)

Creates a wxGraphicsContext associated with a wxImage.

This function is used by wxContext::CreateFromImage() and is not normally called directly.

Since

2.9.3

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContextFromNativeContext (void * *context*) [pure virtual]

Creates a wxGraphicsContext from a native context.

This native context must be a CGContextRef for Core Graphics, a Graphics pointer for GDIPlus, or a cairo_t pointer for cairo.

virtual wxGraphicsContext* wxGraphicsRenderer::CreateContextFromNativeWindow (void * *window*) [pure virtual]

Creates a wxGraphicsContext from a native window.

virtual wxGraphicsFont wxGraphicsRenderer::CreateFont (const wxFont & *font,* const wxColour & *col* = *wxBLACK) [pure virtual]

Creates a native graphics font from a wxFont and a text colour.

virtual wxGraphicsFont wxGraphicsRenderer::CreateFont (double *sizeInPixels,* const wxString & *facename,* int *flags* = wxFONTFLAG_DEFAULT, const wxColour & *col* = *wxBLACK) [pure virtual]

Creates a graphics font with the given characteristics.

If possible, the CreateFont() overload taking wxFont should be used instead. The main advantage of this overload is that it can be used without X server connection under Unix when using Cairo.

Parameters

sizeInPixels	Height of the font in user space units, i.e. normally pixels. Notice that this is different from the overload taking wxFont as wxFont size is specified in points.
facename	The name of the font. The same font name might not be available under all platforms so the font name can also be empty to use the default platform font.
flags	Combination of wxFontFlag enum elements. Currently only `wxFONTFLAG_ITALIC` and `wxFONTFLAG_BOLD` are supported. By default the normal font version is used.
col	The font colour, black by default.

Since

2.9.3

virtual wxImage wxGraphicsRenderer::CreateImageFromBitmap (const wxGraphicsBitmap & *bmp*) [pure virtual]

Creates a wxImage from a wxGraphicsBitmap.

This method is used by the more convenient wxGraphicsBitmap::ConvertToImage.

virtual **wxGraphicsBrush** wxGraphicsRenderer::CreateLinearGradientBrush (**wxDouble** *x1,* **wxDouble** *y1,* **wxDouble** *x2,* **wxDouble** *y2,* const **wxGraphicsGradientStops** & *stops*) [pure virtual]

Creates a native brush with a linear gradient.

Stops support is new since wxWidgets 2.9.1, previously only the start and end colours could be specified.

virtual **wxGraphicsMatrix** wxGraphicsRenderer::CreateMatrix (**wxDouble** *a* = 1.0, **wxDouble** *b* = 0.0, **wxDouble** *c* = 0.0, **wxDouble** *d* = 1.0, **wxDouble** *tx* = 0.0, **wxDouble** *ty* = 0.0) [pure virtual]

Creates a native affine transformation matrix from the passed in values.

The defaults result in an identity matrix.

virtual **wxGraphicsContext**∗ wxGraphicsRenderer::CreateMeasuringContext ( ) [pure virtual]

Creates a wxGraphicsContext that can be used for measuring texts only.

No drawing commands are allowed.

virtual **wxGraphicsPath** wxGraphicsRenderer::CreatePath ( ) [pure virtual]

Creates a native graphics path which is initially empty.

virtual **wxGraphicsPen** wxGraphicsRenderer::CreatePen (const **wxPen** & *pen*) [pure virtual]

Creates a native pen from a wxPen.

virtual **wxGraphicsBrush** wxGraphicsRenderer::CreateRadialGradientBrush (**wxDouble** *xo,* **wxDouble** *yo,* **wxDouble** *xc,* **wxDouble** *yc,* **wxDouble** *radius,* const **wxGraphicsGradientStops** & *stops*) [pure virtual]

Creates a native brush with a radial gradient.

Stops support is new since wxWidgets 2.9.1, previously only the start and end colours could be specified.

virtual **wxGraphicsBitmap** wxGraphicsRenderer::CreateSubBitmap (const **wxGraphicsBitmap** & *bitmap,* **wxDouble** *x,* **wxDouble** *y,* **wxDouble** *w,* **wxDouble** *h*) [pure virtual]

Extracts a sub-bitmap from an existing bitmap.

Currently this function is implemented in the native MSW and OS X versions but not when using Cairo.

static **wxGraphicsRenderer**∗ wxGraphicsRenderer::GetCairoRenderer ( ) [static]

static **wxGraphicsRenderer**∗ wxGraphicsRenderer::GetDefaultRenderer ( ) [static]

Returns the default renderer on this platform.

On OS X this is the Core Graphics (a.k.a. Quartz 2D) renderer, on MSW the GDIPlus renderer, and on GTK we currently default to the cairo renderer.

virtual **wxString** wxGraphicsRenderer::GetName ( ) const [pure virtual]

Returns the name of the technology used by the renderer.

Currently this function returns "gdiplus" for Windows GDI+ implementation, "cairo" for Cairo implementation and "cg" for OS X CoreGraphics implementation.

Note: the string returned by this method is not user-readable and is expected to be used internally by the program only.

Since

3.1.0

virtual void wxGraphicsRenderer::GetVersion (int * *major*, int * *minor* = NULL, int * *micro* = NULL) const [pure virtual]

Returns the version major, minor and micro/build of the technology used by the renderer.

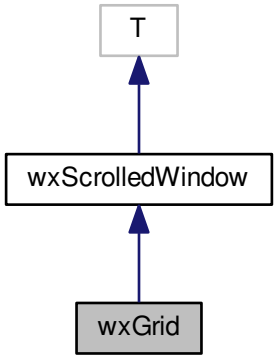
Currently this function returns the OS major and minor versions in the parameters with the matching names and sets *micro* to 0 for the GDI+ and CoreGraphics engines which are considered to be parts of their respective OS.

For Cairo, this is the major,minor,micro version of the Cairo library which is returned.

21.298 wxGrid Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGrid:



21.298.1 Detailed Description

wxGrid and its related classes are used for displaying and editing tabular data.

They provide a rich set of features for display, editing, and interacting with a variety of data sources. For simple applications, and to help you get started, wxGrid is the only class you need to refer to directly. It will set up default instances of the other classes and manage them for you. For more complex applications you can derive your own classes for custom grid views, grid data tables, cell editors and renderers. The wxGrid Overview has examples of simple and more complex applications, explains the relationship between the various grid classes and has a summary of the keyboard shortcuts and mouse functions provided by wxGrid.

A wxGridTableBase class holds the actual data to be displayed by a wxGrid class. One or more wxGrid classes may act as a view for one table class. The default table class is called wxGridStringTable and holds an array of strings. An instance of such a class is created by CreateGrid().

wxGridCellRenderer is the abstract base class for rendering contents in a cell. The following renderers are predefined:

- wxGridCellBoolRenderer
- wxGridCellFloatRenderer
- wxGridCellNumberRenderer
- wxGridCellStringRenderer

The look of a cell can be further defined using wxGridCellAttr. An object of this type may be returned by wxGrid↩ TableBase::GetAttr().

wxGridCellEditor is the abstract base class for editing the value of a cell. The following editors are predefined:

- wxGridCellBoolEditor
- wxGridCellChoiceEditor
- wxGridCellFloatEditor
- wxGridCellNumberEditor
- wxGridCellTextEditor

Please see wxGridEvent, wxGridSizeEvent, wxGridRangeSelectEvent, and wxGridEditorCreatedEvent for the documentation of all event types you can use with wxGrid.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGrid Overview, wxGridUpdateLocker

Public Types

- enum wxGridSelectionModes {
  wxGridSelectCells,
  wxGridSelectRows,
  wxGridSelectColumns,
  wxGridSelectRowsOrColumns }

  Different selection modes supported by the grid.
- enum CellSpan {
  CellSpan_Inside = -1,
  CellSpan_None = 0,
  CellSpan_Main }

  Return values for GetCellSize().
- enum TabBehaviour {
  Tab_Stop,
  Tab_Wrap,
  Tab_Leave }

  Constants defining different support built-in TAB handling behaviours.

Public Member Functions

- virtual void DrawCellHighlight (wxDC &dc, const wxGridCellAttr *attr)
- virtual void DrawRowLabels (wxDC &dc, const wxArrayInt &rows)
- virtual void DrawRowLabel (wxDC &dc, int row)
- virtual void DrawColLabels (wxDC &dc, const wxArrayInt &cols)
- virtual void DrawColLabel (wxDC &dc, int col)
- virtual void DrawCornerLabel (wxDC &dc)
- void DrawTextRectangle (wxDC &dc, const wxString &text, const wxRect &rect, int horizontalAlignment=wxALIGN_LEFT, int verticalAlignment=wxALIGN_TOP, int textOrientation=wxHORIZONTAL) const
- void DrawTextRectangle (wxDC &dc, const wxArrayString &lines, const wxRect &rect, int horizontalAlignment=wxALIGN_LEFT, int verticalAlignment=wxALIGN_TOP, int textOrientation=wxHORIZONTAL) const
- wxColour GetCellHighlightColour () const
- int GetCellHighlightPenWidth () const
- int GetCellHighlightROPenWidth () const
- void SetCellHighlightColour (const wxColour &)
- void SetCellHighlightPenWidth (int width)
- void SetCellHighlightROPenWidth (int width)

Constructors and Initialization

- wxGrid ()
  Default constructor.
- wxGrid (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxWANTS_CHARS, const wxString &name=wxGridNameStr)
  Constructor creating the grid window.
- virtual ~wxGrid ()
  Destructor.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxWANTS_CHARS, const wxString &name=wxGridNameStr)
  Creates the grid window for an object initialized using the default constructor.
- bool CreateGrid (int numRows, int numCols, wxGridSelectionModes selmode=wxGridSelectCells)
  Creates a grid with the specified initial number of rows and columns.
- bool SetTable (wxGridTableBase *table, bool takeOwnership=false, wxGridSelectionModes selmode=wxGridSelectCells)
  Passes a pointer to a custom grid table to be used by the grid.
- bool ProcessTableMessage (wxGridTableMessage &msg)
  Receive and handle a message from the table.

Grid Line Formatting

- void EnableGridLines (bool enable=true)
  Turns the drawing of grid lines on or off.
- virtual wxPen GetColGridLinePen (int col)
  Returns the pen used for vertical grid lines.
- virtual wxPen GetDefaultGridLinePen ()
  Returns the pen used for grid lines.
- wxColour GetGridLineColour () const
  Returns the colour used for grid lines.
- virtual wxPen GetRowGridLinePen (int row)
  Returns the pen used for horizontal grid lines.
- bool GridLinesEnabled () const
  Returns true if drawing of grid lines is turned on, false otherwise.
- void SetGridLineColour (const wxColour &colour)
  Sets the colour used to draw grid lines.

Label Values and Formatting

- void GetColLabelAlignment (int *horiz, int *vert) const
  Sets the arguments to the current column label alignment values.
- int GetColLabelTextOrientation () const
  Returns the orientation of the column labels (either `wxHORIZONTAL` or `wxVERTICAL`).
- wxString GetColLabelValue (int col) const
  Returns the specified column label.
- wxColour GetLabelBackgroundColour () const
  Returns the colour used for the background of row and column labels.
- wxFont GetLabelFont () const
  Returns the font used for row and column labels.
- wxColour GetLabelTextColour () const
  Returns the colour used for row and column label text.
- void GetRowLabelAlignment (int *horiz, int *vert) const
  Returns the alignment used for row labels.
- wxString GetRowLabelValue (int row) const
  Returns the specified row label.
- void HideColLabels ()
  Hides the column labels by calling SetColLabelSize() with a size of 0.
- void HideRowLabels ()
  Hides the row labels by calling SetRowLabelSize() with a size of 0.
- void SetColLabelAlignment (int horiz, int vert)
  Sets the horizontal and vertical alignment of column label text.
- void SetColLabelTextOrientation (int textOrientation)
  Sets the orientation of the column labels (either `wxHORIZONTAL` or `wxVERTICAL`).
- void SetColLabelValue (int col, const wxString &value)
  Set the value for the given column label.
- void SetLabelBackgroundColour (const wxColour &colour)
  Sets the background colour for row and column labels.
- void SetLabelFont (const wxFont &font)
  Sets the font for row and column labels.
- void SetLabelTextColour (const wxColour &colour)
  Sets the colour for row and column label text.
- void SetRowLabelAlignment (int horiz, int vert)
  Sets the horizontal and vertical alignment of row label text.
- void SetRowLabelValue (int row, const wxString &value)
  Sets the value for the given row label.
- void SetUseNativeColLabels (bool native=true)
  Call this in order to make the column labels use a native look by using wxRendererNative::DrawHeaderButton() internally.
- void UseNativeColHeader (bool native=true)
  Enable the use of native header window for column labels.

Cell Formatting

Note that wxGridCellAttr can be used alternatively to most of these methods.

See the "Attributes Management" of wxGridTableBase.

- void GetCellAlignment (int row, int col, int *horiz, int *vert) const
  Sets the arguments to the horizontal and vertical text alignment values for the grid cell at the specified location.
- wxColour GetCellBackgroundColour (int row, int col) const
  Returns the background colour of the cell at the specified location.
- wxFont GetCellFont (int row, int col) const
  Returns the font for text in the grid cell at the specified location.
- wxColour GetCellTextColour (int row, int col) const
  Returns the text colour for the grid cell at the specified location.
- void GetDefaultCellAlignment (int *horiz, int *vert) const
  Returns the default cell alignment.
- wxColour GetDefaultCellBackgroundColour () const
  Returns the current default background colour for grid cells.

- wxFont GetDefaultCellFont () const
  Returns the current default font for grid cell text.
- wxColour GetDefaultCellTextColour () const
  Returns the current default colour for grid cell text.
- void SetCellAlignment (int row, int col, int horiz, int vert)
  Sets the horizontal and vertical alignment for grid cell text at the specified location.
- void SetCellAlignment (int align, int row, int col)
  Sets the horizontal and vertical alignment for grid cell text at the specified location.
- void SetCellBackgroundColour (int row, int col, const wxColour &colour)
  Set the background colour for the given cell or all cells by default.
- void SetCellFont (int row, int col, const wxFont &font)
  Sets the font for text in the grid cell at the specified location.
- void SetCellTextColour (int row, int col, const wxColour &colour)
  Sets the text colour for the given cell.
- void SetCellTextColour (const wxColour &val, int row, int col)
  Sets the text colour for the given cell.
- void SetCellTextColour (const wxColour &colour)
  Sets the text colour for all cells by default.
- void SetDefaultCellAlignment (int horiz, int vert)
  Sets the default horizontal and vertical alignment for grid cell text.
- void SetDefaultCellBackgroundColour (const wxColour &colour)
  Sets the default background colour for grid cells.
- void SetDefaultCellFont (const wxFont &font)
  Sets the default font to be used for grid cell text.
- void SetDefaultCellTextColour (const wxColour &colour)
  Sets the current default colour for grid cell text.

Cell Values, Editors, and Renderers

Note that wxGridCellAttr can be used alternatively to most of these methods.

See the "Attributes Management" of wxGridTableBase.

- bool CanEnableCellControl () const
  Returns true if the in-place edit control for the current grid cell can be used and false otherwise.
- void DisableCellEditControl ()
  Disables in-place editing of grid cells.
- void EnableCellEditControl (bool enable=true)
  Enables or disables in-place editing of grid cell data.
- void EnableEditing (bool edit)
  Makes the grid globally editable or read-only.
- wxGridCellEditor * GetCellEditor (int row, int col) const
  Returns a pointer to the editor for the cell at the specified location.
- wxGridCellRenderer * GetCellRenderer (int row, int col) const
  Returns a pointer to the renderer for the grid cell at the specified location.
- wxString GetCellValue (int row, int col) const
  Returns the string contained in the cell at the specified location.
- wxString GetCellValue (const wxGridCellCoords &coords) const
  Returns the string contained in the cell at the specified location.
- wxGridCellEditor * GetDefaultEditor () const
  Returns a pointer to the current default grid cell editor.
- virtual wxGridCellEditor * GetDefaultEditorForCell (int row, int col) const
  Returns the default editor for the specified cell.
- wxGridCellEditor * GetDefaultEditorForCell (const wxGridCellCoords &c) const
  Returns the default editor for the specified cell.
- virtual wxGridCellEditor * GetDefaultEditorForType (const wxString &typeName) const
  Returns the default editor for the cells containing values of the given type.
- wxGridCellRenderer * GetDefaultRenderer () const
  Returns a pointer to the current default grid cell renderer.
- virtual wxGridCellRenderer * GetDefaultRendererForCell (int row, int col) const

Returns the default renderer for the given cell.

- virtual wxGridCellRenderer * GetDefaultRendererForType (const wxString &typeName) const

  Returns the default renderer for the cell containing values of the given type.
- void HideCellEditControl ()

  Hides the in-place cell edit control.
- bool IsCellEditControlEnabled () const

  Returns true if the in-place edit control is currently enabled.
- bool IsCurrentCellReadOnly () const

  Returns true if the current cell is read-only.
- bool IsEditable () const

  Returns false if the whole grid has been set as read-only or true otherwise.
- bool IsReadOnly (int row, int col) const

  Returns true if the cell at the specified location can't be edited.
- void RegisterDataType (const wxString &typeName, wxGridCellRenderer *renderer, wxGridCellEditor *editor)

  Register a new data type.
- void SaveEditControlValue ()

  Sets the value of the current grid cell to the current in-place edit control value.
- void SetCellEditor (int row, int col, wxGridCellEditor *editor)

  Sets the editor for the grid cell at the specified location.
- void SetCellRenderer (int row, int col, wxGridCellRenderer *renderer)

  Sets the renderer for the grid cell at the specified location.
- void SetCellValue (int row, int col, const wxString &s)

  Sets the string value for the cell at the specified location.
- void SetCellValue (const wxGridCellCoords &coords, const wxString &s)

  Sets the string value for the cell at the specified location.
- void SetCellValue (const wxString &val, int row, int col)
- void SetColFormatBool (int col)

  Sets the specified column to display boolean values.
- void SetColFormatCustom (int col, const wxString &typeName)

  Sets the specified column to display data in a custom format.
- void SetColFormatFloat (int col, int width=-1, int precision=-1)

  Sets the specified column to display floating point values with the given width and precision.
- void SetColFormatNumber (int col)

  Sets the specified column to display integer values.
- void SetDefaultEditor (wxGridCellEditor *editor)

  Sets the default editor for grid cells.
- void SetDefaultRenderer (wxGridCellRenderer *renderer)

  Sets the default renderer for grid cells.
- void SetReadOnly (int row, int col, bool isReadOnly=true)

  Makes the cell at the specified location read-only or editable.
- void ShowCellEditControl ()

  Displays the in-place cell edit control for the current cell.

Column and Row Sizes

See also

Column and Row Sizes

- void AutoSize ()

  Automatically sets the height and width of all rows and columns to fit their contents.
- void AutoSizeColLabelSize (int col)

  Automatically adjusts width of the column to fit its label.
- void AutoSizeColumn (int col, bool setAsMin=true)

  Automatically sizes the column to fit its contents.
- void AutoSizeColumns (bool setAsMin=true)

  Automatically sizes all columns to fit their contents.
- void AutoSizeRow (int row, bool setAsMin=true)

  Automatically sizes the row to fit its contents.

- void AutoSizeRowLabelSize (int col)

  Automatically adjusts height of the row to fit its label.
- void AutoSizeRows (bool setAsMin=true)

  Automatically sizes all rows to fit their contents.
- bool GetCellOverflow (int row, int col) const

  Returns true if the cell value can overflow.
- int GetColLabelSize () const

  Returns the current height of the column labels.
- int GetColMinimalAcceptableWidth () const

  Returns the minimal width to which a column may be resized.
- int GetColSize (int col) const

  Returns the width of the specified column.
- bool IsColShown (int col) const

  Returns true if the specified column is not currently hidden.
- bool GetDefaultCellOverflow () const

  Returns true if the cells can overflow by default.
- int GetDefaultColLabelSize () const

  Returns the default height for column labels.
- int GetDefaultColSize () const

  Returns the current default width for grid columns.
- int GetDefaultRowLabelSize () const

  Returns the default width for the row labels.
- int GetDefaultRowSize () const

  Returns the current default height for grid rows.
- int GetRowMinimalAcceptableHeight () const

  Returns the minimal size to which rows can be resized.
- int GetRowLabelSize () const

  Returns the current width of the row labels.
- int GetRowSize (int row) const

  Returns the height of the specified row.
- bool IsRowShown (int row) const

  Returns true if the specified row is not currently hidden.
- void SetCellOverflow (int row, int col, bool allow)

  Sets the overflow permission of the cell.
- void SetColLabelSize (int height)

  Sets the height of the column labels.
- void SetColMinimalAcceptableWidth (int width)

  Sets the minimal width to which the user can resize columns.
- void SetColMinimalWidth (int col, int width)

  Sets the minimal width for the specified column col.
- void SetColSize (int col, int width)

  Sets the width of the specified column.
- void HideCol (int col)

  Hides the specified column.
- void ShowCol (int col)

  Shows the previously hidden column by resizing it to non-0 size.
- void SetDefaultCellOverflow (bool allow)

  Sets the default overflow permission of the cells.
- void SetDefaultColSize (int width, bool resizeExistingCols=false)

  Sets the default width for columns in the grid.
- void SetDefaultRowSize (int height, bool resizeExistingRows=false)

  Sets the default height for rows in the grid.
- void SetRowLabelSize (int width)

  Sets the width of the row labels.
- void SetRowMinimalAcceptableHeight (int height)

  Sets the minimal row height used by default.
- void SetRowMinimalHeight (int row, int height)

  Sets the minimal height for the specified row.
- void SetRowSize (int row, int height)

Sets the height of the specified row.

- void HideRow (int col)

  Hides the specified row.
- void ShowRow (int col)

  Shows the previously hidden row.
- wxGridSizesInfo GetColSizes () const

  Get size information for all columns at once.
- wxGridSizesInfo GetRowSizes () const

  Get size information for all row at once.
- void SetColSizes (const wxGridSizesInfo &sizeInfo)

  Restore all columns sizes.
- void SetRowSizes (const wxGridSizesInfo &sizeInfo)

  Restore all rows sizes.
- void SetCellSize (int row, int col, int num_rows, int num_cols)

  Set the size of the cell.
- CellSpan GetCellSize (int row, int col, int *num_rows, int *num_cols) const

  Get the size of the cell in number of cells covered by it.
- wxSize GetCellSize (const wxGridCellCoords &coords)

  Get the number of rows and columns allocated for this cell.

User-Resizing and Dragging

Functions controlling various interactive mouse operations.

By default, columns and rows can be resized by dragging the edges of their labels (this can be disabled using DisableDragColSize() and DisableDragRowSize() methods). And if grid line dragging is enabled with EnableDragGridSize() they can also be resized by dragging the right or bottom edge of the grid cells.

Columns can also be moved to interactively change their order but this needs to be explicitly enabled with EnableDragColMove().

- bool CanDragCell () const

  Return true if the dragging of cells is enabled or false otherwise.
- bool CanDragColMove () const

  Returns true if columns can be moved by dragging with the mouse.
- bool CanDragColSize (int col) const

  Returns true if the given column can be resized by dragging with the mouse.
- bool CanDragGridSize () const

  Return true if the dragging of grid lines to resize rows and columns is enabled or false otherwise.
- bool CanDragRowSize (int row) const

  Returns true if the given row can be resized by dragging with the mouse.
- void DisableColResize (int col)

  Disable interactive resizing of the specified column.
- void DisableRowResize (int row)

  Disable interactive resizing of the specified row.
- void DisableDragColMove ()

  Disables column moving by dragging with the mouse.
- void DisableDragColSize ()

  Disables column sizing by dragging with the mouse.
- void DisableDragGridSize ()

  Disable mouse dragging of grid lines to resize rows and columns.
- void DisableDragRowSize ()

  Disables row sizing by dragging with the mouse.
- void EnableDragCell (bool enable=true)

  Enables or disables cell dragging with the mouse.
- void EnableDragColMove (bool enable=true)

  Enables or disables column moving by dragging with the mouse.
- void EnableDragColSize (bool enable=true)

  Enables or disables column sizing by dragging with the mouse.
- void EnableDragGridSize (bool enable=true)

  Enables or disables row and column resizing by dragging gridlines with the mouse.

- void EnableDragRowSize (bool enable=true)

  Enables or disables row sizing by dragging with the mouse.
- int GetColAt (int colPos) const

  Returns the column ID of the specified column position.
- int GetColPos (int colID) const

  Returns the position of the specified column.
- void SetColPos (int colID, int newPos)

  Sets the position of the specified column.
- void SetColumnsOrder (const wxArrayInt &order)

  Sets the positions of all columns at once.
- void ResetColPos ()

  Resets the position of the columns to the default.

Cursor Movement

- int GetGridCursorCol () const

  Returns the current grid cell column position.
- int GetGridCursorRow () const

  Returns the current grid cell row position.
- void GoToCell (int row, int col)

  Make the given cell current and ensure it is visible.
- void GoToCell (const wxGridCellCoords &coords)

  Make the given cell current and ensure it is visible.
- bool MoveCursorDown (bool expandSelection)

  Moves the grid cursor down by one row.
- bool MoveCursorDownBlock (bool expandSelection)

  Moves the grid cursor down in the current column such that it skips to the beginning or end of a block of non-empty cells.
- bool MoveCursorLeft (bool expandSelection)

  Moves the grid cursor left by one column.
- bool MoveCursorLeftBlock (bool expandSelection)

  Moves the grid cursor left in the current row such that it skips to the beginning or end of a block of non-empty cells.
- bool MoveCursorRight (bool expandSelection)

  Moves the grid cursor right by one column.
- bool MoveCursorRightBlock (bool expandSelection)

  Moves the grid cursor right in the current row such that it skips to the beginning or end of a block of non-empty cells.
- bool MoveCursorUp (bool expandSelection)

  Moves the grid cursor up by one row.
- bool MoveCursorUpBlock (bool expandSelection)

  Moves the grid cursor up in the current column such that it skips to the beginning or end of a block of non-empty cells.
- bool MovePageDown ()

  Moves the grid cursor down by some number of rows so that the previous bottom visible row becomes the top visible row.
- bool MovePageUp ()

  Moves the grid cursor up by some number of rows so that the previous top visible row becomes the bottom visible row.
- void SetGridCursor (int row, int col)

  Set the grid cursor to the specified cell.
- void SetGridCursor (const wxGridCellCoords &coords)

  Set the grid cursor to the specified cell.
- void SetTabBehaviour (TabBehaviour behaviour)

  Set the grid's behaviour when the user presses the TAB key.

User Selection

- void ClearSelection ()

  Deselects all cells that are currently selected.

- wxGridCellCoordsArray GetSelectedCells () const
  Returns an array of individually selected cells.
- wxArrayInt GetSelectedCols () const
  Returns an array of selected columns.
- wxArrayInt GetSelectedRows () const
  Returns an array of selected rows.
- wxColour GetSelectionBackground () const
  Returns the colour used for drawing the selection background.
- wxGridCellCoordsArray GetSelectionBlockBottomRight () const
  Returns an array of the bottom right corners of blocks of selected cells.
- wxGridCellCoordsArray GetSelectionBlockTopLeft () const
  Returns an array of the top left corners of blocks of selected cells.
- wxColour GetSelectionForeground () const
  Returns the colour used for drawing the selection foreground.
- wxGridSelectionModes GetSelectionMode () const
  Returns the current selection mode.
- bool IsInSelection (int row, int col) const
  Returns true if the given cell is selected.
- bool IsInSelection (const wxGridCellCoords &coords) const
  Returns true if the given cell is selected.
- bool IsSelection () const
  Returns true if there are currently any selected cells, rows, columns or blocks.
- void SelectAll ()
  Selects all cells in the grid.
- void SelectBlock (int topRow, int leftCol, int bottomRow, int rightCol, bool addToSelected=false)
  Selects a rectangular block of cells.
- void SelectBlock (const wxGridCellCoords &topLeft, const wxGridCellCoords &bottomRight, bool addTo↩ Selected=false)
  Selects a rectangular block of cells.
- void SelectCol (int col, bool addToSelected=false)
  Selects the specified column.
- void SelectRow (int row, bool addToSelected=false)
  Selects the specified row.
- void SetSelectionBackground (const wxColour &c)
  Set the colour to be used for drawing the selection background.
- void SetSelectionForeground (const wxColour &c)
  Set the colour to be used for drawing the selection foreground.
- void SetSelectionMode (wxGridSelectionModes selmode)
  Set the selection behaviour of the grid.

Scrolling

- int GetScrollLineX () const
  Returns the number of pixels per horizontal scroll increment.
- int GetScrollLineY () const
  Returns the number of pixels per vertical scroll increment.
- bool IsVisible (int row, int col, bool wholeCellVisible=true) const
  Returns true if a cell is either entirely or at least partially visible in the grid window.
- bool IsVisible (const wxGridCellCoords &coords, bool wholeCellVisible=true) const
  Returns true if a cell is either entirely or at least partially visible in the grid window.
- void MakeCellVisible (int row, int col)
  Brings the specified cell into the visible grid cell area with minimal scrolling.
- void MakeCellVisible (const wxGridCellCoords &coords)
  Brings the specified cell into the visible grid cell area with minimal scrolling.
- void SetScrollLineX (int x)
  Sets the number of pixels per horizontal scroll increment.
- void SetScrollLineY (int y)
  Sets the number of pixels per vertical scroll increment.

Cell and Device Coordinate Translation

- wxRect BlockToDeviceRect (const wxGridCellCoords &topLeft, const wxGridCellCoords &bottomRight) const

  Convert grid cell coordinates to grid window pixel coordinates.
- wxRect CellToRect (int row, int col) const

  Return the rectangle corresponding to the grid cell's size and position in logical coordinates.
- wxRect CellToRect (const wxGridCellCoords &coords) const

  Return the rectangle corresponding to the grid cell's size and position in logical coordinates.
- int XToCol (int x, bool clipToMinMax=false) const

  Returns the column at the given pixel position.
- int XToEdgeOfCol (int x) const

  Returns the column whose right hand edge is close to the given logical x position.
- wxGridCellCoords XYToCell (int x, int y) const

  Translates logical pixel coordinates to the grid cell coordinates.
- wxGridCellCoords XYToCell (const wxPoint &pos) const

  Translates logical pixel coordinates to the grid cell coordinates.
- int YToEdgeOfRow (int y) const

  Returns the row whose bottom edge is close to the given logical y position.
- int YToRow (int y, bool clipToMinMax=false) const

  Returns the grid row that corresponds to the logical y coordinate.

Miscellaneous Functions

- bool AppendCols (int numCols=1, bool updateLabels=true)

  Appends one or more new columns to the right of the grid.
- bool AppendRows (int numRows=1, bool updateLabels=true)

  Appends one or more new rows to the bottom of the grid.
- bool AreHorzGridLinesClipped () const

  Return true if the horizontal grid lines stop at the last column boundary or false if they continue to the end of the window.
- bool AreVertGridLinesClipped () const

  Return true if the vertical grid lines stop at the last row boundary or false if they continue to the end of the window.
- void BeginBatch ()

  Increments the grid's batch count.
- void ClearGrid ()

  Clears all data in the underlying grid table and repaints the grid.
- void ClipHorzGridLines (bool clip)

  Change whether the horizontal grid lines are clipped by the end of the last column.
- void ClipVertGridLines (bool clip)

  Change whether the vertical grid lines are clipped by the end of the last row.
- bool DeleteCols (int pos=0, int numCols=1, bool updateLabels=true)

  Deletes one or more columns from a grid starting at the specified position.
- bool DeleteRows (int pos=0, int numRows=1, bool updateLabels=true)

  Deletes one or more rows from a grid starting at the specified position.
- void EndBatch ()

  Decrements the grid's batch count.
- virtual void Fit ()

  Overridden wxWindow method.
- void ForceRefresh ()

  Causes immediate repainting of the grid.
- int GetBatchCount ()

  Returns the number of times that BeginBatch() has been called without (yet) matching calls to EndBatch().
- int GetNumberCols () const

  Returns the total number of grid columns.
- int GetNumberRows () const

  Returns the total number of grid rows.
- wxGridCellAttr * GetOrCreateCellAttr (int row, int col) const

  Returns the attribute for the given cell creating one if necessary.

- wxGridTableBase * GetTable () const
  Returns a base pointer to the current table object.
- bool InsertCols (int pos=0, int numCols=1, bool updateLabels=true)
  Inserts one or more new columns into a grid with the first new column at the specified position.
- bool InsertRows (int pos=0, int numRows=1, bool updateLabels=true)
  Inserts one or more new rows into a grid with the first new row at the specified position.
- void RefreshAttr (int row, int col)
  Invalidates the cached attribute for the given cell.
- void Render (wxDC &dc, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, const wxGridCellCoords &topLeft=wxGridCellCoords(-1,-1), const wxGridCellCoords &bottomRight=wxGridCellCoords(-1,-1), int style=wxGRID_DRAW_DEFAULT)
  Draws part or all of a wxGrid on a wxDC for printing or display.
- void SetColAttr (int col, wxGridCellAttr *attr)
  Sets the cell attributes for all cells in the specified column.
- void SetMargins (int extraWidth, int extraHeight)
  Sets the extra margins used around the grid area.
- void SetRowAttr (int row, wxGridCellAttr *attr)
  Sets the cell attributes for all cells in the specified row.
- wxArrayInt CalcRowLabelsExposed (const wxRegion &reg)
  Appends one or more new columns to the right of the grid.
- wxArrayInt CalcColLabelsExposed (const wxRegion &reg)
  Appends one or more new columns to the right of the grid.
- wxGridCellCoordsArray CalcCellsExposed (const wxRegion &reg)
  Appends one or more new columns to the right of the grid.

Sorting support.

wxGrid doesn't provide any support for sorting the data but it does generate events allowing the user code to sort it and supports displaying the sort indicator in the column used for sorting.

To use wxGrid sorting support you need to handle wxEVT_GRID_COL_SORT event (and not veto it) and resort the data displayed in the grid. The grid will automatically update the sorting indicator on the column which was clicked.

You can also call the functions in this section directly to update the sorting indicator. Once again, they don't do anything with the grid data, it remains your responsibility to actually sort it appropriately.

- int GetSortingColumn () const
  Return the column in which the sorting indicator is currently displayed.
- bool IsSortingBy (int col) const
  Return true if this column is currently used for sorting.
- bool IsSortOrderAscending () const
  Return true if the current sorting order is ascending or false if it is descending.
- void SetSortingColumn (int col, bool ascending=true)
  Set the column to display the sorting indicator in and its direction.
- void UnsetSortingColumn ()
  Remove any currently shown sorting indicator.

Accessors for component windows.

Return the various child windows of wxGrid.

wxGrid is an empty parent window for 4 children representing the column labels window (top), the row labels window (left), the corner window (top left) and the main grid window. It may be necessary to use these individual windows and not the wxGrid window itself if you need to handle events for them (this can be done using wxEvtHandler::Connect() or wxWindow::PushEventHandler()) or do something else requiring the use of the correct window pointer. Notice that you should not, however, change these windows (e.g. reposition them or draw over them) because they are managed by wxGrid itself.

- wxWindow * GetGridWindow () const
  Return the main grid window containing the grid cells.
- wxWindow * GetGridRowLabelWindow () const

Return the row labels window.

- wxWindow * GetGridColLabelWindow () const

  Return the column labels window.

- wxWindow * GetGridCornerLabelWindow () const

  Return the window in the top left grid corner.

- wxHeaderCtrl * GetGridColHeader () const

  Return the header control used for column labels display.

Protected Member Functions

- bool CanHaveAttributes () const

  Returns true if this grid has support for cell attributes.

- int GetColMinimalWidth (int col) const

  Get the minimal width of the given column/row.

- int GetColRight (int col) const

  Returns the coordinate of the right border specified column.

- int GetColLeft (int col) const

  Returns the coordinate of the left border specified column.

- int GetRowMinimalHeight (int col) const

  Returns the minimal size for the given column.

21.298.2 Member Enumeration Documentation

enum wxGrid::CellSpan

Return values for GetCellSize().

Since

2.9.1

Enumerator

CellSpan_Inside This cell is inside a span covered by another cell.

CellSpan_None This is a normal, non-spanning cell.

CellSpan_Main This cell spans several physical wxGrid cells.

enum wxGrid::TabBehaviour

Constants defining different support built-in TAB handling behaviours.

The elements of this enum determine what happens when TAB is pressed when the cursor is in the rightmost column (or Shift-TAB is pressed when the cursor is in the leftmost one).

See also

SetTabBehaviour(), `wxEVT_GRID_TABBING`

Since

2.9.5

Enumerator

Tab_Stop Do nothing, this is default.

Tab_Wrap Move to the beginning of the next (or the end of the previous) row.

Tab_Leave Move to the next (or the previous) control after the grid.

enum wxGrid::wxGridSelectionModes

Different selection modes supported by the grid.

Enumerator

wxGridSelectCells The default selection mode allowing selection of the individual cells as well as of the entire rows and columns.

wxGridSelectRows The selection mode allowing the selection of the entire rows only. The user won't be able to select any cells or columns in this mode.

wxGridSelectColumns The selection mode allowing the selection of the entire columns only. The user won't be able to select any cells or rows in this mode.

wxGridSelectRowsOrColumns The selection mode allowing the user to select either the entire columns or the entire rows but not individual cells nor blocks. Notice that while this constant is defined as

```
wxGridSelectColumns | wxGridSelectRows
```

this doesn't mean that all the other combinations are valid – at least currently they are not.

Since

2.9.1

21.298.3 Constructor & Destructor Documentation

wxGrid::wxGrid ( )

Default constructor.

You must call Create() to really create the grid window and also call CreateGrid() or SetTable() to initialize the grid contents.

wxGrid::wxGrid (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxWANTS_CHARS, const wxString & *name* = `wxGridNameStr`)

Constructor creating the grid window.

You must call either CreateGrid() or SetTable() to initialize the grid contents before using it.

virtual wxGrid::~wxGrid ( ) `[virtual]`

Destructor.

This will also destroy the associated grid table unless you passed a table object to the grid and specified that the grid should not take ownership of the table (see SetTable()).

21.298.4 Member Function Documentation

bool wxGrid::AppendCols (int *numCols* = `1`, bool *updateLabels* = `true`)

Appends one or more new columns to the right of the grid.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::AppendCols(). See InsertCols() for further information.

Returns

true on success or false if appending columns failed.

bool wxGrid::AppendRows (int *numRows* = `1`, bool *updateLabels* = `true`)

Appends one or more new rows to the bottom of the grid.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::AppendRows(). See InsertRows() for further information.

Returns

true on success or false if appending rows failed.

bool wxGrid::AreHorzGridLinesClipped ( ) const

Return true if the horizontal grid lines stop at the last column boundary or false if they continue to the end of the window.

The default is to clip grid lines.

See also

ClipHorzGridLines(), AreVertGridLinesClipped()

bool wxGrid::AreVertGridLinesClipped ( ) const

Return true if the vertical grid lines stop at the last row boundary or false if they continue to the end of the window.

The default is to clip grid lines.

See also

ClipVertGridLines(), AreHorzGridLinesClipped()

void wxGrid::AutoSize ( )

Automatically sets the height and width of all rows and columns to fit their contents.

void wxGrid::AutoSizeColLabelSize (int *col*)

Automatically adjusts width of the column to fit its label.

void wxGrid::AutoSizeColumn (int *col*, bool *setAsMin* = `true`)

Automatically sizes the column to fit its contents.

If *setAsMin* is true the calculated width will also be set as the minimal width for the column.

void wxGrid::AutoSizeColumns (bool *setAsMin* = `true`)

Automatically sizes all columns to fit their contents.

If *setAsMin* is true the calculated widths will also be set as the minimal widths for the columns.

void wxGrid::AutoSizeRow (int *row*, bool *setAsMin* = `true`)

Automatically sizes the row to fit its contents.

If *setAsMin* is true the calculated height will also be set as the minimal height for the row.

void wxGrid::AutoSizeRowLabelSize (int *col*)

Automatically adjusts height of the row to fit its label.

void wxGrid::AutoSizeRows (bool *setAsMin* = `true`)

Automatically sizes all rows to fit their contents.

If *setAsMin* is true the calculated heights will also be set as the minimal heights for the rows.

void wxGrid::BeginBatch ( )

Increments the grid's batch count.

When the count is greater than zero repainting of the grid is suppressed. Each call to BeginBatch must be matched by a later call to EndBatch(). Code that does a lot of grid modification can be enclosed between BeginBatch() and EndBatch() calls to avoid screen flicker. The final EndBatch() call will cause the grid to be repainted.

Notice that you should use wxGridUpdateLocker which ensures that there is always a matching EndBatch() call for this BeginBatch() if possible instead of calling this method directly.

wxRect wxGrid::BlockToDeviceRect (const wxGridCellCoords & *topLeft*, const wxGridCellCoords & *bottomRight*) const

Convert grid cell coordinates to grid window pixel coordinates.

This function returns the rectangle that encloses the block of cells limited by *topLeft* and *bottomRight* cell in device coords and clipped to the client size of the grid window.

See also

CellToRect()

wxGridCellCoordsArray wxGrid::CalcCellsExposed (const wxRegion & *reg*)

Appends one or more new columns to the right of the grid.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::AppendCols(). See InsertCols() for further information.

Returns

true on success or false if appending columns failed.

wxArrayInt wxGrid::CalcColLabelsExposed (const wxRegion & *reg*)

Appends one or more new columns to the right of the grid.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::AppendCols(). See InsertCols() for further information.

Returns

true on success or false if appending columns failed.

wxArrayInt wxGrid::CalcRowLabelsExposed (const wxRegion & *reg*)

Appends one or more new columns to the right of the grid.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::AppendCols(). See InsertCols() for further information.

Returns

true on success or false if appending columns failed.

bool wxGrid::CanDragCell ( ) const

Return true if the dragging of cells is enabled or false otherwise.

bool wxGrid::CanDragColMove ( ) const

Returns true if columns can be moved by dragging with the mouse.

Columns can be moved by dragging on their labels.

bool wxGrid::CanDragColSize (int *col*) const

Returns true if the given column can be resized by dragging with the mouse.

This function returns true if resizing the columns interactively is globally enabled, i.e. if DisableDragColSize() hadn't been called, and if this column wasn't explicitly marked as non-resizable with DisableColResize().

bool wxGrid::CanDragGridSize ( ) const

Return true if the dragging of grid lines to resize rows and columns is enabled or false otherwise.

bool wxGrid::CanDragRowSize (int *row*) const

Returns true if the given row can be resized by dragging with the mouse.

This is the same as CanDragColSize() but for rows.

bool wxGrid::CanEnableCellControl ( ) const

Returns true if the in-place edit control for the current grid cell can be used and false otherwise.

This function always returns false for the read-only cells.

bool wxGrid::CanHaveAttributes ( ) const `[protected]`

Returns true if this grid has support for cell attributes.

The grid supports attributes if it has the associated table which, in turn, has attributes support, i.e. wxGridTableBase::CanHaveAttributes() returns true.

wxRect wxGrid::CellToRect (int *row,* int *col*) const

Return the rectangle corresponding to the grid cell's size and position in logical coordinates.

See also

BlockToDeviceRect()

wxRect wxGrid::CellToRect (const wxGridCellCoords & *coords*) const

Return the rectangle corresponding to the grid cell's size and position in logical coordinates.

See also

BlockToDeviceRect()

void wxGrid::ClearGrid ( )

Clears all data in the underlying grid table and repaints the grid.

The table is not deleted by this function. If you are using a derived table class then you need to override wxGrid↩ TableBase::Clear() for this function to have any effect.

void wxGrid::ClearSelection ( )

Deselects all cells that are currently selected.

void wxGrid::ClipHorzGridLines (bool *clip*)

Change whether the horizontal grid lines are clipped by the end of the last column.

By default the grid lines are not drawn beyond the end of the last column but after calling this function with *clip* set to false they will be drawn across the entire grid window.

See also

AreHorzGridLinesClipped(), ClipVertGridLines()

void wxGrid::ClipVertGridLines (bool *clip*)

Change whether the vertical grid lines are clipped by the end of the last row.

By default the grid lines are not drawn beyond the end of the last row but after calling this function with *clip* set to false they will be drawn across the entire grid window.

See also

AreVertGridLinesClipped(), ClipHorzGridLines()

bool wxGrid::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxWANTS_CHARS, const wxString & *name* = `wxGridNameStr`)

Creates the grid window for an object initialized using the default constructor.

You must call either CreateGrid() or SetTable() to initialize the grid contents before using it.

bool wxGrid::CreateGrid (int *numRows,* int *numCols,* wxGridSelectionModes *selmode* = wxGridSelectCells)

Creates a grid with the specified initial number of rows and columns.

Call this directly after the grid constructor. When you use this function wxGrid will create and manage a simple table of string values for you. All of the grid data will be stored in memory.

For applications with more complex data types or relationships, or for dealing with very large datasets, you should derive your own grid table class and pass a table object to the grid with SetTable().

bool wxGrid::DeleteCols (int *pos* = 0, int *numCols* = 1, bool *updateLabels* = true)

Deletes one or more columns from a grid starting at the specified position.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::DeleteCols(). See InsertCols() for further information.

Returns

true on success or false if deleting columns failed.

bool wxGrid::DeleteRows (int *pos* = 0, int *numRows* = 1, bool *updateLabels* = true)

Deletes one or more rows from a grid starting at the specified position.

The *updateLabels* argument is not used at present. If you are using a derived grid table class you will need to override wxGridTableBase::DeleteRows(). See InsertRows() for further information.

Returns

true on success or false if appending rows failed.

void wxGrid::DisableCellEditControl ( )

Disables in-place editing of grid cells.

Equivalent to calling EnableCellEditControl(false).

void wxGrid::DisableColResize (int *col*)

Disable interactive resizing of the specified column.

This method allows to disable resizing of an individual column in a grid where the columns are otherwise resizable (which is the case by default).

Notice that currently there is no way to make some columns resizable in a grid where columns can't be resized by default as there doesn't seem to be any need for this in practice. There is also no way to make the column marked as fixed using this method resizable again because it is supposed that fixed columns are used for static parts of the grid and so should remain fixed during the entire grid lifetime.

Also notice that disabling interactive column resizing will not prevent the program from changing the column size.

See also

EnableDragColSize()

void wxGrid::DisableDragColMove ( )

Disables column moving by dragging with the mouse.

Equivalent to passing false to EnableDragColMove().

void wxGrid::DisableDragColSize ( )

Disables column sizing by dragging with the mouse.

Equivalent to passing false to EnableDragColSize().

void wxGrid::DisableDragGridSize ( )

Disable mouse dragging of grid lines to resize rows and columns.

Equivalent to passing false to EnableDragGridSize()

void wxGrid::DisableDragRowSize ( )

Disables row sizing by dragging with the mouse.

Equivalent to passing false to EnableDragRowSize().

void wxGrid::DisableRowResize (int *row*)

Disable interactive resizing of the specified row.

This is the same as DisableColResize() but for rows.

See also

EnableDragRowSize()

virtual void wxGrid::DrawCellHighlight (wxDC & *dc,* const wxGridCellAttr * *attr*) [virtual]

virtual void wxGrid::DrawColLabel (wxDC & *dc,* int *col*) [virtual]

virtual void wxGrid::DrawColLabels (wxDC & *dc,* const wxArrayInt & *cols*) [virtual]

virtual void wxGrid::DrawCornerLabel (wxDC & *dc*) [virtual]

virtual void wxGrid::DrawRowLabel (wxDC & *dc,* int *row*) [virtual]

virtual void wxGrid::DrawRowLabels (wxDC & *dc,* const wxArrayInt & *rows*) [virtual]

void wxGrid::DrawTextRectangle (wxDC & *dc,* const wxString & *text,* const wxRect & *rect,* int *horizontalAlignment* = wxALIGN_LEFT, int *verticalAlignment* = wxALIGN_TOP, int *textOrientation* = wxHORIZONTAL) const

void wxGrid::DrawTextRectangle (wxDC & *dc,* const wxArrayString & *lines,* const wxRect & *rect,* int *horizontalAlignment* = wxALIGN_LEFT, int *verticalAlignment* = wxALIGN_TOP, int *textOrientation* = wxHORIZONTAL) const

void wxGrid::EnableCellEditControl (bool *enable* = true)

Enables or disables in-place editing of grid cell data.

The grid will issue either a wxEVT_GRID_EDITOR_SHOWN or wxEVT_GRID_EDITOR_HIDDEN event.

void wxGrid::EnableDragCell (bool *enable* = `true`)

Enables or disables cell dragging with the mouse.

void wxGrid::EnableDragColMove (bool *enable* = `true`)

Enables or disables column moving by dragging with the mouse.

void wxGrid::EnableDragColSize (bool *enable* = `true`)

Enables or disables column sizing by dragging with the mouse.

See also

DisableColResize()

void wxGrid::EnableDragGridSize (bool *enable* = `true`)

Enables or disables row and column resizing by dragging gridlines with the mouse.

void wxGrid::EnableDragRowSize (bool *enable* = `true`)

Enables or disables row sizing by dragging with the mouse.

See also

DisableRowResize()

void wxGrid::EnableEditing (bool *edit*)

Makes the grid globally editable or read-only.

If the edit argument is false this function sets the whole grid as read-only. If the argument is true the grid is set to the default state where cells may be editable. In the default state you can set single grid cells and whole rows and columns to be editable or read-only via wxGridCellAttr::SetReadOnly(). For single cells you can also use the shortcut function SetReadOnly().

For more information about controlling grid cell attributes see the wxGridCellAttr class and the wxGrid Overview.

void wxGrid::EnableGridLines (bool *enable* = `true`)

Turns the drawing of grid lines on or off.

void wxGrid::EndBatch ( )

Decrements the grid's batch count.

When the count is greater than zero repainting of the grid is suppressed. Each previous call to BeginBatch() must be matched by a later call to EndBatch(). Code that does a lot of grid modification can be enclosed between BeginBatch() and EndBatch() calls to avoid screen flicker. The final EndBatch() will cause the grid to be repainted.

See also

wxGridUpdateLocker

virtual void wxGrid::Fit ( ) [virtual]

Overridden wxWindow method.

void wxGrid::ForceRefresh ( )

Causes immediate repainting of the grid.

Use this instead of the usual wxWindow::Refresh().

int wxGrid::GetBatchCount ( )

Returns the number of times that BeginBatch() has been called without (yet) matching calls to EndBatch().

While the grid's batch count is greater than zero the display will not be updated.

void wxGrid::GetCellAlignment (int *row,* int *col,* int * *horiz,* int * *vert*) const

Sets the arguments to the horizontal and vertical text alignment values for the grid cell at the specified location.

Horizontal alignment will be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment will be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

wxColour wxGrid::GetCellBackgroundColour (int *row,* int *col*) const

Returns the background colour of the cell at the specified location.

wxGridCellEditor* wxGrid::GetCellEditor (int *row,* int *col*) const

Returns a pointer to the editor for the cell at the specified location.

See wxGridCellEditor and the wxGrid Overview for more information about cell editors and renderers.

The caller must call DecRef() on the returned pointer.

wxFont wxGrid::GetCellFont (int *row,* int *col*) const

Returns the font for text in the grid cell at the specified location.

wxColour wxGrid::GetCellHighlightColour ( ) const

int wxGrid::GetCellHighlightPenWidth ( ) const

int wxGrid::GetCellHighlightROPenWidth ( ) const

bool wxGrid::GetCellOverflow (int *row,* int *col*) const

Returns true if the cell value can overflow.

A cell can overflow if the next cell in the row is empty.

wxGridCellRenderer* wxGrid::GetCellRenderer (int *row,* int *col*) const

Returns a pointer to the renderer for the grid cell at the specified location.

See wxGridCellRenderer and the wxGrid Overview for more information about cell editors and renderers.

The caller must call DecRef() on the returned pointer.

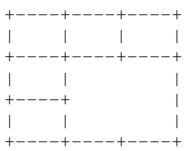
CellSpan wxGrid::GetCellSize (int *row,* int *col,* int * *num_rows,* int * *num_cols*) const

Get the size of the cell in number of cells covered by it.

For normal cells, the function fills both *num_rows* and *num_cols* with 1 and returns CellSpan_None. For cells which span multiple cells, i.e. for which SetCellSize() had been called, the returned values are the same ones as were passed to SetCellSize() call and the function return value is CellSpan_Main.

More unexpectedly, perhaps, the returned values may be *negative* for the cells which are inside a span covered by a cell occupying multiple rows or columns. They correspond to the offset of the main cell of the span from the cell passed to this functions and the function returns CellSpan_Inside value to indicate this.

As an example, consider a 3*3 grid with the cell (1, 1) (the one in the middle) having a span of 2 rows and 2 columns, i.e. the grid looks like



Then the function returns 2 and 2 in *num_rows* and *num_cols* for the cell (1, 1) itself and -1 and -1 for the cell (2, 2) as well as -1 and 0 for the cell (2, 1).

Parameters

row	The row of the cell.
col	The column of the cell.
num_rows	Pointer to variable receiving the number of rows, must not be NULL.
num_cols	Pointer to variable receiving the number of columns, must not be NULL.

Returns

The kind of this cell span (the return value is new in wxWidgets 2.9.1, this function was void in previous wxWidgets versions).

wxSize wxGrid::GetCellSize (const wxGridCellCoords & *coords*)

Get the number of rows and columns allocated for this cell.

This overload doesn't return a CellSpan value but the values returned may still be negative, see GetCellSize(int, int, int *, int *) for details.

wxColour wxGrid::GetCellTextColour (int *row,* int *col*) const

Returns the text colour for the grid cell at the specified location.

wxString wxGrid::GetCellValue (int *row,* int *col*) const

Returns the string contained in the cell at the specified location.

For simple applications where a grid object automatically uses a default grid table of string values you use this function together with SetCellValue() to access cell values. For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

See wxGridTableBase::CanGetValueAs() and the wxGrid Overview for more information.

wxString wxGrid::GetCellValue (const wxGridCellCoords & *coords*) const

Returns the string contained in the cell at the specified location.

For simple applications where a grid object automatically uses a default grid table of string values you use this function together with SetCellValue() to access cell values. For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

See wxGridTableBase::CanGetValueAs() and the wxGrid Overview for more information.

int wxGrid::GetColAt (int *colPos*) const

Returns the column ID of the specified column position.

virtual wxPen wxGrid::GetColGridLinePen (int *col*) [virtual]

Returns the pen used for vertical grid lines.

This virtual function may be overridden in derived classes in order to change the appearance of individual grid lines for the given column *col*.

See GetRowGridLinePen() for an example.

void wxGrid::GetColLabelAlignment (int * *horiz*, int * *vert*) const

Sets the arguments to the current column label alignment values.

Horizontal alignment will be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment will be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

int wxGrid::GetColLabelSize ( ) const

Returns the current height of the column labels.

int wxGrid::GetColLabelTextOrientation ( ) const

Returns the orientation of the column labels (either `wxHORIZONTAL` or `wxVERTICAL`).

wxString wxGrid::GetColLabelValue (int *col*) const

Returns the specified column label.

The default grid table class provides column labels of the form A,B...Z,AA,AB...ZZ,AAA... If you are using a custom grid table you can override wxGridTableBase::GetColLabelValue() to provide your own labels.

int wxGrid::GetColLeft (int *col*) const [protected]

Returns the coordinate of the left border specified column.

int wxGrid::GetColMinimalAcceptableWidth ( ) const

Returns the minimal width to which a column may be resized.

Use SetColMinimalAcceptableWidth() to change this value globally or SetColMinimalWidth() to do it for individual columns.

See also

GetRowMinimalAcceptableHeight()

int wxGrid::GetColMinimalWidth (int *col*) const `[protected]`

Get the minimal width of the given column/row.

The value returned by this function may be different than that returned by GetColMinimalAcceptableWidth() if SetColMinimalWidth() had been called for this column.

int wxGrid::GetColPos (int *colID*) const

Returns the position of the specified column.

int wxGrid::GetColRight (int *col*) const `[protected]`

Returns the coordinate of the right border specified column.

int wxGrid::GetColSize (int *col*) const

Returns the width of the specified column.

wxGridSizesInfo wxGrid::GetColSizes ( ) const

Get size information for all columns at once.

This method is useful when the information about all column widths needs to be saved. The widths can be later restored using SetColSizes().

See also

wxGridSizesInfo, GetRowSizes()

void wxGrid::GetDefaultCellAlignment (int * *horiz,* int * *vert*) const

Returns the default cell alignment.

Horizontal alignment will be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment will be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

See also

SetDefaultCellAlignment()

wxColour wxGrid::GetDefaultCellBackgroundColour ( ) const

Returns the current default background colour for grid cells.

wxFont wxGrid::GetDefaultCellFont ( ) const

Returns the current default font for grid cell text.

bool wxGrid::GetDefaultCellOverflow ( ) const

Returns true if the cells can overflow by default.

wxColour wxGrid::GetDefaultCellTextColour ( ) const

Returns the current default colour for grid cell text.

int wxGrid::GetDefaultColLabelSize ( ) const

Returns the default height for column labels.

int wxGrid::GetDefaultColSize ( ) const

Returns the current default width for grid columns.

wxGridCellEditor* wxGrid::GetDefaultEditor ( ) const

Returns a pointer to the current default grid cell editor.

See wxGridCellEditor and the wxGrid Overview for more information about cell editors and renderers.

virtual wxGridCellEditor* wxGrid::GetDefaultEditorForCell (int *row*, int *col*) const [virtual]

Returns the default editor for the specified cell.

The base class version returns the editor appropriate for the current cell type but this method may be overridden in the derived classes to use custom editors for some cells by default.

Notice that the same may be achieved in a usually simpler way by associating a custom editor with the given cell or cells.

The caller must call DecRef() on the returned pointer.

wxGridCellEditor* wxGrid::GetDefaultEditorForCell (const wxGridCellCoords & *c*) const

Returns the default editor for the specified cell.

The base class version returns the editor appropriate for the current cell type but this method may be overridden in the derived classes to use custom editors for some cells by default.

Notice that the same may be achieved in a usually simpler way by associating a custom editor with the given cell or cells.

The caller must call DecRef() on the returned pointer.

virtual wxGridCellEditor* wxGrid::GetDefaultEditorForType (const wxString & *typeName*) const [virtual]

Returns the default editor for the cells containing values of the given type.

The base class version returns the editor which was associated with the specified *typeName* when it was registered RegisterDataType() but this function may be overridden to return something different. This allows to override an editor used for one of the standard types.

The caller must call DecRef() on the returned pointer.

virtual wxPen wxGrid::GetDefaultGridLinePen ( ) [virtual]

Returns the pen used for grid lines.

This virtual function may be overridden in derived classes in order to change the appearance of grid lines. Note that currently the pen width must be 1.

See also

GetColGridLinePen(), GetRowGridLinePen()

wxGridCellRenderer* wxGrid::GetDefaultRenderer ( ) const

Returns a pointer to the current default grid cell renderer.

See wxGridCellRenderer and the wxGrid Overview for more information about cell editors and renderers.

The caller must call DecRef() on the returned pointer.

virtual wxGridCellRenderer* wxGrid::GetDefaultRendererForCell (int *row*, int *col*) const [virtual]

Returns the default renderer for the given cell.

The base class version returns the renderer appropriate for the current cell type but this method may be overridden in the derived classes to use custom renderers for some cells by default.

The caller must call DecRef() on the returned pointer.

virtual wxGridCellRenderer* wxGrid::GetDefaultRendererForType (const wxString & *typeName*) const [virtual]

Returns the default renderer for the cell containing values of the given type.

See also

GetDefaultEditorForType()

int wxGrid::GetDefaultRowLabelSize ( ) const

Returns the default width for the row labels.

int wxGrid::GetDefaultRowSize ( ) const

Returns the current default height for grid rows.

wxHeaderCtrl* wxGrid::GetGridColHeader ( ) const

Return the header control used for column labels display.

This function can only be called if UseNativeColHeader() had been called.

wxWindow* wxGrid::GetGridColLabelWindow ( ) const

Return the column labels window.

This window is not shown if the columns labels were hidden using HideColLabels().

Depending on whether UseNativeColHeader() was called or not this can be either a wxHeaderCtrl or a plain wxWindow. This function returns a valid window pointer in either case but in the former case you can also use GetGridColHeader() to access it if you need wxHeaderCtrl-specific functionality.

wxWindow* wxGrid::GetGridCornerLabelWindow ( ) const

Return the window in the top left grid corner.

This window is shown only of both columns and row labels are shown and normally doesn't contain anything. Clicking on it is handled by wxGrid however and can be used to select the entire grid.

int wxGrid::GetGridCursorCol ( ) const

Returns the current grid cell column position.

int wxGrid::GetGridCursorRow ( ) const

Returns the current grid cell row position.

wxColour wxGrid::GetGridLineColour ( ) const

Returns the colour used for grid lines.

See also

GetDefaultGridLinePen()

wxWindow* wxGrid::GetGridRowLabelWindow ( ) const

Return the row labels window.

This window is not shown if the row labels were hidden using HideRowLabels().

wxWindow* wxGrid::GetGridWindow ( ) const

Return the main grid window containing the grid cells.

This window is always shown.

wxColour wxGrid::GetLabelBackgroundColour ( ) const

Returns the colour used for the background of row and column labels.

wxFont wxGrid::GetLabelFont ( ) const

Returns the font used for row and column labels.

wxColour wxGrid::GetLabelTextColour ( ) const

Returns the colour used for row and column label text.

int wxGrid::GetNumberCols ( ) const

Returns the total number of grid columns.

This is the same as the number of columns in the underlying grid table.

int wxGrid::GetNumberRows ( ) const

Returns the total number of grid rows.

This is the same as the number of rows in the underlying grid table.

wxGridCellAttr* wxGrid::GetOrCreateCellAttr (int *row*, int *col*) const

Returns the attribute for the given cell creating one if necessary.

If the cell already has an attribute, it is returned. Otherwise a new attribute is created, associated with the cell and returned. In any case the caller must call DecRef() on the returned pointer.

This function may only be called if CanHaveAttributes() returns true.

virtual wxPen wxGrid::GetRowGridLinePen (int *row*) [virtual]

Returns the pen used for horizontal grid lines.

This virtual function may be overridden in derived classes in order to change the appearance of individual grid line for the given *row*.

Example:

```
// in a grid displaying music notation, use a solid black pen between
// octaves (C0=row 127, C1=row 115 etc.)
wxPen MidiGrid::GetRowGridLinePen(int row)
{
    if ( row % 12 == 7 )
        return wxPen(*wxBLACK, 1, wxSOLID);
    else
        return GetDefaultGridLinePen();
}
```

void wxGrid::GetRowLabelAlignment (int * *horiz*, int * *vert*) const

Returns the alignment used for row labels.

Horizontal alignment will be one of wxALIGN_LEFT, wxALIGN_CENTRE or wxALIGN_RIGHT.

Vertical alignment will be one of wxALIGN_TOP, wxALIGN_CENTRE or wxALIGN_BOTTOM.

int wxGrid::GetRowLabelSize ( ) const

Returns the current width of the row labels.

wxString wxGrid::GetRowLabelValue (int *row*) const

Returns the specified row label.

The default grid table class provides numeric row labels. If you are using a custom grid table you can override wxGridTableBase::GetRowLabelValue() to provide your own labels.

int wxGrid::GetRowMinimalAcceptableHeight ( ) const

Returns the minimal size to which rows can be resized.

Use SetRowMinimalAcceptableHeight() to change this value globally or SetRowMinimalHeight() to do it for individual cells.

See also

GetColMinimalAcceptableWidth()

int wxGrid::GetRowMinimalHeight (int *col*) const `[protected]`

Returns the minimal size for the given column.

The value returned by this function may be different than that returned by GetRowMinimalAcceptableHeight() if SetRowMinimalHeight() had been called for this row.

int wxGrid::GetRowSize (int *row*) const

Returns the height of the specified row.

wxGridSizesInfo wxGrid::GetRowSizes ( ) const

Get size information for all row at once.

See also

wxGridSizesInfo, GetColSizes()

int wxGrid::GetScrollLineX ( ) const

Returns the number of pixels per horizontal scroll increment.

The default is 15.

See also

GetScrollLineY(), SetScrollLineX(), SetScrollLineY()

int wxGrid::GetScrollLineY ( ) const

Returns the number of pixels per vertical scroll increment.

The default is 15.

See also

GetScrollLineX(), SetScrollLineX(), SetScrollLineY()

wxGridCellCoordsArray wxGrid::GetSelectedCells ( ) const

Returns an array of individually selected cells.

Notice that this array does *not* contain all the selected cells in general as it doesn't include the cells selected as part of column, row or block selection. You must use this method, GetSelectedCols(), GetSelectedRows() and GetSelectionBlockTopLeft() and GetSelectionBlockBottomRight() methods to obtain the entire selection in general.

Please notice this behaviour is by design and is needed in order to support grids of arbitrary size (when an entire column is selected in a grid with a million of columns, we don't want to create an array with a million of entries in this function, instead it returns an empty array and GetSelectedCols() returns an array containing one element).

wxArrayInt wxGrid::GetSelectedCols ( ) const

Returns an array of selected columns.

Please notice that this method alone is not sufficient to find all the selected columns as it contains only the columns which were individually selected but not those being part of the block selection or being selected in virtue of all of their cells being selected individually, please see GetSelectedCells() for more details.

wxArrayInt wxGrid::GetSelectedRows ( ) const

Returns an array of selected rows.

Please notice that this method alone is not sufficient to find all the selected rows as it contains only the rows which were individually selected but not those being part of the block selection or being selected in virtue of all of their cells being selected individually, please see GetSelectedCells() for more details.

wxColour wxGrid::GetSelectionBackground ( ) const

Returns the colour used for drawing the selection background.

wxGridCellCoordsArray wxGrid::GetSelectionBlockBottomRight ( ) const

Returns an array of the bottom right corners of blocks of selected cells.

Please see GetSelectedCells() for more information about the selection representation in wxGrid.

See also

GetSelectionBlockTopLeft()

wxGridCellCoordsArray wxGrid::GetSelectionBlockTopLeft ( ) const

Returns an array of the top left corners of blocks of selected cells.

Please see GetSelectedCells() for more information about the selection representation in wxGrid.

See also

GetSelectionBlockBottomRight()

wxColour wxGrid::GetSelectionForeground ( ) const

Returns the colour used for drawing the selection foreground.

wxGridSelectionModes wxGrid::GetSelectionMode ( ) const

Returns the current selection mode.

See also

SetSelectionMode().

int wxGrid::GetSortingColumn ( ) const

Return the column in which the sorting indicator is currently displayed.

Returns `wxNOT_FOUND` if sorting indicator is not currently displayed at all.

See also

SetSortingColumn()

wxGridTableBase* wxGrid::GetTable ( ) const

Returns a base pointer to the current table object.

The returned pointer is still owned by the grid.

void wxGrid::GoToCell (int *row,* int *col*)

Make the given cell current and ensure it is visible.

This method is equivalent to calling MakeCellVisible() and SetGridCursor() and so, as with the latter, a `wxEVT_GRID_SELECT_CELL` event is generated by it and the selected cell doesn't change if the event is vetoed.

void wxGrid::GoToCell (const wxGridCellCoords & *coords*)

Make the given cell current and ensure it is visible.

This method is equivalent to calling MakeCellVisible() and SetGridCursor() and so, as with the latter, a `wxEVT_GRID_SELECT_CELL` event is generated by it and the selected cell doesn't change if the event is vetoed.

bool wxGrid::GridLinesEnabled ( ) const

Returns true if drawing of grid lines is turned on, false otherwise.

void wxGrid::HideCellEditControl ( )

Hides the in-place cell edit control.

void wxGrid::HideCol (int *col*)

Hides the specified column.

To show the column later you need to call SetColSize() with non-0 width or ShowCol() to restore the previous column width.

If the column is already hidden, this method doesn't do anything.

Parameters

col	The column index.

void wxGrid::HideColLabels ( )

Hides the column labels by calling SetColLabelSize() with a size of 0.

Show labels again by calling that method with a width greater than 0.

void wxGrid::HideRow (int *col*)

Hides the specified row.

To show the row later you need to call SetRowSize() with non-0 width or ShowRow() to restore its original height.

If the row is already hidden, this method doesn't do anything.

Parameters

col	The row index.

void wxGrid::HideRowLabels ( )

Hides the row labels by calling SetRowLabelSize() with a size of 0.

The labels can be shown again by calling SetRowLabelSize() with a width greater than 0.

bool wxGrid::InsertCols (int *pos* = `0`, int *numCols* = `1`, bool *updateLabels* = `true`)

Inserts one or more new columns into a grid with the first new column at the specified position.

Notice that inserting the columns in the grid requires grid table cooperation: when this method is called, grid object begins by requesting the underlying grid table to insert new columns. If this is successful the table notifies the grid and the grid updates the display. For a default grid (one where you have called CreateGrid()) this process is automatic. If you are using a custom grid table (specified with SetTable()) then you must override wxGridTable↩Base::InsertCols() in your derived table class.

Parameters

pos	The position which the first newly inserted column will have.
numCols	The number of columns to insert.
updateLabels	Currently not used.

Returns

true if the columns were successfully inserted, false if an error occurred (most likely the table couldn't be updated).

bool wxGrid::InsertRows (int *pos* = `0`, int *numRows* = `1`, bool *updateLabels* = `true`)

Inserts one or more new rows into a grid with the first new row at the specified position.

Notice that you must implement wxGridTableBase::InsertRows() if you use a grid with a custom table, please see InsertCols() for more information.

Parameters

pos	The position which the first newly inserted row will have.
numRows	The number of rows to insert.
updateLabels	Currently not used.

Returns

true if the rows were successfully inserted, false if an error occurred (most likely the table couldn't be updated).

bool wxGrid::IsCellEditControlEnabled ( ) const

Returns true if the in-place edit control is currently enabled.

bool wxGrid::IsColShown (int *col*) const

Returns true if the specified column is not currently hidden.

bool wxGrid::IsCurrentCellReadOnly ( ) const

Returns true if the current cell is read-only.

See also

SetReadOnly(), IsReadOnly()

bool wxGrid::IsEditable ( ) const

Returns false if the whole grid has been set as read-only or true otherwise.

See EnableEditing() for more information about controlling the editing status of grid cells.

bool wxGrid::IsInSelection (int *row,* int *col*) const

Returns true if the given cell is selected.

bool wxGrid::IsInSelection (const wxGridCellCoords & *coords*) const

Returns true if the given cell is selected.

bool wxGrid::IsReadOnly (int *row,* int *col*) const

Returns true if the cell at the specified location can't be edited.

See also

SetReadOnly(), IsCurrentCellReadOnly()

bool wxGrid::IsRowShown (int *row*) const

Returns true if the specified row is not currently hidden.

bool wxGrid::IsSelection ( ) const

Returns true if there are currently any selected cells, rows, columns or blocks.

bool wxGrid::IsSortingBy (int *col*) const

Return true if this column is currently used for sorting.

See also

GetSortingColumn()

bool wxGrid::IsSortOrderAscending ( ) const

Return true if the current sorting order is ascending or false if it is descending.

It only makes sense to call this function if GetSortingColumn() returns a valid column index and not `wxNOT_FOUND`.

See also

SetSortingColumn()

bool wxGrid::IsVisible (int *row*, int *col*, bool *wholeCellVisible* = `true`) const

Returns true if a cell is either entirely or at least partially visible in the grid window.

By default, the cell must be entirely visible for this function to return true but if *wholeCellVisible* is false, the function returns true even if the cell is only partially visible.

bool wxGrid::IsVisible (const wxGridCellCoords & *coords*, bool *wholeCellVisible* = `true`) const

Returns true if a cell is either entirely or at least partially visible in the grid window.

By default, the cell must be entirely visible for this function to return true but if *wholeCellVisible* is false, the function returns true even if the cell is only partially visible.

void wxGrid::MakeCellVisible (int *row*, int *col*)

Brings the specified cell into the visible grid cell area with minimal scrolling.

Does nothing if the cell is already visible.

void wxGrid::MakeCellVisible (const wxGridCellCoords & *coords*)

Brings the specified cell into the visible grid cell area with minimal scrolling.

Does nothing if the cell is already visible.

bool wxGrid::MoveCursorDown (bool *expandSelection*)

Moves the grid cursor down by one row.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorDownBlock (bool *expandSelection*)

Moves the grid cursor down in the current column such that it skips to the beginning or end of a block of non-empty cells.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorLeft (bool *expandSelection*)

Moves the grid cursor left by one column.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorLeftBlock (bool *expandSelection*)

Moves the grid cursor left in the current row such that it skips to the beginning or end of a block of non-empty cells.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorRight (bool *expandSelection*)

Moves the grid cursor right by one column.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorRightBlock (bool *expandSelection*)

Moves the grid cursor right in the current row such that it skips to the beginning or end of a block of non-empty cells.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorUp (bool *expandSelection*)

Moves the grid cursor up by one row.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MoveCursorUpBlock (bool *expandSelection*)

Moves the grid cursor up in the current column such that it skips to the beginning or end of a block of non-empty cells.

If a block of cells was previously selected it will expand if the argument is true or be cleared if the argument is false.

bool wxGrid::MovePageDown ( )

Moves the grid cursor down by some number of rows so that the previous bottom visible row becomes the top visible row.

bool wxGrid::MovePageUp ( )

Moves the grid cursor up by some number of rows so that the previous top visible row becomes the bottom visible row.

bool wxGrid::ProcessTableMessage (wxGridTableMessage & *msg*)

Receive and handle a message from the table.

void wxGrid::RefreshAttr (int *row*, int *col*)

Invalidates the cached attribute for the given cell.

For efficiency reasons, wxGrid may cache the recently used attributes (currently it caches only the single most recently used one, in fact) which can result in the cell appearance not being refreshed even when the attribute returned by your custom wxGridCellAttrProvider-derived class has changed. To force the grid to refresh the cell attribute, this function may be used. Notice that calling it will not result in actually redrawing the cell, you still need to call wxWindow::RefreshRect() to invalidate the area occupied by the cell in the window to do this. Also note that you don't need to call this function if you store the attributes in wxGrid itself, i.e. use its SetAttr() and similar methods, it is only useful when using a separate custom attributes provider.

Parameters

row	The row of the cell whose attribute needs to be queried again.
col	The column of the cell whose attribute needs to be queried again.

Since

2.9.2

void wxGrid::RegisterDataType (const wxString & *typeName*, wxGridCellRenderer * *renderer*, wxGridCellEditor * *editor*)

Register a new data type.

The data types allow to naturally associate specific renderers and editors to the cells containing values of the given type. For example, the grid automatically registers a data type with the name `wxGRID_VALUE_STRING` which uses wxGridCellStringRenderer and wxGridCellTextEditor as its renderer and editor respectively – this is the data type used by all the cells of the default wxGridStringTable, so this renderer and editor are used by default for all grid cells.

However if a custom table returns `wxGRID_VALUE_BOOL` from its wxGridTableBase::GetTypeName() method, then wxGridCellBoolRenderer and wxGridCellBoolEditor are used for it because the grid also registers a boolean data type with this name.

And as this mechanism is completely generic, you may register your own data types using your own custom renderers and editors. Just remember that the table must identify a cell as being of the given type for them to be used for this cell.

Parameters

typeName	Name of the new type. May be any string, but if the type name is the same as the name of an already registered type, including one of the standard ones (which are `wxGRID_VALUE_STRING`, `wxGRID_VALUE_BOOL`, `wxGRID_VALUE_NUMBER`, `wxGRID_VALUE_FLOAT` and `wxGRID_VALUE_CHOICE`), then the new registration information replaces the previously used renderer and editor.

renderer	The renderer to use for the cells of this type. Its ownership is taken by the grid, i.e. it will call DecRef() on this pointer when it doesn't need it any longer.
editor	The editor to use for the cells of this type. Its ownership is also taken by the grid.

void wxGrid::Render (wxDC & *dc*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, const wxGridCellCoords & *topLeft* = wxGridCellCoords (-1, -1), const wxGridCellCoords & *bottomRight* = wxGridCellCoords (-1, -1), int *style* = wxGRID_DRAW_DEFAULT)

Draws part or all of a wxGrid on a wxDC for printing or display.

Pagination can be accomplished by using sequential Render() calls with appropriate values in wxGridCellCoords topLeft and bottomRight.

Parameters

dc	The wxDC to be drawn on.
pos	The position on the wxDC where rendering should begin. If not specified drawing will begin at the wxDC MaxX() and MaxY().
size	The size of the area on the wxDC that the rendered wxGrid should occupy. If not specified the drawing will be scaled to fit the available dc width or height. The wxGrid's aspect ratio is maintained whether or not size is specified.
topLeft	The top left cell of the block to be drawn. Defaults to (0, 0).
bottomRight	The bottom right cell of the block to be drawn. Defaults to row and column counts.
style	A combination of values from wxGridRenderStyle.

Since

2.9.4

void wxGrid::ResetColPos ( )

Resets the position of the columns to the default.

void wxGrid::SaveEditControlValue ( )

Sets the value of the current grid cell to the current in-place edit control value.

This is called automatically when the grid cursor moves from the current cell to a new cell. It is also a good idea to call this function when closing a grid since any edits to the final cell location will not be saved otherwise.

void wxGrid::SelectAll ( )

Selects all cells in the grid.

void wxGrid::SelectBlock (int *topRow*, int *leftCol*, int *bottomRow*, int *rightCol*, bool *addToSelected* = false)

Selects a rectangular block of cells.

If *addToSelected* is false then any existing selection will be deselected; if true the column will be added to the existing selection.

void wxGrid::SelectBlock (const wxGridCellCoords & *topLeft*, const wxGridCellCoords & *bottomRight*, bool *addToSelected* = false)

Selects a rectangular block of cells.

If *addToSelected* is false then any existing selection will be deselected; if true the column will be added to the existing selection.

void wxGrid::SelectCol (int *col,* bool *addToSelected* = `false`)

Selects the specified column.

If *addToSelected* is false then any existing selection will be deselected; if true the column will be added to the existing selection.

This method won't select anything if the current selection mode is wxGridSelectRows.

void wxGrid::SelectRow (int *row,* bool *addToSelected* = `false`)

Selects the specified row.

If *addToSelected* is false then any existing selection will be deselected; if true the row will be added to the existing selection.

This method won't select anything if the current selection mode is wxGridSelectColumns.

void wxGrid::SetCellAlignment (int *row,* int *col,* int *horiz,* int *vert*)

Sets the horizontal and vertical alignment for grid cell text at the specified location.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

void wxGrid::SetCellAlignment (int *align,* int *row,* int *col*)

Sets the horizontal and vertical alignment for grid cell text at the specified location.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`.

Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

Deprecated Please use SetCellAlignment(row, col, horiz, vert) instead.

void wxGrid::SetCellBackgroundColour (int *row,* int *col,* const **wxColour & *colour*)**

Set the background colour for the given cell or all cells by default.

void wxGrid::SetCellEditor (int *row,* int *col,* **wxGridCellEditor * *editor*)**

Sets the editor for the grid cell at the specified location.

The grid will take ownership of the pointer.

See wxGridCellEditor and the wxGrid Overview for more information about cell editors and renderers.

void wxGrid::SetCellFont (int *row,* int *col,* const **wxFont & *font*)**

Sets the font for text in the grid cell at the specified location.

void wxGrid::SetCellHighlightColour (const wxColour &)

void wxGrid::SetCellHighlightPenWidth (int *width*)

void wxGrid::SetCellHighlightROPenWidth (int *width*)

void wxGrid::SetCellOverflow (int *row,* int *col,* bool *allow*)

Sets the overflow permission of the cell.

void wxGrid::SetCellRenderer (int *row,* int *col,* wxGridCellRenderer * *renderer*)

Sets the renderer for the grid cell at the specified location.

The grid will take ownership of the pointer.

See wxGridCellRenderer and the wxGrid Overview for more information about cell editors and renderers.

void wxGrid::SetCellSize (int *row,* int *col,* int *num_rows,* int *num_cols*)

Set the size of the cell.

Specifying a value of more than 1 in *num_rows* or *num_cols* will make the cell at (*row*, *col*) span the block of the specified size, covering the other cells which would be normally shown in it. Passing 1 for both arguments resets the cell to normal appearance.

See also

GetCellSize()

Parameters

row	The row of the cell.
col	The column of the cell.
num_rows	Number of rows to be occupied by this cell, must be >= 1.
num_cols	Number of columns to be occupied by this cell, must be >= 1.

void wxGrid::SetCellTextColour (int *row,* int *col,* const wxColour & *colour*)

Sets the text colour for the given cell.

void wxGrid::SetCellTextColour (const wxColour & *val,* int *row,* int *col*)

Sets the text colour for the given cell.

Deprecated Please use SetCellTextColour(row, col, colour)

void wxGrid::SetCellTextColour (const wxColour & *colour*)

Sets the text colour for all cells by default.

Deprecated Please use SetDefaultCellTextColour(colour) instead.

void wxGrid::SetCellValue (int *row,* int *col,* const wxString & *s*)

Sets the string value for the cell at the specified location.

For simple applications where a grid object automatically uses a default grid table of string values you use this function together with GetCellValue() to access cell values. For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

See wxGridTableBase::CanSetValueAs() and the wxGrid Overview for more information.

void wxGrid::SetCellValue (const wxGridCellCoords & *coords,* const wxString & *s*)

Sets the string value for the cell at the specified location.

For simple applications where a grid object automatically uses a default grid table of string values you use this function together with GetCellValue() to access cell values. For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

See wxGridTableBase::CanSetValueAs() and the wxGrid Overview for more information.

void wxGrid::SetCellValue (const wxString & *val,* int *row,* int *col*)

Deprecated Please use SetCellValue(int,int,const wxString&) or SetCellValue(const wxGridCellCoords&,const wxString&) instead.

Sets the string value for the cell at the specified location.

For simple applications where a grid object automatically uses a default grid table of string values you use this function together with GetCellValue() to access cell values. For more complex applications where you have derived your own grid table class that contains various data types (e.g. numeric, boolean or user-defined custom types) then you only use this function for those cells that contain string values.

See wxGridTableBase::CanSetValueAs() and the wxGrid Overview for more information.

void wxGrid::SetColAttr (int *col,* wxGridCellAttr * *attr*)

Sets the cell attributes for all cells in the specified column.

For more information about controlling grid cell attributes see the wxGridCellAttr cell attribute class and the wxGrid Overview.

void wxGrid::SetColFormatBool (int *col*)

Sets the specified column to display boolean values.

See also

SetColFormatCustom()

void wxGrid::SetColFormatCustom (int *col,* const wxString & *typeName*)

Sets the specified column to display data in a custom format.

This method provides an alternative to defining a custom grid table which would return *typeName* from its GetTypeName() method for the cells in this column: while it doesn't really change the type of the cells in this column, it does associate the renderer and editor used for the cells of the specified type with them.

See the wxGrid Overview for more information on working with custom data types.

void wxGrid::SetColFormatFloat (int *col,* int *width* = -1, int *precision* = -1)

Sets the specified column to display floating point values with the given width and precision.

See also

SetColFormatCustom()

void wxGrid::SetColFormatNumber (int *col*)

Sets the specified column to display integer values.

See also

SetColFormatCustom()

void wxGrid::SetColLabelAlignment (int *horiz,* int *vert*)

Sets the horizontal and vertical alignment of column label text.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`. Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

void wxGrid::SetColLabelSize (int *height*)

Sets the height of the column labels.

If *height* equals to `wxGRID_AUTOSIZE` then height is calculated automatically so that no label is truncated. Note that this could be slow for a large table.

void wxGrid::SetColLabelTextOrientation (int *textOrientation*)

Sets the orientation of the column labels (either `wxHORIZONTAL` or `wxVERTICAL`).

void wxGrid::SetColLabelValue (int *col,* const wxString & *value*)

Set the value for the given column label.

If you are using a custom grid table you must override wxGridTableBase::SetColLabelValue() for this to have any effect.

void wxGrid::SetColMinimalAcceptableWidth (int *width*)

Sets the minimal *width* to which the user can resize columns.

See also

GetColMinimalAcceptableWidth()

void wxGrid::SetColMinimalWidth (int *col,* int *width*)

Sets the minimal *width* for the specified column *col*.

It is usually best to call this method during grid creation as calling it later will not resize the column to the given minimal width even if it is currently narrower than it.

width must be greater than the minimal acceptable column width as returned by GetColMinimalAcceptableWidth().

void wxGrid::SetColPos (int *colID,* int *newPos*)

Sets the position of the specified column.

void wxGrid::SetColSize (int *col,* int *width*)

Sets the width of the specified column.

Parameters

col	The column index.
width	The new column width in pixels, 0 to hide the column or -1 to fit the column width to its label width.

void wxGrid::SetColSizes (const wxGridSizesInfo & *sizeInfo*)

Restore all columns sizes.

This is usually called with wxGridSizesInfo object previously returned by GetColSizes().

See also

SetRowSizes()

void wxGrid::SetColumnsOrder (const wxArrayInt & *order*)

Sets the positions of all columns at once.

This method takes an array containing the indices of the columns in their display order, i.e. uses the same convention as wxHeaderCtrl::SetColumnsOrder().

void wxGrid::SetDefaultCellAlignment (int *horiz,* int *vert*)

Sets the default horizontal and vertical alignment for grid cell text.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`. Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

void wxGrid::SetDefaultCellBackgroundColour (const wxColour & *colour*)

Sets the default background colour for grid cells.

void wxGrid::SetDefaultCellFont (const wxFont & *font*)

Sets the default font to be used for grid cell text.

void wxGrid::SetDefaultCellOverflow (bool *allow*)

Sets the default overflow permission of the cells.

void wxGrid::SetDefaultCellTextColour (const wxColour & *colour*)

Sets the current default colour for grid cell text.

void wxGrid::SetDefaultColSize (int *width*, bool *resizeExistingCols* = `false`)

Sets the default width for columns in the grid.

This will only affect columns subsequently added to the grid unless *resizeExistingCols* is true.

If *width* is less than GetColMinimalAcceptableWidth(), then the minimal acceptable width is used instead of it.

void wxGrid::SetDefaultEditor (wxGridCellEditor * *editor*)

Sets the default editor for grid cells.

The grid will take ownership of the pointer.

See wxGridCellEditor and the wxGrid Overview for more information about cell editors and renderers.

void wxGrid::SetDefaultRenderer (wxGridCellRenderer * *renderer*)

Sets the default renderer for grid cells.

The grid will take ownership of the pointer.

See wxGridCellRenderer and the wxGrid Overview for more information about cell editors and renderers.

void wxGrid::SetDefaultRowSize (int *height*, bool *resizeExistingRows* = `false`)

Sets the default height for rows in the grid.

This will only affect rows subsequently added to the grid unless *resizeExistingRows* is true.

If *height* is less than GetRowMinimalAcceptableHeight(), then the minimal acceptable height is used instead of it.

void wxGrid::SetGridCursor (int *row*, int *col*)

Set the grid cursor to the specified cell.

The grid cursor indicates the current cell and can be moved by the user using the arrow keys or the mouse.

Calling this function generates a `wxEVT_GRID_SELECT_CELL` event and if the event handler vetoes this event, the cursor is not moved.

This function doesn't make the target call visible, use GoToCell() to do this.

void wxGrid::SetGridCursor (const wxGridCellCoords & *coords*)

Set the grid cursor to the specified cell.

The grid cursor indicates the current cell and can be moved by the user using the arrow keys or the mouse.

Calling this function generates a `wxEVT_GRID_SELECT_CELL` event and if the event handler vetoes this event, the cursor is not moved.

This function doesn't make the target call visible, use GoToCell() to do this.

void wxGrid::SetGridLineColour (const wxColour & *colour*)

Sets the colour used to draw grid lines.

void wxGrid::SetLabelBackgroundColour (const wxColour & *colour*)

Sets the background colour for row and column labels.

void wxGrid::SetLabelFont (const wxFont & *font*)

Sets the font for row and column labels.

void wxGrid::SetLabelTextColour (const wxColour & *colour*)

Sets the colour for row and column label text.

void wxGrid::SetMargins (int *extraWidth,* int *extraHeight*)

Sets the extra margins used around the grid area.

A grid may occupy more space than needed for its data display and this function allows to set how big this extra space is

void wxGrid::SetReadOnly (int *row,* int *col,* bool *isReadOnly* = `true`)

Makes the cell at the specified location read-only or editable.

See also

IsReadOnly()

void wxGrid::SetRowAttr (int *row,* wxGridCellAttr * *attr*)

Sets the cell attributes for all cells in the specified row.

The grid takes ownership of the attribute pointer.

See the wxGridCellAttr class for more information about controlling cell attributes.

void wxGrid::SetRowLabelAlignment (int *horiz,* int *vert*)

Sets the horizontal and vertical alignment of row label text.

Horizontal alignment should be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT`. Vertical alignment should be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

void wxGrid::SetRowLabelSize (int *width*)

Sets the width of the row labels.

If *width* equals `wxGRID_AUTOSIZE` then width is calculated automatically so that no label is truncated. Note that this could be slow for a large table.

void wxGrid::SetRowLabelValue (int *row,* const wxString & *value*)

Sets the value for the given row label.

If you are using a derived grid table you must override wxGridTableBase::SetRowLabelValue() for this to have any effect.

void wxGrid::SetRowMinimalAcceptableHeight (int *height*)

Sets the minimal row *height* used by default.

See SetColMinimalAcceptableWidth() for more information.

void wxGrid::SetRowMinimalHeight (int *row,* int *height*)

Sets the minimal *height* for the specified *row*.

See SetColMinimalWidth() for more information.

void wxGrid::SetRowSize (int *row,* int *height*)

Sets the height of the specified row.

See SetColSize() for more information.

void wxGrid::SetRowSizes (const wxGridSizesInfo & *sizeInfo*)

Restore all rows sizes.

See also

SetColSizes()

void wxGrid::SetScrollLineX (int *x*)

Sets the number of pixels per horizontal scroll increment.

The default is 15.

See also

GetScrollLineX(), GetScrollLineY(), SetScrollLineY()

void wxGrid::SetScrollLineY (int *y*)

Sets the number of pixels per vertical scroll increment.

The default is 15.

See also

GetScrollLineX(), GetScrollLineY(), SetScrollLineX()

void wxGrid::SetSelectionBackground (const wxColour & *c*)

Set the colour to be used for drawing the selection background.

void wxGrid::SetSelectionForeground (const wxColour & *c*)

Set the colour to be used for drawing the selection foreground.

void wxGrid::SetSelectionMode (wxGridSelectionModes *selmode*)

Set the selection behaviour of the grid.

The existing selection is converted to conform to the new mode if possible and discarded otherwise (e.g. any individual selected cells are deselected if the new mode allows only the selection of the entire rows or columns).

void wxGrid::SetSortingColumn (int *col*, bool *ascending* = `true`)

Set the column to display the sorting indicator in and its direction.

Parameters

col	The column to display the sorting indicator in or `wxNOT_FOUND` to remove any currently displayed sorting indicator.
ascending	If true, display the ascending sort indicator, otherwise display the descending sort indicator.

See also

GetSortingColumn(), IsSortOrderAscending()

void wxGrid::SetTabBehaviour (TabBehaviour *behaviour*)

Set the grid's behaviour when the user presses the TAB key.

Pressing the TAB key moves the grid cursor right in the current row, if there is a cell at the right and, similarly, Shift-TAB moves the cursor to the left in the current row if it's not in the first column.

What happens if the cursor can't be moved because it it's already at the beginning or end of the row can be configured using this function, see wxGrid::TabBehaviour documentation for the detailed description.

IF none of the standard behaviours is appropriate, you can always handle `wxEVT_GRID_TABBING` event directly to implement a custom TAB-handling logic.

Since

2.9.5

bool wxGrid::SetTable (wxGridTableBase * *table*, bool *takeOwnership* = `false`, wxGridSelectionModes *selmode* = wxGridSelectCells)

Passes a pointer to a custom grid table to be used by the grid.

This should be called after the grid constructor and before using the grid object. If *takeOwnership* is set to true then the table will be deleted by the wxGrid destructor.

Use this function instead of CreateGrid() when your application involves complex or non-string data or data sets that are too large to fit wholly in memory.

void wxGrid::SetUseNativeColLabels (bool *native* = `true`)

Call this in order to make the column labels use a native look by using wxRendererNative::DrawHeaderButton() internally.

There is no equivalent method for drawing row columns as there is not native look for that. This option is useful when using wxGrid for displaying tables and not as a spread-sheet.

See also

UseNativeColHeader()

void wxGrid::ShowCellEditControl ( )

Displays the in-place cell edit control for the current cell.

void wxGrid::ShowCol (int *col*)

Shows the previously hidden column by resizing it to non-0 size.

The column is shown again with the same width that it had before HideCol() call.

If the column is currently shown, this method doesn't do anything.

See also

HideCol(), SetColSize()

void wxGrid::ShowRow (int *col*)

Shows the previously hidden row.

The row is shown again with the same height that it had before HideRow() call.

If the row is currently shown, this method doesn't do anything.

See also

HideRow(), SetRowSize()

void wxGrid::UnsetSortingColumn ( )

Remove any currently shown sorting indicator.

This is equivalent to calling SetSortingColumn() with `wxNOT_FOUND` first argument.

void wxGrid::UseNativeColHeader (bool *native* = `true`)

Enable the use of native header window for column labels.

If this function is called with true argument, a wxHeaderCtrl is used instead to display the column labels instead of drawing them in wxGrid code itself. This has the advantage of making the grid look and feel perfectly the same as native applications (using SetUseNativeColLabels() the grid can be made to look more natively but it still doesn't feel natively, notably the column resizing and dragging still works slightly differently as it is implemented in wxWidgets itself) but results in different behaviour for column and row headers, for which there is no equivalent function, and, most importantly, is unsuitable for grids with huge numbers of columns as wxHeaderCtrl doesn't support virtual mode. Because of this, by default the grid does not use the native header control but you should call this function to enable it if you are using the grid to display tabular data and don't have thousands of columns in it.

Another difference between the default behaviour and the native header behaviour is that the latter provides the user with a context menu (which appears on right clicking the header) allowing to rearrange the grid columns if CanDragColMove() returns true. If you want to prevent this from happening for some reason, you need to define a handler for `wxEVT_GRID_LABEL_RIGHT_CLICK` event which simply does nothing (in particular doesn't skip the event) as this will prevent the default right click handling from working.

Also note that currently `wxEVT_GRID_LABEL_RIGHT_DCLICK` event is not generated for the column labels if the native columns header is used (but this limitation could possibly be lifted in the future).

int wxGrid::XToCol (int *x*, bool *clipToMinMax* = `false`) const

Returns the column at the given pixel position.

Parameters

x	The x position to evaluate.
clipToMinMax	If true, rather than returning `wxNOT_FOUND`, it returns either the first or last column depending on whether *x* is too far to the left or right respectively.

Returns

The column index or `wxNOT_FOUND`.

int wxGrid::XToEdgeOfCol (int *x*) const

Returns the column whose right hand edge is close to the given logical *x* position.

If no column edge is near to this position `wxNOT_FOUND` is returned.

wxGridCellCoords wxGrid::XYToCell (int *x*, int *y*) const

Translates logical pixel coordinates to the grid cell coordinates.

Notice that this function expects logical coordinates on input so if you use this function in a mouse event handler you need to translate the mouse position, which is expressed in device coordinates, to logical ones.

See also

XToCol(), YToRow()

wxGridCellCoords wxGrid::XYToCell (const wxPoint & *pos*) const

Translates logical pixel coordinates to the grid cell coordinates.

Notice that this function expects logical coordinates on input so if you use this function in a mouse event handler you need to translate the mouse position, which is expressed in device coordinates, to logical ones.

See also

XToCol(), YToRow()

int wxGrid::YToEdgeOfRow (int *y*) const

Returns the row whose bottom edge is close to the given logical *y* position.

If no row edge is near to this position `wxNOT_FOUND` is returned.

int wxGrid::YToRow (int *y*, bool *clipToMinMax* = `false`) const

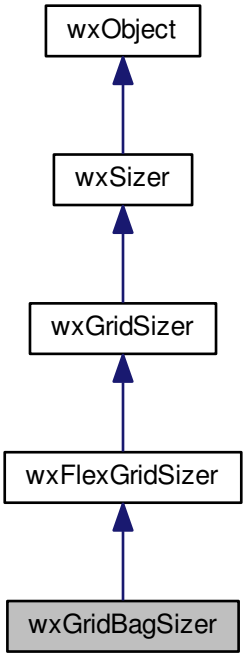
Returns the grid row that corresponds to the logical *y* coordinate.

Returns `wxNOT_FOUND` if there is no row at the *y* position.

21.299 wxGridBagSizer Class Reference

```
#include <wx/gbsizer.h>
```

Inheritance diagram for wxGridBagSizer:



21.299.1 Detailed Description

A wxSizer that can lay out items in a virtual grid like a wxFlexGridSizer but in this case explicit positioning of the items is allowed using wxGBPosition, and items can optionally span more than one row and/or column using wxGBSpan.

Library: wxCore

Category: Window Layout

Public Member Functions

- wxGridBagSizer (int vgap=0, int hgap=0)

  Constructor, with optional parameters to specify the gap between the rows and columns.
- wxSize CalcMin ()

  Called when the managed size of the sizer is needed or when layout needs done.
- wxGBSizerItem * FindItemAtPoint (const wxPoint &pt)

  Return the sizer item located at the point given in pt, or NULL if there is no item at that point.
- wxGBSizerItem * FindItemAtPosition (const wxGBPosition &pos)

  Return the sizer item for the given grid cell, or NULL if there is no item at that position.
- wxGBSizerItem * FindItemWithData (const wxObject *userData)

  Return the sizer item that has a matching user data (it only compares pointer values) or NULL if not found.
- wxSize GetCellSize (int row, int col) const

Get the size of the specified cell, including hgap and vgap.

- wxSize GetEmptyCellSize () const

  Get the size used for cells in the grid with no item.

- void RecalcSizes ()

  Called when the managed size of the sizer is needed or when layout needs done.

- void SetEmptyCellSize (const wxSize &sz)

  Set the size used for cells in the grid with no item.

- wxSizerItem * Add (wxWindow *window, const wxGBPosition &pos, const wxGBSpan &span=wxDefaultSpan, int flag=0, int border=0, wxObject *userData=NULL)

  Adds the given item to the given position.

- wxSizerItem * Add (wxSizer *sizer, const wxGBPosition &pos, const wxGBSpan &span=wxDefaultSpan, int flag=0, int border=0, wxObject *userData=NULL)

  Adds the given item to the given position.

- wxSizerItem * Add (wxGBSizerItem *item)

  Adds the given item to the given position.

- wxSizerItem * Add (int width, int height, const wxGBPosition &pos, const wxGBSpan &span=wxDefaultSpan, int flag=0, int border=0, wxObject *userData=NULL)

  Adds a spacer to the given position.

- bool CheckForIntersection (wxGBSizerItem *item, wxGBSizerItem *excludeItem=NULL)

  Look at all items and see if any intersect (or would overlap) the given item.

- bool CheckForIntersection (const wxGBPosition &pos, const wxGBSpan &span, wxGBSizerItem *excludeItem=NULL)

  Look at all items and see if any intersect (or would overlap) the given item.

- wxGBSizerItem * FindItem (wxWindow *window)

  Find the sizer item for the given window or subsizer, returns NULL if not found.

- wxGBSizerItem * FindItem (wxSizer *sizer)

  Find the sizer item for the given window or subsizer, returns NULL if not found.

- wxGBPosition GetItemPosition (wxWindow *window)

  Get the grid position of the specified item.

- wxGBPosition GetItemPosition (wxSizer *sizer)

  Get the grid position of the specified item.

- wxGBPosition GetItemPosition (size_t index)

  Get the grid position of the specified item.

- wxGBSpan GetItemSpan (wxWindow *window)

  Get the row/col spanning of the specified item.

- wxGBSpan GetItemSpan (wxSizer *sizer)

  Get the row/col spanning of the specified item.

- wxGBSpan GetItemSpan (size_t index)

  Get the row/col spanning of the specified item.

- bool SetItemPosition (wxWindow *window, const wxGBPosition &pos)

  Set the grid position of the specified item.

- bool SetItemPosition (wxSizer *sizer, const wxGBPosition &pos)

  Set the grid position of the specified item.

- bool SetItemPosition (size_t index, const wxGBPosition &pos)

  Set the grid position of the specified item.

- bool SetItemSpan (wxWindow *window, const wxGBSpan &span)

  Set the row/col spanning of the specified item.
- bool SetItemSpan (wxSizer *sizer, const wxGBSpan &span)

  Set the row/col spanning of the specified item.
- bool SetItemSpan (size_t index, const wxGBSpan &span)

  Set the row/col spanning of the specified item.

Additional Inherited Members

21.299.2 Constructor & Destructor Documentation

wxGridBagSizer::wxGridBagSizer (int *vgap* = 0, int *hgap* = 0)

Constructor, with optional parameters to specify the gap between the rows and columns.

21.299.3 Member Function Documentation

wxSizerItem* wxGridBagSizer::Add (wxWindow * *window*, const wxGBPosition & *pos*, const wxGBSpan & *span* = wxDefaultSpan, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Adds the given item to the given position.

Returns

A valid pointer if the item was successfully placed at the given position, or NULL if something was already there.

wxSizerItem* wxGridBagSizer::Add (wxSizer * *sizer*, const wxGBPosition & *pos*, const wxGBSpan & *span* = wxDefaultSpan, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Adds the given item to the given position.

Returns

A valid pointer if the item was successfully placed at the given position, or NULL if something was already there.

wxSizerItem* wxGridBagSizer::Add (wxGBSizerItem * *item*)

Adds the given item to the given position.

Returns

A valid pointer if the item was successfully placed at the given position, or NULL if something was already there.

wxSizerItem* wxGridBagSizer::Add (int *width*, int *height*, const wxGBPosition & *pos*, const wxGBSpan & *span* = wxDefaultSpan, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Adds a spacer to the given position.

width and *height* specify the dimension of the spacer to be added.

Returns

A valid pointer if the spacer was successfully placed at the given position, or NULL if something was already there.

wxSize wxGridBagSizer::CalcMin ( ) [virtual]

Called when the managed size of the sizer is needed or when layout needs done.

Reimplemented from wxFlexGridSizer.

bool wxGridBagSizer::CheckForIntersection (wxGBSizerItem * *item*, wxGBSizerItem * *excludeItem* = NULL)

Look at all items and see if any intersect (or would overlap) the given item.

Returns true if so, false if there would be no overlap. If an *excludeItem* is given then it will not be checked for intersection, for example it may be the item we are checking the position of.

bool wxGridBagSizer::CheckForIntersection (const wxGBPosition & *pos*, const wxGBSpan & *span*, wxGBSizerItem * *excludeItem* = NULL)

Look at all items and see if any intersect (or would overlap) the given item.

Returns true if so, false if there would be no overlap. If an *excludeItem* is given then it will not be checked for intersection, for example it may be the item we are checking the position of.

wxGBSizerItem* wxGridBagSizer::FindItem (wxWindow * *window*)

Find the sizer item for the given window or subsizer, returns NULL if not found.

(non-recursive)

wxGBSizerItem* wxGridBagSizer::FindItem (wxSizer * *sizer*)

Find the sizer item for the given window or subsizer, returns NULL if not found.

(non-recursive)

wxGBSizerItem* wxGridBagSizer::FindItemAtPoint (const wxPoint & *pt*)

Return the sizer item located at the point given in pt, or NULL if there is no item at that point.

The (x,y) coordinates in *pt* correspond to the client coordinates of the window using the sizer for layout. (non-recursive)

wxGBSizerItem* wxGridBagSizer::FindItemAtPosition (const wxGBPosition & *pos*)

Return the sizer item for the given grid cell, or NULL if there is no item at that position.

(non-recursive)

wxGBSizerItem* wxGridBagSizer::FindItemWithData (const wxObject * *userData*)

Return the sizer item that has a matching user data (it only compares pointer values) or NULL if not found.

(non-recursive)

wxSize wxGridBagSizer::GetCellSize (int *row,* int *col*) const

Get the size of the specified cell, including hgap and vgap.

Only valid after window layout has been performed.

wxSize wxGridBagSizer::GetEmptyCellSize ( ) const

Get the size used for cells in the grid with no item.

wxGBPosition wxGridBagSizer::GetItemPosition (wxWindow * *window*)

Get the grid position of the specified item.

wxGBPosition wxGridBagSizer::GetItemPosition (wxSizer * *sizer*)

Get the grid position of the specified item.

wxGBPosition wxGridBagSizer::GetItemPosition (size_t *index*)

Get the grid position of the specified item.

wxGBSpan wxGridBagSizer::GetItemSpan (wxWindow * *window*)

Get the row/col spanning of the specified item.

wxGBSpan wxGridBagSizer::GetItemSpan (wxSizer * *sizer*)

Get the row/col spanning of the specified item.

wxGBSpan wxGridBagSizer::GetItemSpan (size_t *index*)

Get the row/col spanning of the specified item.

void wxGridBagSizer::RecalcSizes ( ) [virtual]

Called when the managed size of the sizer is needed or when layout needs done.

Reimplemented from wxFlexGridSizer.

void wxGridBagSizer::SetEmptyCellSize (const wxSize & *sz*)

Set the size used for cells in the grid with no item.

bool wxGridBagSizer::SetItemPosition (wxWindow * *window,* const wxGBPosition & *pos*)

Set the grid position of the specified item.

Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

bool wxGridBagSizer::SetItemPosition (wxSizer * *sizer,* const wxGBPosition & *pos*)

Set the grid position of the specified item.

Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

bool wxGridBagSizer::SetItemPosition (size_t *index,* const wxGBPosition & *pos*)

Set the grid position of the specified item.

Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

bool wxGridBagSizer::SetItemSpan (wxWindow * *window,* const wxGBSpan & *span*)

Set the row/col spanning of the specified item.

Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

bool wxGridBagSizer::SetItemSpan (wxSizer * *sizer,* const wxGBSpan & *span*)

Set the row/col spanning of the specified item.

Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

bool wxGridBagSizer::SetItemSpan (size_t *index,* const wxGBSpan & *span*)

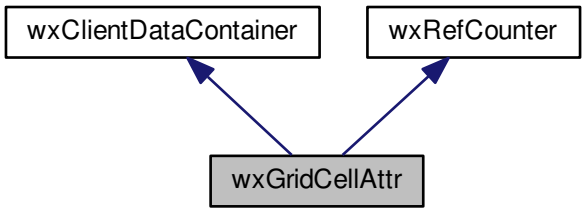
Set the row/col spanning of the specified item.

Returns true on success. If the move is not allowed (because an item is already there) then false is returned.

21.300 wxGridCellAttr Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellAttr:



21.300.1 Detailed Description

This class can be used to alter the cells' appearance in the grid by changing their attributes from the defaults.

An object of this class may be returned by wxGridTableBase::GetAttr().

Library: wxAdvanced

Category: Grid Related Classes

Public Types

- enum wxAttrKind {
  Any,
  Cell,
  Row,
  Col }

  Kind of the attribute to retrieve.

Public Member Functions

- wxGridCellAttr (wxGridCellAttr *attrDefault=NULL)

  Default constructor.
- wxGridCellAttr (const wxColour &colText, const wxColour &colBack, const wxFont &font, int hAlign, int vAlign)

  Constructor specifying some of the often used attributes.
- wxGridCellAttr * Clone () const

  Creates a new copy of this object.
- void DecRef ()

  This class is reference counted: it is created with ref count of 1, so calling DecRef() once will delete it.
- void GetAlignment (int *hAlign, int *vAlign) const

  Get the alignment to use for the cell with the given attribute.
- const wxColour & GetBackgroundColour () const

  Returns the background colour.
- wxGridCellEditor * GetEditor (const wxGrid *grid, int row, int col) const

  Returns the cell editor.
- const wxFont & GetFont () const

  Returns the font.
- void GetNonDefaultAlignment (int *hAlign, int *vAlign) const

  Get the alignment defined by this attribute.
- wxGridCellRenderer * GetRenderer (const wxGrid *grid, int row, int col) const

  Returns the cell renderer.
- const wxColour & GetTextColour () const

  Returns the text colour.
- bool HasAlignment () const

  Returns true if this attribute has a valid alignment set.
- bool HasBackgroundColour () const

  Returns true if this attribute has a valid background colour set.
- bool HasEditor () const

  Returns true if this attribute has a valid cell editor set.
- bool HasFont () const

  Returns true if this attribute has a valid font set.
- bool HasRenderer () const

  Returns true if this attribute has a valid cell renderer set.
- bool HasTextColour () const

  Returns true if this attribute has a valid text colour set.
- void IncRef ()

This class is reference counted: it is created with ref count of 1, so calling DecRef() once will delete it.

- bool IsReadOnly () const

  Returns true if this cell is set as read-only.
- void SetAlignment (int hAlign, int vAlign)

  Sets the alignment.
- void SetBackgroundColour (const wxColour &colBack)

  Sets the background colour.
- void SetDefAttr (wxGridCellAttr *defAttr)
- void SetEditor (wxGridCellEditor *editor)

  Sets the editor to be used with the cells with this attribute.
- void SetFont (const wxFont &font)

  Sets the font.
- void SetReadOnly (bool isReadOnly=true)

  Sets the cell as read-only.
- void SetRenderer (wxGridCellRenderer *renderer)

  Sets the renderer to be used for cells with this attribute.
- void SetTextColour (const wxColour &colText)

  Sets the text colour.

Protected Member Functions

- virtual ~wxGridCellAttr ()

  The destructor is private because only DecRef() can delete us.

21.300.2 Member Enumeration Documentation

enum wxGridCellAttr::wxAttrKind

Kind of the attribute to retrieve.

See also

wxGridCellAttrProvider::GetAttr(), wxGridTableBase::GetAttr()

Enumerator

Any Return the combined effective attribute for the cell.

Cell Return the attribute explicitly set for this cell.

Row Return the attribute set for this cells row.

Col Return the attribute set for this cells column.

21.300.3 Constructor & Destructor Documentation

wxGridCellAttr::wxGridCellAttr (wxGridCellAttr * *attrDefault* = NULL)

Default constructor.

wxGridCellAttr::wxGridCellAttr (const wxColour & *colText*, const wxColour & *colBack*, const wxFont & *font*, int *hAlign*, int *vAlign*)

Constructor specifying some of the often used attributes.

virtual wxGridCellAttr::~wxGridCellAttr ( ) [protected], [virtual]

The destructor is private because only DecRef() can delete us.

21.300.4 Member Function Documentation

wxGridCellAttr* wxGridCellAttr::Clone ( ) const

Creates a new copy of this object.

void wxGridCellAttr::DecRef ( )

This class is reference counted: it is created with ref count of 1, so calling DecRef() once will delete it.

Calling IncRef() allows to lock it until the matching DecRef() is called.

void wxGridCellAttr::GetAlignment (int * *hAlign*, int * *vAlign*) const

Get the alignment to use for the cell with the given attribute.

If this attribute doesn't specify any alignment, the default attribute alignment is used (which can be changed using wxGrid::SetDefaultCellAlignment() but is left and top by default).

Notice that *hAlign* and *vAlign* values are always overwritten by this function, use GetNonDefaultAlignment() if this is not desirable.

Parameters

hAlign	Horizontal alignment is returned here if this argument is non-NULL. It is one of wxALIGN_LEFT, wxALIGN_CENTRE or wxALIGN_RIGHT.
vAlign	Vertical alignment is returned here if this argument is non-NULL. It is one of wxALIGN_TOP, wxALIGN_CENTRE or wxALIGN_BOTTOM.

const wxColour& wxGridCellAttr::GetBackgroundColour ( ) const

Returns the background colour.

wxGridCellEditor* wxGridCellAttr::GetEditor (const wxGrid * *grid*, int *row*, int *col*) const

Returns the cell editor.

const wxFont& wxGridCellAttr::GetFont ( ) const

Returns the font.

void wxGridCellAttr::GetNonDefaultAlignment (int * *hAlign*, int * *vAlign*) const

Get the alignment defined by this attribute.

Unlike GetAlignment() this function only modifies *hAlign* and *vAlign* if this attribute does define a non-default alignment. This means that they must be initialized before calling this function and that their values will be preserved unchanged if they are different from wxALIGN_INVALID.

For example, the following fragment can be used to use the cell alignment if one is defined but right-align its contents by default (instead of left-aligning it by default) while still using the default vertical alignment:

```
int hAlign = wxALIGN_RIGHT,
    vAlign = wxALIGN_INVALID;
attr.GetNonDefaultAlignment(&hAlign, &vAlign);
```

Since

2.9.1

wxGridCellRenderer* wxGridCellAttr::GetRenderer (const wxGrid * *grid,* int *row,* int *col*) const

Returns the cell renderer.

const wxColour& wxGridCellAttr::GetTextColour ( ) const

Returns the text colour.

bool wxGridCellAttr::HasAlignment ( ) const

Returns true if this attribute has a valid alignment set.

bool wxGridCellAttr::HasBackgroundColour ( ) const

Returns true if this attribute has a valid background colour set.

bool wxGridCellAttr::HasEditor ( ) const

Returns true if this attribute has a valid cell editor set.

bool wxGridCellAttr::HasFont ( ) const

Returns true if this attribute has a valid font set.

bool wxGridCellAttr::HasRenderer ( ) const

Returns true if this attribute has a valid cell renderer set.

bool wxGridCellAttr::HasTextColour ( ) const

Returns true if this attribute has a valid text colour set.

void wxGridCellAttr::IncRef ( )

This class is reference counted: it is created with ref count of 1, so calling DecRef() once will delete it.

Calling IncRef() allows to lock it until the matching DecRef() is called.

bool wxGridCellAttr::IsReadOnly ( ) const

Returns true if this cell is set as read-only.

void wxGridCellAttr::SetAlignment (int *hAlign,* int *vAlign*)

Sets the alignment.

hAlign can be one of `wxALIGN_LEFT`, `wxALIGN_CENTRE` or `wxALIGN_RIGHT` and *vAlign* can be one of `wxALIGN_TOP`, `wxALIGN_CENTRE` or `wxALIGN_BOTTOM`.

void wxGridCellAttr::SetBackgroundColour (const wxColour & *colBack*)

Sets the background colour.

void wxGridCellAttr::SetDefAttr (wxGridCellAttr * *defAttr*)

Todo Needs documentation.

void wxGridCellAttr::SetEditor (wxGridCellEditor * *editor*)

Sets the editor to be used with the cells with this attribute.

void wxGridCellAttr::SetFont (const wxFont & *font*)

Sets the font.

void wxGridCellAttr::SetReadOnly (bool *isReadOnly* = `true`)

Sets the cell as read-only.

void wxGridCellAttr::SetRenderer (wxGridCellRenderer * *renderer*)

Sets the renderer to be used for cells with this attribute.

Takes ownership of the pointer.

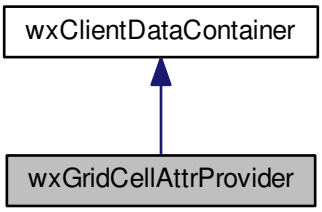
void wxGridCellAttr::SetTextColour (const wxColour & *colText*)

Sets the text colour.

21.301 wxGridCellAttrProvider Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellAttrProvider:



21.301.1 Detailed Description

Class providing attributes to be used for the grid cells.

This class both defines an interface which grid cell attributes providers should implement – and which can be implemented differently in derived classes – and a default implementation of this interface which is often good enough to be used without modification, especially with not very large grids for which the efficiency of attributes storage hardly matters (see the discussion below).

An object of this class can be associated with a wxGrid using wxGridTableBase::SetAttrProvider() but it's not necessary to call it if you intend to use the default provider as it is used by wxGridTableBase by default anyhow.

Notice that while attributes provided by this class can be set for individual cells using SetAttr() or the entire rows or columns using SetRowAttr() and SetColAttr() they are always retrieved using GetAttr() function.

The default implementation of this class stores the attributes passed to its SetAttr(), SetRowAttr() and SetColAttr() in a straightforward way. A derived class may use its knowledge about how the attributes are used in your program to implement it much more efficiently: for example, using a special background colour for all even-numbered rows can be implemented by simply returning the same attribute from GetAttr() if the row number is even instead of having to store N/2 row attributes where N is the total number of rows in the grid.

Notice that objects of this class can't be copied.

Public Member Functions

- wxGridCellAttrProvider ()

  Trivial default constructor.
- virtual ~wxGridCellAttrProvider ()

  Destructor releases any attributes held by this class.
- virtual wxGridCellAttr * GetAttr (int row, int col, wxGridCellAttr::wxAttrKind kind) const

  Get the attribute to use for the specified cell.

- virtual void SetAttr (wxGridCellAttr *attr, int row, int col)

  Setting attributes.
- virtual void SetRowAttr (wxGridCellAttr *attr, int row)

  Set attribute for the specified row.
- virtual void SetColAttr (wxGridCellAttr *attr, int col)

  Set attribute for the specified column.

- virtual const wxGridColumnHeaderRenderer & GetColumnHeaderRenderer (int col)

Getting header renderers.

- virtual const wxGridRowHeaderRenderer & GetRowHeaderRenderer (int row)

  Return the renderer used for drawing row headers.

- virtual const wxGridCornerHeaderRenderer & GetCornerRenderer ()

  Return the renderer used for drawing the corner window.

21.301.2 Constructor & Destructor Documentation

wxGridCellAttrProvider::wxGridCellAttrProvider ( )

Trivial default constructor.

virtual wxGridCellAttrProvider::~wxGridCellAttrProvider ( ) [virtual]

Destructor releases any attributes held by this class.

21.301.3 Member Function Documentation

virtual wxGridCellAttr* wxGridCellAttrProvider::GetAttr (int *row,* int *col,* wxGridCellAttr::wxAttrKind *kind*) const [virtual]

Get the attribute to use for the specified cell.

If wxGridCellAttr::Any is used as *kind* value, this function combines the attributes set for this cell using SetAttr() and those for its row or column (set with SetRowAttr() or SetColAttr() respectively), with the cell attribute having the highest precedence.

Notice that the caller must call DecRef() on the returned pointer if it is non-NULL.

Parameters

row	The row of the cell.
col	The column of the cell.
kind	The kind of the attribute to return.

Returns

The attribute to use which should be DecRef()'d by caller or NULL if no attributes are defined for this cell.

virtual const wxGridColumnHeaderRenderer& wxGridCellAttrProvider::GetColumnHeaderRenderer (int *col*) [virtual]

Getting header renderers.

These functions return the renderers for the given row or column header label and the corner window. Unlike cell attributes, these objects are not reference counted and are never NULL so they are returned by reference and not pointer and DecRef() shouldn't (and can't) be called for them.

All these functions were added in wxWidgets 2.9.1. Return the renderer used for drawing column headers.

By default wxGridColumnHeaderRendererDefault is returned.

See also

wxGrid::SetUseNativeColLabels(), wxGrid::UseNativeColHeader()

Since

2.9.1

virtual const wxGridCornerHeaderRenderer& wxGridCellAttrProvider::GetCornerRenderer ( ) [virtual]

Return the renderer used for drawing the corner window.

By default wxGridCornerHeaderRendererDefault is returned.

Since

2.9.1

virtual const wxGridRowHeaderRenderer& wxGridCellAttrProvider::GetRowHeaderRenderer (int *row*) [virtual]

Return the renderer used for drawing row headers.

By default wxGridRowHeaderRendererDefault is returned.

Since

2.9.1

virtual void wxGridCellAttrProvider::SetAttr (wxGridCellAttr * *attr,* int *row,* int *col*) [virtual]

Setting attributes.

All these functions take ownership of the attribute passed to them, i.e. will call DecRef() on it themselves later and so it should not be destroyed by the caller. And the attribute can be NULL to reset a previously set value.Set attribute for the specified cell.

virtual void wxGridCellAttrProvider::SetColAttr (wxGridCellAttr * *attr,* int *col*) [virtual]

Set attribute for the specified column.

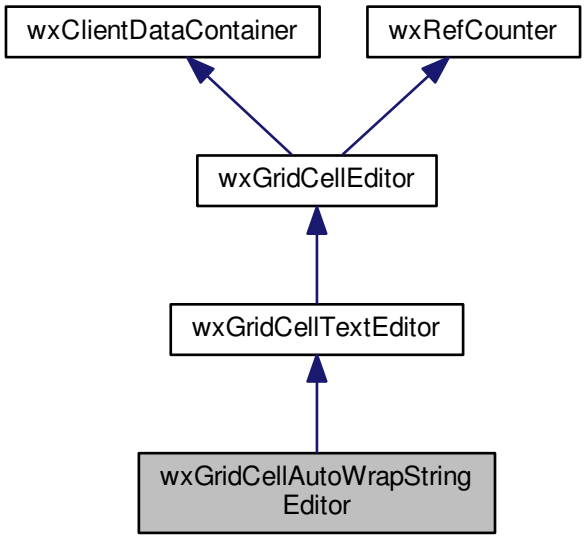
virtual void wxGridCellAttrProvider::SetRowAttr (wxGridCellAttr * *attr,* int *row*) [virtual]

Set attribute for the specified row.

21.302 wxGridCellAutoWrapStringEditor Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridCellAutoWrapStringEditor:



21.302.1 Detailed Description

Grid cell editor for wrappable string/text data.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellBoolEditor, wxGridCellChoiceEditor, wxGridCellEnumEditor, wxGridCellFloat↩ Editor, wxGridCellNumberEditor, wxGridCellTextEditor

Public Member Functions

- wxGridCellAutoWrapStringEditor ()

Additional Inherited Members

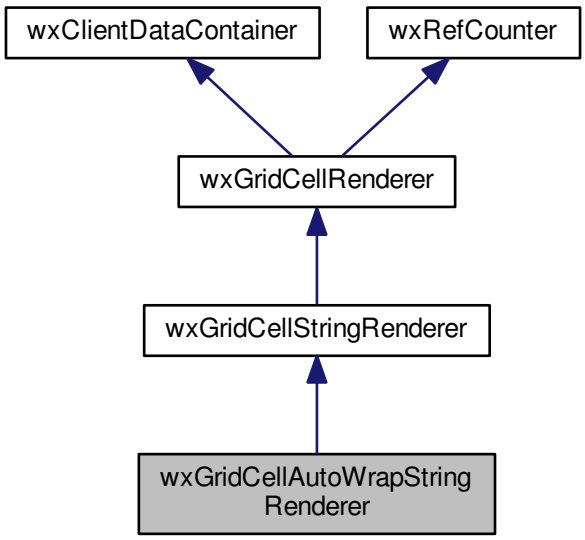
21.302.2 Constructor & Destructor Documentation

wxGridCellAutoWrapStringEditor::wxGridCellAutoWrapStringEditor ( )

21.303 wxGridCellAutoWrapStringRenderer Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridCellAutoWrapStringRenderer:



21.303.1 Detailed Description

This class may be used to format string data in a cell.

The too long lines are wrapped to be shown entirely at word boundaries.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellBoolRenderer, wxGridCellDateTimeRenderer, wxGridCellEnumRenderer, wxGridCellFloatRenderer, wxGridCellNumberRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellAutoWrapStringRenderer ()

  Default constructor.

Additional Inherited Members

21.303.2 Constructor & Destructor Documentation

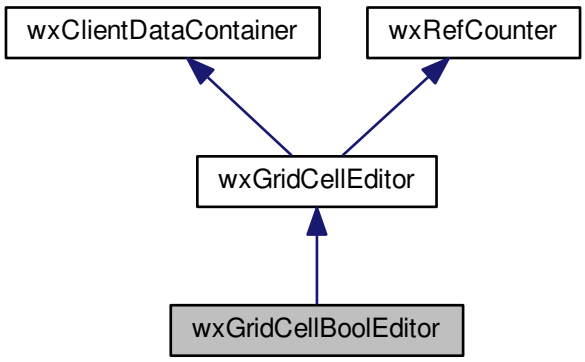
wxGridCellAutoWrapStringRenderer::wxGridCellAutoWrapStringRenderer ( )

Default constructor.

21.304 wxGridCellBoolEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellBoolEditor:



21.304.1 Detailed Description

Grid cell editor for boolean data.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellAutoWrapStringEditor, wxGridCellChoiceEditor, wxGridCellEnumEditor, wxGridCellFloatEditor, wxGridCellNumberEditor, wxGridCellTextEditor

Public Member Functions

- wxGridCellBoolEditor ()

  Default constructor.

Static Public Member Functions

- static bool IsTrueValue (const wxString &value)

  Returns true if the given value is equal to the string representation of the truth value we currently use (see UseStringValues()).

- static void UseStringValues (const wxString &valueTrue="1", const wxString &valueFalse=wxEmptyString)

  This method allows you to customize the values returned by GetValue() for the cell using this editor.

Additional Inherited Members

21.304.2 Constructor & Destructor Documentation

wxGridCellBoolEditor::wxGridCellBoolEditor ( )

Default constructor.

21.304.3 Member Function Documentation

static bool wxGridCellBoolEditor::IsTrueValue (const wxString & *value*) `[static]`

Returns true if the given *value* is equal to the string representation of the truth value we currently use (see UseStringValues()).

static void wxGridCellBoolEditor::UseStringValues (const wxString & *valueTrue* = "1", const wxString & *valueFalse* = wxEmptyString) `[static]`

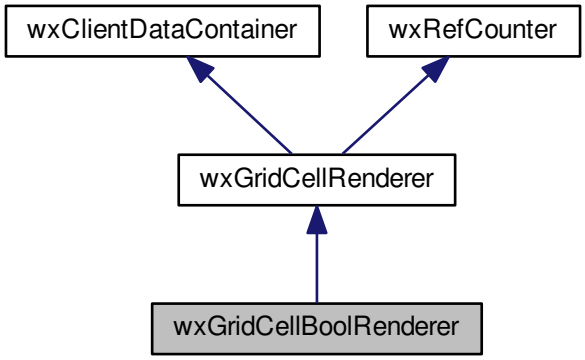
This method allows you to customize the values returned by GetValue() for the cell using this editor.

By default, the default values of the arguments are used, i.e. "1" is returned if the cell is checked and an empty string otherwise.

21.305 wxGridCellBoolRenderer Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridCellBoolRenderer:



21.305.1 Detailed Description

This class may be used to format boolean data in a cell.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellAutoWrapStringRenderer, wxGridCellDateTimeRenderer, wxGridCellEnumRenderer, wxGridCellFloatRenderer, wxGridCellNumberRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellBoolRenderer ()

  Default constructor.

Additional Inherited Members

21.305.2 Constructor & Destructor Documentation

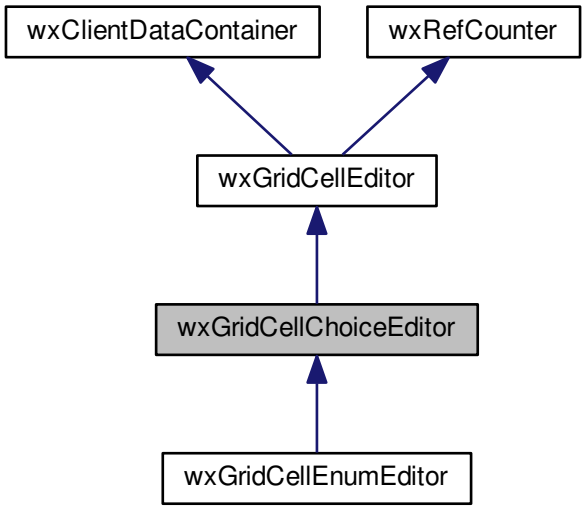
wxGridCellBoolRenderer::wxGridCellBoolRenderer ( )

Default constructor.

21.306 wxGridCellChoiceEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellChoiceEditor:



21.306.1 Detailed Description

Grid cell editor for string data providing the user a choice from a list of strings.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellAutoWrapStringEditor, wxGridCellBoolEditor, wxGridCellEnumEditor, wxGridCellFloatEditor, wxGridCellNumberEditor, wxGridCellTextEditor

Public Member Functions

- wxGridCellChoiceEditor (size_t count=0, const wxString choices[]=NULL, bool allowOthers=false)

  Choice cell renderer ctor.
- wxGridCellChoiceEditor (const wxArrayString &choices, bool allowOthers=false)

  Choice cell renderer ctor.
- virtual void SetParameters (const wxString &params)

  Parameters string format is "item1[,item2[...,itemN]]".

Additional Inherited Members

21.306.2 Constructor & Destructor Documentation

wxGridCellChoiceEditor::wxGridCellChoiceEditor (size_t *count* = `0`, const wxString *choices[]* = `NULL`, bool *allowOthers* = `false`)

Choice cell renderer ctor.

Parameters

count	Number of strings from which the user can choose.
choices	An array of strings from which the user can choose.
allowOthers	If allowOthers is true, the user can type a string not in choices array.

wxGridCellChoiceEditor::wxGridCellChoiceEditor (const wxArrayString & *choices*, bool *allowOthers* = `false`)

Choice cell renderer ctor.

Parameters

choices	An array of strings from which the user can choose.
allowOthers	If allowOthers is true, the user can type a string not in choices array.

21.306.3 Member Function Documentation

virtual void wxGridCellChoiceEditor::SetParameters (const wxString & *params*) `[virtual]`

Parameters string format is "item1[,item2[...,itemN]]".

21.307 wxGridCellCoords Class Reference

`#include <wx/grid.h>`

21.307.1 Detailed Description

Represents coordinates of a grid cell.

An object of this class is simply a (row, column) pair.

Public Member Functions

- wxGridCellCoords ()
  Default constructor initializes the object to invalid state.
- wxGridCellCoords (int row, int col)
  Constructor taking a row and a column.
- int GetRow () const
  Return the row of the coordinate.
- void SetRow (int n)
  Set the row of the coordinate.
- int GetCol () const
  Return the column of the coordinate.
- void SetCol (int n)
  Set the column of the coordinate.
- void Set (int row, int col)
  Set the row and column of the coordinate.
- wxGridCellCoords & operator= (const wxGridCellCoords &other)
  Assignment operator for coordinate types.
- bool operator== (const wxGridCellCoords &other) const
  Equality operator.
- bool operator!= (const wxGridCellCoords &other) const
  Inequality operator.
- bool operator! () const
  Checks whether the coordinates are invalid.

21.307.2 Constructor & Destructor Documentation

wxGridCellCoords::wxGridCellCoords ( )

Default constructor initializes the object to invalid state.

Initially the row and column are both invalid (-1) and so operator!() for an uninitialized wxGridCellCoords returns false.

wxGridCellCoords::wxGridCellCoords (int *row,* int *col*)

Constructor taking a row and a column.

21.307.3 Member Function Documentation

int wxGridCellCoords::GetCol ( ) const

Return the column of the coordinate.

int wxGridCellCoords::GetRow ( ) const

Return the row of the coordinate.

bool wxGridCellCoords::operator! ( ) const

Checks whether the coordinates are invalid.

Returns false only if both row and column are -1. Notice that if either row or column (but not both) are -1, this method returns true even if the object is invalid. This is done because objects in such state should actually never exist, i.e. either both coordinates should be -1 or none of them should be -1.

bool wxGridCellCoords::operator!= (const wxGridCellCoords & *other*) const

Inequality operator.

wxGridCellCoords& wxGridCellCoords::operator= (const wxGridCellCoords & *other*)

Assignment operator for coordinate types.

bool wxGridCellCoords::operator== (const wxGridCellCoords & *other*) const

Equality operator.

void wxGridCellCoords::Set (int *row,* int *col*)

Set the row and column of the coordinate.

void wxGridCellCoords::SetCol (int *n*)

Set the column of the coordinate.

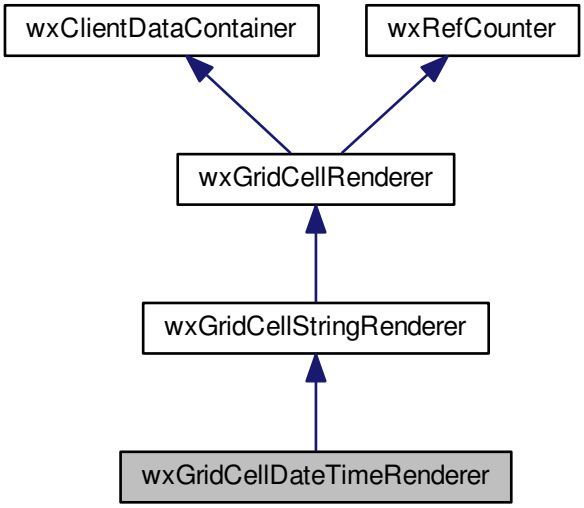
void wxGridCellCoords::SetRow (int *n*)

Set the row of the coordinate.

21.308 wxGridCellDateTimeRenderer Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridCellDateTimeRenderer:



21.308.1 Detailed Description

This class may be used to format a date/time data in a cell.

The class wxDateTime is used internally to display the local date/time or to parse the string date entered in the cell thanks to the defined format.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellAutoWrapStringRenderer, wxGridCellBoolRenderer, wxGridCellEnumRenderer, wxGridCellFloatRenderer, wxGridCellNumberRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellDateTimeRenderer (const wxString &outformat=wxDefaultDateTimeFormat, const wxString &informat=wxDefaultDateTimeFormat)

  Date/time renderer constructor.
- virtual void SetParameters (const wxString &params)

  Sets the strptime()-like format string which will be used to parse the date/time.

Additional Inherited Members

21.308.2 Constructor & Destructor Documentation

wxGridCellDateTimeRenderer::wxGridCellDateTimeRenderer (const wxString & *outformat* = `wxDefaultDateTimeFormat`**, const wxString & *informat* =** `wxDefaultDateTimeFormat` **)**

Date/time renderer constructor.

Parameters

outformat	strptime()-like format string used the parse the output date/time.
informat	strptime()-like format string used to parse the string entered in the cell.

21.308.3 Member Function Documentation

virtual void wxGridCellDateTimeRenderer::SetParameters (const wxString & *params*) `[virtual]`

Sets the strptime()-like format string which will be used to parse the date/time.

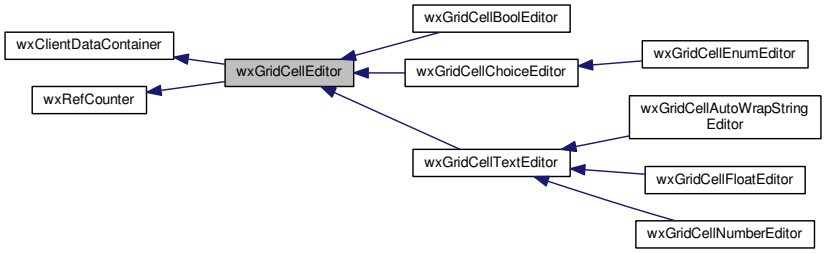
Parameters

params	strptime()-like format string used to parse the date/time.

21.309 wxGridCellEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellEditor:



21.309.1 Detailed Description

This class is responsible for providing and manipulating the in-place edit controls for the grid.

Instances of wxGridCellEditor (actually, instances of derived classes since it is an abstract class) can be associated with the cell attributes for individual cells, rows, columns, or even for the entire grid.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellAutoWrapStringEditor, wxGridCellBoolEditor, wxGridCellChoiceEditor, wxGridCellEnumEditor, wxGridCellFloatEditor, wxGridCellNumberEditor, wxGridCellTextEditor

Public Member Functions

- wxGridCellEditor ()

Default constructor.

- virtual void BeginEdit (int row, int col, wxGrid *grid)=0

  Fetch the value from the table and prepare the edit control to begin editing.

- virtual wxGridCellEditor * Clone () const =0

  Create a new object which is the copy of this one.

- virtual void Create (wxWindow *parent, wxWindowID id, wxEvtHandler *evtHandler)=0

  Creates the actual edit control.

- virtual void Destroy ()

  Final cleanup.

- virtual bool EndEdit (int row, int col, const wxGrid *grid, const wxString &oldval, wxString *newval)=0

  End editing the cell.

- virtual void ApplyEdit (int row, int col, wxGrid *grid)=0

  Effectively save the changes in the grid.

- virtual void HandleReturn (wxKeyEvent &event)

  Some types of controls on some platforms may need some help with the Return key.

- bool IsCreated ()

  Returns true if the edit control has been created.

- virtual void PaintBackground (wxDC &dc, const wxRect &rectCell, wxGridCellAttr &attr)

  Draws the part of the cell not occupied by the control: the base class version just fills it with background colour from the attribute.

- virtual void Reset ()=0

  Reset the value in the control back to its starting value.

- virtual void SetSize (const wxRect &rect)

  Size and position the edit control.

- virtual void Show (bool show, wxGridCellAttr *attr=NULL)

  Show or hide the edit control, use the specified attributes to set colours/fonts for it.

- virtual void StartingClick ()

  If the editor is enabled by clicking on the cell, this method will be called.

- virtual void StartingKey (wxKeyEvent &event)

  If the editor is enabled by pressing keys on the grid, this will be called to let the editor do something about that first key if desired.

- virtual wxString GetValue () const =0

  Returns the value currently in the editor control.

- wxControl * GetControl () const

  Get the wxControl used by this editor.

- void SetControl (wxControl *control)

  Set the wxControl that will be used by this cell editor for editing the value.

Protected Member Functions

- virtual ~wxGridCellEditor ()

  The destructor is private because only DecRef() can delete us.

21.309.2 Constructor & Destructor Documentation

wxGridCellEditor::wxGridCellEditor ( )

Default constructor.

virtual wxGridCellEditor::~wxGridCellEditor ( ) [protected], [virtual]

The destructor is private because only DecRef() can delete us.

21.309.3 Member Function Documentation

virtual void wxGridCellEditor::ApplyEdit (int *row,* int *col,* wxGrid * *grid*) [pure virtual]

Effectively save the changes in the grid.

This function should save the value of the control in the grid. It is called only after EndEdit() returns true.

virtual void wxGridCellEditor::BeginEdit (int *row,* int *col,* wxGrid * *grid*) [pure virtual]

Fetch the value from the table and prepare the edit control to begin editing.

This function should save the original value of the grid cell at the given *row* and *col* and show the control allowing the user to change it.

See also

EndEdit()

virtual wxGridCellEditor* wxGridCellEditor::Clone ( ) const [pure virtual]

Create a new object which is the copy of this one.

virtual void wxGridCellEditor::Create (wxWindow * *parent,* wxWindowID *id,* wxEvtHandler * *evtHandler*) [pure virtual]

Creates the actual edit control.

virtual void wxGridCellEditor::Destroy ( ) [virtual]

Final cleanup.

virtual bool wxGridCellEditor::EndEdit (int *row,* int *col,* const wxGrid * *grid,* const wxString & *oldval,* wxString * *newval*) [pure virtual]

End editing the cell.

This function must check if the current value of the editing control is valid and different from the original value (available as *oldval* in its string form and possibly saved internally using its real type by BeginEdit()). If it isn't, it just returns false, otherwise it must do the following:

- Save the new value internally so that ApplyEdit() could apply it.
- Fill *newval* (which is never NULL) with the string representation of the new value.
- Return true

Notice that it must *not* modify the grid as the change could still be vetoed.

If the user-defined wxEVT_GRID_CELL_CHANGING event handler doesn't veto this change, ApplyEdit() will be called next.

wxControl* wxGridCellEditor::GetControl ( ) const

Get the wxControl used by this editor.

virtual wxString wxGridCellEditor::GetValue ( ) const [pure virtual]

Returns the value currently in the editor control.

virtual void wxGridCellEditor::HandleReturn (wxKeyEvent & *event*) [virtual]

Some types of controls on some platforms may need some help with the Return key.

bool wxGridCellEditor::IsCreated ( )

Returns true if the edit control has been created.

virtual void wxGridCellEditor::PaintBackground (wxDC & *dc,* const wxRect & *rectCell,* wxGridCellAttr & *attr*) [virtual]

Draws the part of the cell not occupied by the control: the base class version just fills it with background colour from the attribute.

virtual void wxGridCellEditor::Reset ( ) [pure virtual]

Reset the value in the control back to its starting value.

void wxGridCellEditor::SetControl (wxControl * *control*)

Set the wxControl that will be used by this cell editor for editing the value.

virtual void wxGridCellEditor::SetSize (const wxRect & *rect*) [virtual]

Size and position the edit control.

virtual void wxGridCellEditor::Show (bool *show,* wxGridCellAttr * *attr* = NULL) [virtual]

Show or hide the edit control, use the specified attributes to set colours/fonts for it.

virtual void wxGridCellEditor::StartingClick ( ) [virtual]

If the editor is enabled by clicking on the cell, this method will be called.

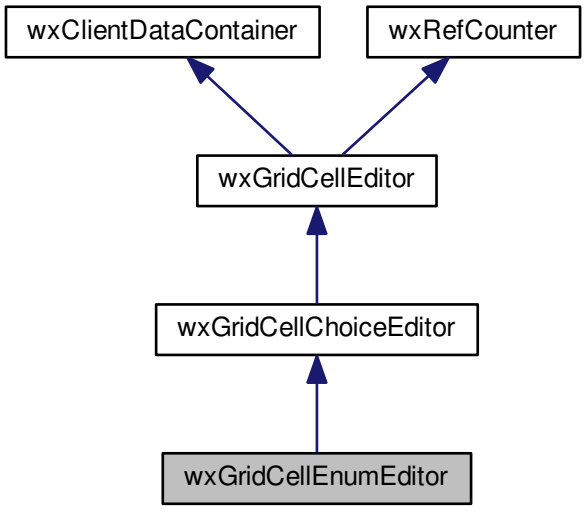
virtual void wxGridCellEditor::StartingKey (wxKeyEvent & *event*) [virtual]

If the editor is enabled by pressing keys on the grid, this will be called to let the editor do something about that first key if desired.

21.310 wxGridCellEnumEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellEnumEditor:



21.310.1 Detailed Description

Grid cell editor which displays an enum number as a textual equivalent (eg.

data in cell is 0,1,2 ... n the cell could be displayed as "John","Fred"..."Bob" in the combo choice box).

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellAutoWrapStringEditor, wxGridCellBoolEditor, wxGridCellChoiceEditor, wxGridCellTextEditor, wxGridCellFloatEditor, wxGridCellNumberEditor

Public Member Functions

- wxGridCellEnumEditor (const wxString &choices=wxEmptyString)

  Enum cell editor ctor.

Additional Inherited Members

21.310.2 Constructor & Destructor Documentation

wxGridCellEnumEditor::wxGridCellEnumEditor (const wxString & *choices* = wxEmptyString)

Enum cell editor ctor.

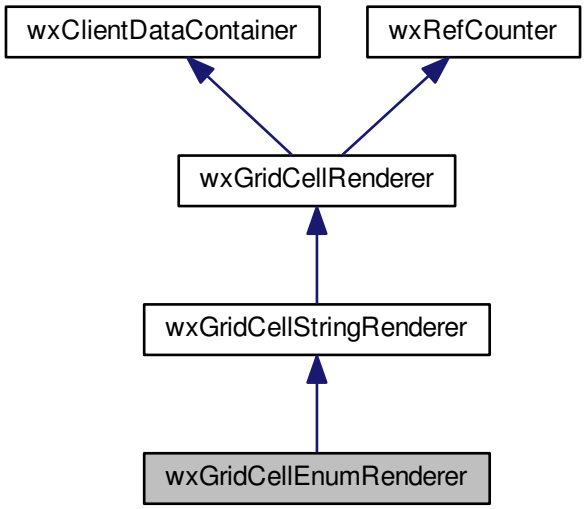
Parameters

choices	Comma separated choice parameters "item1[,item2[...,itemN]]".

21.311 wxGridCellEnumRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellEnumRenderer:



21.311.1 Detailed Description

This class may be used to render in a cell a number as a textual equivalent.

The corresponding text strings are specified as comma-separated items in the string passed to this renderer ctor or SetParameters() method. For example, if this string is `"John,Fred,Bob"` the cell will be rendered as "John", "Fred" or "Bob" if its contents is 0, 1 or 2 respectively.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellAutoWrapStringRenderer, wxGridCellBoolRenderer, wxGridCellDateTime↩ Renderer, wxGridCellFloatRenderer, wxGridCellNumberRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellEnumRenderer (const wxString &choices=wxEmptyString)

Enum renderer ctor.

- virtual void SetParameters (const wxString &params)

  Sets the comma separated string content of the enum.

Additional Inherited Members

21.311.2 Constructor & Destructor Documentation

wxGridCellEnumRenderer::wxGridCellEnumRenderer (const wxString & *choices* = wxEmptyString)

Enum renderer ctor.

Parameters

choices	Comma separated string parameters "item1[,item2[...,itemN]]".

21.311.3 Member Function Documentation

virtual void wxGridCellEnumRenderer::SetParameters (const wxString & *params*) `[virtual]`

Sets the comma separated string content of the enum.

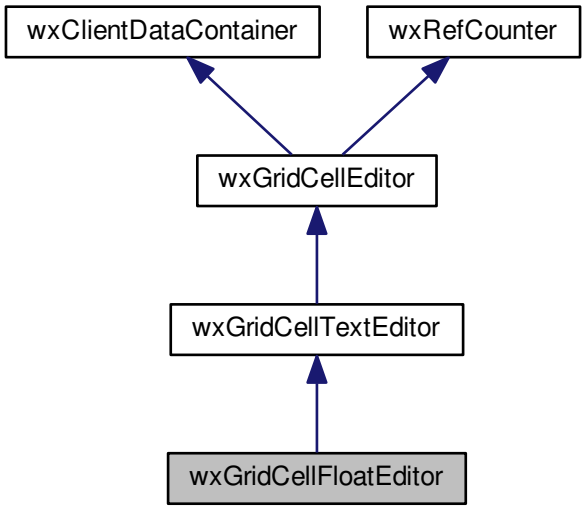
Parameters

params	Comma separated string parameters "item1[,item2[...,itemN]]".

21.312 wxGridCellFloatEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellFloatEditor:



21.312.1 Detailed Description

The editor for floating point numbers data.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellAutoWrapStringEditor, wxGridCellBoolEditor, wxGridCellChoiceEditor, wxGridCellEnumEditor, wxGridCellNumberEditor, wxGridCellTextEditor

Public Member Functions

- wxGridCellFloatEditor (int width=-1, int precision=-1, int format=wxGRID_FLOAT_FORMAT_DEFAULT)

  Float cell editor ctor.
- virtual void SetParameters (const wxString &params)

  The parameters string format is "width[,precision[,format]]" where `format` should be chosen between f|e|g|E|G (f is used by default)

Additional Inherited Members

21.312.2 Constructor & Destructor Documentation

wxGridCellFloatEditor::wxGridCellFloatEditor (int *width* = `-1`, int *precision* = `-1`, int *format* = wxGRID_FLOAT_FORMAT_DEFAULT)

Float cell editor ctor.

Parameters

width	Minimum number of characters to be shown.
precision	Number of digits after the decimal dot.
format	The format to use for displaying the number, a combination of wxGridCellFloatFormat enum elements. This parameter is only available since wxWidgets 2.9.3.

21.312.3 Member Function Documentation

virtual void wxGridCellFloatEditor::SetParameters (const wxString & *params*) `[virtual]`

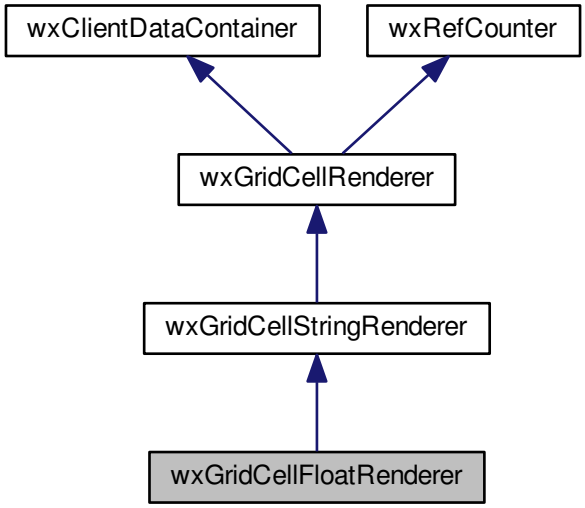
The parameters string format is "width[,precision[,format]]" where `format` should be chosen between f|e|g|E|G (f is used by default)

Reimplemented from wxGridCellTextEditor.

21.313 wxGridCellFloatRenderer Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridCellFloatRenderer:



21.313.1 Detailed Description

This class may be used to format floating point data in a cell.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellAutoWrapStringRenderer, wxGridCellBoolRenderer, wxGridCellDateTime↩ Renderer, wxGridCellEnumRenderer, wxGridCellNumberRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellFloatRenderer (int width=-1, int precision=-1, int format=wxGRID_FLOAT_FORMAT_DEFAULT)

  Float cell renderer ctor.
- int GetFormat () const

  Returns the specifier used to format the data to string.
- int GetPrecision () const

  Returns the precision.
- int GetWidth () const

  Returns the width.
- void SetFormat (int format)

  Set the format to use for display the number.
- virtual void SetParameters (const wxString &params)

The parameters string format is "width[,precision[,format]]" where `format` should be chosen between f|e|g|E|G (f is used by default)

- void SetPrecision (int precision)

  Sets the precision.

- void SetWidth (int width)

  Sets the width.

Additional Inherited Members

21.313.2 Constructor & Destructor Documentation

wxGridCellFloatRenderer::wxGridCellFloatRenderer (int *width* = −1, int *precision* = −1, int *format* = wxGRID_FLOAT_FORMAT_DEFAULT)

Float cell renderer ctor.

Parameters

width	Minimum number of characters to be shown.
precision	Number of digits after the decimal dot.
format	The format used to display the string, must be a combination of wxGridCellFloatFormat enum elements. This parameter is only available since wxWidgets 2.9.3.

21.313.3 Member Function Documentation

int wxGridCellFloatRenderer::GetFormat ( ) const

Returns the specifier used to format the data to string.

The returned value is a combination of wxGridCellFloatFormat elements.

Since

2.9.3

int wxGridCellFloatRenderer::GetPrecision ( ) const

Returns the precision.

int wxGridCellFloatRenderer::GetWidth ( ) const

Returns the width.

void wxGridCellFloatRenderer::SetFormat (int *format*)

Set the format to use for display the number.

Parameters

format	Must be a combination of wxGridCellFloatFormat enum elements.

Since

2.9.3

virtual void wxGridCellFloatRenderer::SetParameters (const wxString & *params*) `[virtual]`

The parameters string format is "width[,precision[,format]]" where `format` should be chosen between f|e|g|E|G (f is used by default)

void wxGridCellFloatRenderer::SetPrecision (int *precision*)

Sets the precision.

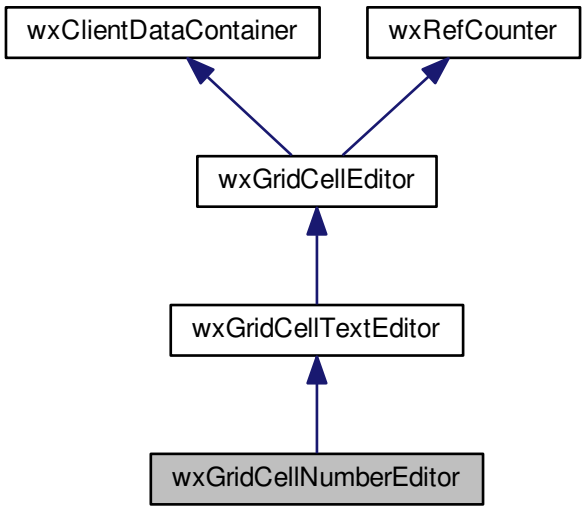
void wxGridCellFloatRenderer::SetWidth (int *width*)

Sets the width.

21.314 wxGridCellNumberEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellNumberEditor:



21.314.1 Detailed Description

Grid cell editor for numeric integer data.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellAutoWrapStringEditor, wxGridCellBoolEditor, wxGridCellChoiceEditor, wxGridCellEnumEditor, wxGridCellFloatEditor, wxGridCellTextEditor

Public Member Functions

- wxGridCellNumberEditor (int min=-1, int max=-1)

  Allows you to specify the range for acceptable data.
- virtual void SetParameters (const wxString &params)

  Parameters string format is "min,max".

Protected Member Functions

- bool HasRange () const

  If the return value is true, the editor uses a wxSpinCtrl to get user input, otherwise it uses a wxTextCtrl.
- wxString GetString () const

  String representation of the value.

21.314.2 Constructor & Destructor Documentation

wxGridCellNumberEditor::wxGridCellNumberEditor (int *min* = `-1`, int *max* = `-1`)

Allows you to specify the range for acceptable data.

Values equal to -1 for both *min* and *max* indicate that no range checking should be done.

21.314.3 Member Function Documentation

wxString wxGridCellNumberEditor::GetString ( ) const `[protected]`

String representation of the value.

bool wxGridCellNumberEditor::HasRange ( ) const `[protected]`

If the return value is true, the editor uses a wxSpinCtrl to get user input, otherwise it uses a wxTextCtrl.

virtual void wxGridCellNumberEditor::SetParameters (const wxString & *params*) `[virtual]`

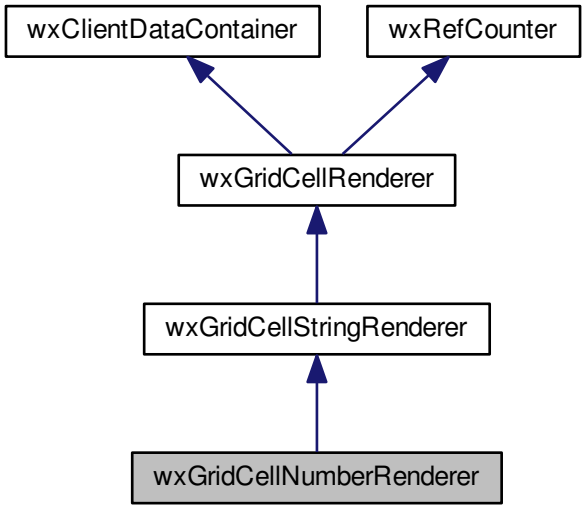
Parameters string format is "min,max".

Reimplemented from wxGridCellTextEditor.

21.315 wxGridCellNumberRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellNumberRenderer:



21.315.1 Detailed Description

This class may be used to format integer data in a cell.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellAutoWrapStringRenderer, wxGridCellBoolRenderer, wxGridCellDateTimeRenderer, wxGridCellEnumRenderer, wxGridCellFloatRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellNumberRenderer ()

  Default constructor.

Additional Inherited Members

21.315.2 Constructor & Destructor Documentation

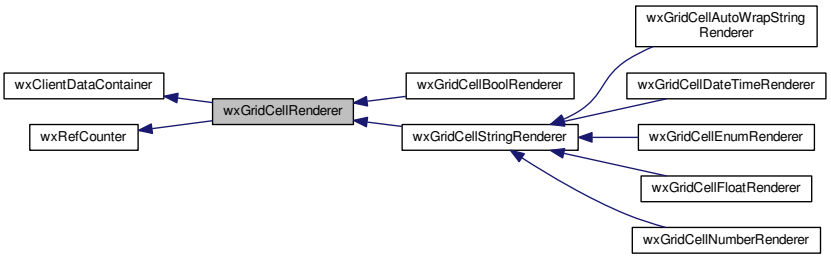
wxGridCellNumberRenderer::wxGridCellNumberRenderer ( )

Default constructor.

21.316 wxGridCellRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellRenderer:



21.316.1 Detailed Description

This class is responsible for actually drawing the cell in the grid.

You may pass it to the wxGridCellAttr (below) to change the format of one given cell or to wxGrid::SetDefaultRenderer() to change the view of all cells. This is an abstract class, and you will normally use one of the predefined derived classes or derive your own class from it.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellAutoWrapStringRenderer, wxGridCellBoolRenderer, wxGridCellDateTimeRenderer, wxGridCellEnumRenderer, wxGridCellFloatRenderer, wxGridCellNumberRenderer, wxGridCellStringRenderer

Public Member Functions

- wxGridCellRenderer ()
- virtual wxGridCellRenderer * Clone () const =0

  This function must be implemented in derived classes to return a copy of itself.
- virtual void Draw (wxGrid &grid, wxGridCellAttr &attr, wxDC &dc, const wxRect &rect, int row, int col, bool isSelected)=0

  Draw the given cell on the provided DC inside the given rectangle using the style specified by the attribute and the default or selected state corresponding to the isSelected value.
- virtual wxSize GetBestSize (wxGrid &grid, wxGridCellAttr &attr, wxDC &dc, int row, int col)=0

  Get the preferred size of the cell for its contents.
- virtual wxSize GetBestHeight (wxGrid &grid, wxGridCellAttr &attr, wxDC &dc, int row, int col, int width)

  Get the preferred height of the cell at the given width.
- virtual wxSize GetBestWidth (wxGrid &grid, wxGridCellAttr &attr, wxDC &dc, int row, int col, int height)

  Get the preferred width of the cell at the given height.

Protected Member Functions

- virtual ~wxGridCellRenderer ()

  The destructor is private because only DecRef() can delete us.

21.316.2 Constructor & Destructor Documentation

wxGridCellRenderer::wxGridCellRenderer ( )

virtual wxGridCellRenderer::~wxGridCellRenderer ( ) [protected], [virtual]

The destructor is private because only DecRef() can delete us.

21.316.3 Member Function Documentation

virtual wxGridCellRenderer* wxGridCellRenderer::Clone ( ) const [pure virtual]

This function must be implemented in derived classes to return a copy of itself.

virtual void wxGridCellRenderer::Draw (wxGrid & *grid*, wxGridCellAttr & *attr*, wxDC & *dc*, const wxRect & *rect*, int *row*, int *col*, bool *isSelected*) [pure virtual]

Draw the given cell on the provided DC inside the given rectangle using the style specified by the attribute and the default or selected state corresponding to the isSelected value.

This pure virtual function has a default implementation which will prepare the DC using the given attribute: it will draw the rectangle with the background colour from attr and set the text colour and font.

virtual wxSize wxGridCellRenderer::GetBestHeight (wxGrid & *grid*, wxGridCellAttr & *attr*, wxDC & *dc*, int *row*, int *col*, int *width*) [virtual]

Get the preferred height of the cell at the given width.

Some renderers may not have a well-defined best size, but only be able to provide the best height at the given width, e.g. this is the case of the standard wxGridCellAutoWrapStringRenderer. In this case, they should override this method, in addition to GetBestSize().

See also

GetBestWidth()

Since

3.1.0

virtual wxSize wxGridCellRenderer::GetBestSize (wxGrid & *grid*, wxGridCellAttr & *attr*, wxDC & *dc*, int *row*, int *col*) [pure virtual]

Get the preferred size of the cell for its contents.

This method must be overridden in the derived classes to return the minimal fitting size for displaying the content of the given grid cell.

See also

GetBestHeight(), GetBestWidth()

virtual wxSize wxGridCellRenderer::GetBestWidth (wxGrid & *grid,* wxGridCellAttr & *attr,* wxDC & *dc,* int *row,* int *col,* int *height*) `[virtual]`

Get the preferred width of the cell at the given height.

See GetBestHeight(), this method is symmetric to it.

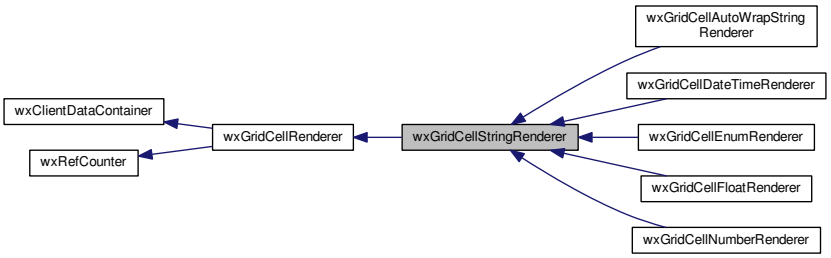
Since

3.1.0

21.317 wxGridCellStringRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellStringRenderer:



21.317.1 Detailed Description

This class may be used to format string data in a cell; it is the default for string cells.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellRenderer, wxGridCellAutoWrapStringRenderer, wxGridCellBoolRenderer, wxGridCellDateTime↩ Renderer, wxGridCellEnumRenderer, wxGridCellFloatRenderer, wxGridCellNumberRenderer

Public Member Functions

- wxGridCellStringRenderer ()

  Default constructor.

Additional Inherited Members

21.317.2 Constructor & Destructor Documentation

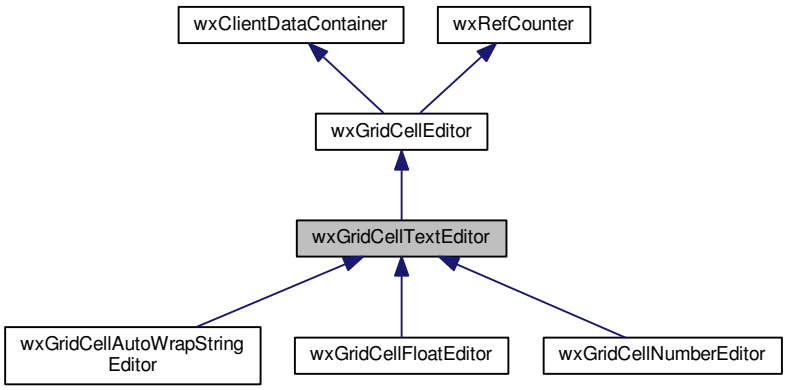
wxGridCellStringRenderer::wxGridCellStringRenderer ( )

Default constructor.

21.318 wxGridCellTextEditor Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCellTextEditor:



21.318.1 Detailed Description

Grid cell editor for string/text data.

Library: wxAdvanced

Category: Grid Related Classes

See also

wxGridCellEditor, wxGridCellAutoWrapStringEditor, wxGridCellBoolEditor, wxGridCellChoiceEditor, wxGridCellEnumEditor, wxGridCellFloatEditor, wxGridCellNumberEditor

Public Member Functions

- wxGridCellTextEditor (size_t maxChars=0)

  Text cell editor constructor.
- virtual void SetParameters (const wxString &params)

  The parameters string format is "n" where n is a number representing the maximum width.
- virtual void SetValidator (const wxValidator &validator)

  Set validator to validate user input.

Additional Inherited Members

21.318.2 Constructor & Destructor Documentation

wxGridCellTextEditor::wxGridCellTextEditor (size_t *maxChars* = 0) `[explicit]`

Text cell editor constructor.

Parameters

maxChars	Maximum width of text (this parameter is supported starting since wxWidgets 2.9.5).

21.318.3 Member Function Documentation

virtual void wxGridCellTextEditor::SetParameters (const wxString & *params*) `[virtual]`

The parameters string format is "n" where n is a number representing the maximum width.

Reimplemented in wxGridCellNumberEditor, and wxGridCellFloatEditor.

virtual void wxGridCellTextEditor::SetValidator (const wxValidator & *validator*) `[virtual]`

Set validator to validate user input.

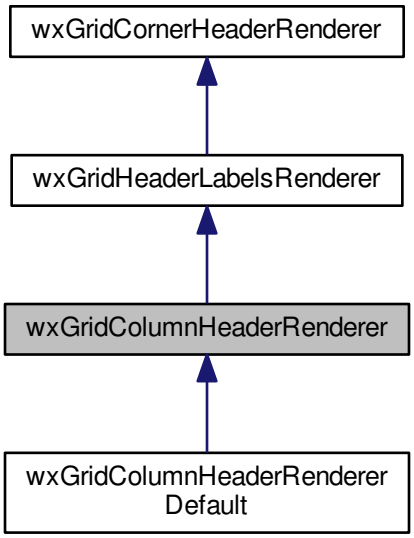
Since

2.9.5

21.319 wxGridColumnHeaderRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridColumnHeaderRenderer:



21.319.1 Detailed Description

Base class for column headers renderer.

This is the same as wxGridHeaderLabelsRenderer currently but we still use a separate class for it to distinguish it from wxGridRowHeaderRenderer.

See also

wxGridColumnHeaderRendererDefault
wxGridCellAttrProvider::GetColumnHeaderRenderer()

Since

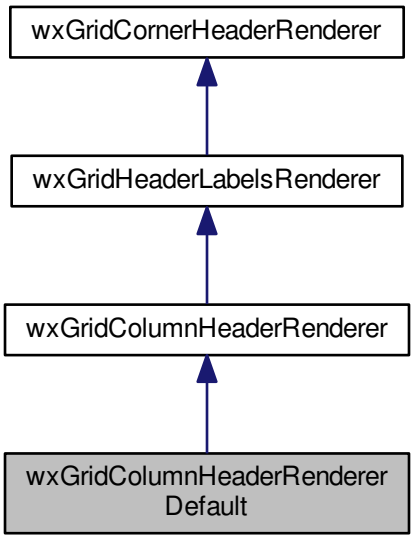
2.9.1

Additional Inherited Members

21.320 wxGridColumnHeaderRendererDefault Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridColumnHeaderRendererDefault:



21.320.1 Detailed Description

Default column header renderer.

See also

wxGridRowHeaderRendererDefault

Since

2.9.1

Public Member Functions

- virtual void DrawBorder (const wxGrid &grid, wxDC &dc, wxRect &rect) const

  Implement border drawing for the column labels.

21.320.2 Member Function Documentation

virtual void wxGridColumnHeaderRendererDefault::DrawBorder (const wxGrid & *grid,* wxDC & *dc,* wxRect & *rect*) const `[virtual]`

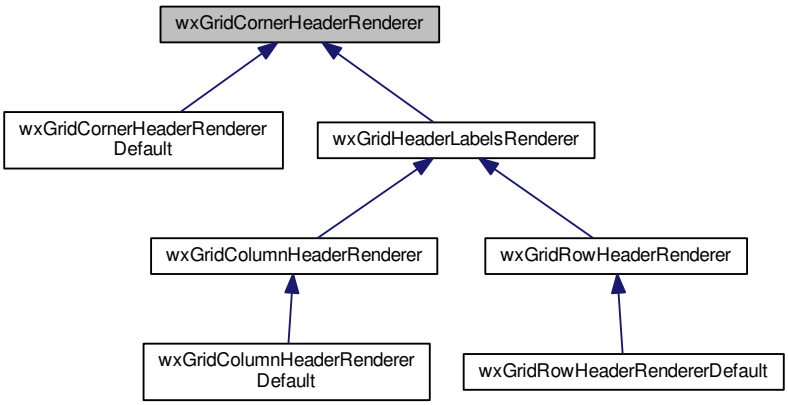
Implement border drawing for the column labels.

Implements wxGridCornerHeaderRenderer.

21.321 wxGridCornerHeaderRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCornerHeaderRenderer:



21.321.1 Detailed Description

Base class for corner window renderer.

This is the simplest of all header renderers and only has a single function.

See also

wxGridCellAttrProvider::GetCornerRenderer()

Since

2.9.1

Public Member Functions

- virtual void DrawBorder (const wxGrid &grid, wxDC &dc, wxRect &rect) const =0

  Called by the grid to draw the corner window border.

21.321.2 Member Function Documentation

virtual void wxGridCornerHeaderRenderer::DrawBorder (const wxGrid & *grid*, wxDC & *dc*, wxRect & *rect*) const `[pure virtual]`

Called by the grid to draw the corner window border.

This method is responsible for drawing the border inside the given *rect* and adjusting the rectangle size to correspond to the area inside the border, i.e. usually call wxRect::Deflate() to account for the border width.

Parameters

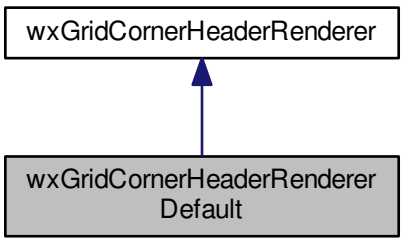
grid	The grid whose corner window is being drawn.
dc	The device context to use for drawing.
rect	Input/output parameter which contains the border rectangle on input and should be updated to contain the area inside the border on function return.

Implemented in wxGridCornerHeaderRendererDefault, wxGridColumnHeaderRendererDefault, and wxGridRowHeaderRendererDefault.

21.322 wxGridCornerHeaderRendererDefault Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridCornerHeaderRendererDefault:



21.322.1 Detailed Description

Default corner window renderer.

See also

wxGridColumnHeaderRendererDefault, wxGridRowHeaderRendererDefault

Since

2.9.1

Public Member Functions

- virtual void DrawBorder (const wxGrid &grid, wxDC &dc, wxRect &rect) const

  Implement border drawing for the corner window.

21.322.2 Member Function Documentation

virtual void wxGridCornerHeaderRendererDefault::DrawBorder (const wxGrid & *grid,* wxDC & *dc,* wxRect & *rect*) const `[virtual]`

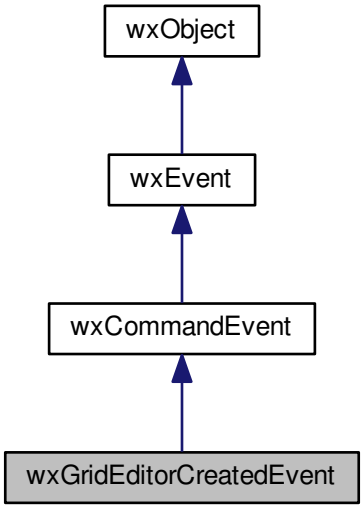
Implement border drawing for the corner window.

Implements wxGridCornerHeaderRenderer.

21.323 wxGridEditorCreatedEvent Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridEditorCreatedEvent:



21.323.1 Detailed Description

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxGridEditorCreatedEvent& event)

Event macros:

- EVT_GRID_EDITOR_CREATED(func): The editor for a cell was created. Processes a `wxEVT_GRID_E↩DITOR_CREATED` event type.
- EVT_GRID_CMD_EDITOR_CREATED(id, func): The editor for a cell was created; variant taking a window identifier. Processes a `wxEVT_GRID_EDITOR_CREATED` event type.

Library: wxAdvanced

Category: Grid Related Classes, Events

Public Member Functions

- wxGridEditorCreatedEvent ()

  Default constructor.
- wxGridEditorCreatedEvent (int id, wxEventType type, wxObject *obj, int row, int col, wxControl *ctrl)

  Constructor for initializing all event attributes.
- int GetCol ()

  Returns the column at which the event occurred.
- wxControl * GetControl ()

  Returns the edit control.
- int GetRow ()

  Returns the row at which the event occurred.
- void SetCol (int col)

  Sets the column at which the event occurred.
- void SetControl (wxControl *ctrl)

  Sets the edit control.
- void SetRow (int row)

  Sets the row at which the event occurred.

Additional Inherited Members

21.323.2 Constructor & Destructor Documentation

wxGridEditorCreatedEvent::wxGridEditorCreatedEvent ( )

Default constructor.

wxGridEditorCreatedEvent::wxGridEditorCreatedEvent (int *id*, wxEventType *type*, wxObject * *obj*, int *row*, int *col*, wxControl * *ctrl*)

Constructor for initializing all event attributes.

21.323.3 Member Function Documentation

int wxGridEditorCreatedEvent::GetCol ( )

Returns the column at which the event occurred.

wxControl* wxGridEditorCreatedEvent::GetControl ( )

Returns the edit control.

int wxGridEditorCreatedEvent::GetRow ( )

Returns the row at which the event occurred.

void wxGridEditorCreatedEvent::SetCol (int *col*)

Sets the column at which the event occurred.

void wxGridEditorCreatedEvent::SetControl (wxControl * *ctrl*)

Sets the edit control.

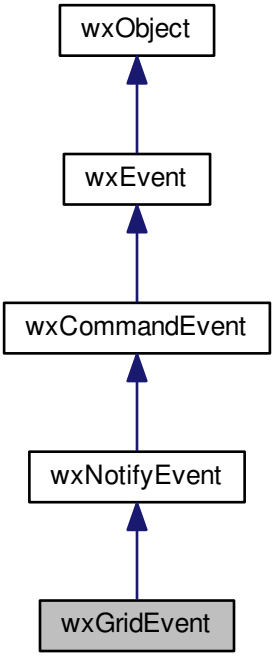
void wxGridEditorCreatedEvent::SetRow (int *row*)

Sets the row at which the event occurred.

21.324 wxGridEvent Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridEvent:



21.324.1 Detailed Description

This event class contains information about various grid events.

Notice that all grid event table macros are available in two versions: `EVT_GRID_XXX` and `EVT_GRID_CMD_X↩XX`. The only difference between the two is that the former doesn't allow to specify the grid window identifier and so takes a single parameter, the event handler, but is not suitable if there is more than one grid control in the window where the event table is used (as it would catch the events from all the grids). The version with `CMD` takes the id as

first argument and the event handler as the second one and so can be used with multiple grids as well. Otherwise there are no difference between the two and only the versions without the id are documented below for brevity.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxGridEvent& event)

Event macros:

- EVT_GRID_CELL_CHANGING(func): The user is about to change the data in a cell. The new cell value as string is available from GetString() event object method. This event can be vetoed if the change is not allowed. Processes a `wxEVT_GRID_CELL_CHANGING` event type.
- EVT_GRID_CELL_CHANGED(func): The user changed the data in a cell. The old cell value as string is available from GetString() event object method. Notice that vetoing this event still works for backwards compatibility reasons but any new code should only veto EVT_GRID_CELL_CHANGING event and not this one. Processes a `wxEVT_GRID_CELL_CHANGED` event type.
- EVT_GRID_CELL_LEFT_CLICK(func): The user clicked a cell with the left mouse button. Processes a `wxEVT_GRID_CELL_LEFT_CLICK` event type.
- EVT_GRID_CELL_LEFT_DCLICK(func): The user double-clicked a cell with the left mouse button. Processes a `wxEVT_GRID_CELL_LEFT_DCLICK` event type.
- EVT_GRID_CELL_RIGHT_CLICK(func): The user clicked a cell with the right mouse button. Processes a `wxEVT_GRID_CELL_RIGHT_CLICK` event type.
- EVT_GRID_CELL_RIGHT_DCLICK(func): The user double-clicked a cell with the right mouse button. Processes a `wxEVT_GRID_CELL_RIGHT_DCLICK` event type.
- EVT_GRID_EDITOR_HIDDEN(func): The editor for a cell was hidden. Processes a `wxEVT_GRID_EDITOR_HIDDEN` event type.
- EVT_GRID_EDITOR_SHOWN(func): The editor for a cell was shown. Processes a `wxEVT_GRID_EDITOR_SHOWN` event type.
- EVT_GRID_LABEL_LEFT_CLICK(func): The user clicked a label with the left mouse button. Processes a `wxEVT_GRID_LABEL_LEFT_CLICK` event type.
- EVT_GRID_LABEL_LEFT_DCLICK(func): The user double-clicked a label with the left mouse button. Processes a `wxEVT_GRID_LABEL_LEFT_DCLICK` event type.
- EVT_GRID_LABEL_RIGHT_CLICK(func): The user clicked a label with the right mouse button. Processes a `wxEVT_GRID_LABEL_RIGHT_CLICK` event type.
- EVT_GRID_LABEL_RIGHT_DCLICK(func): The user double-clicked a label with the right mouse button. Processes a `wxEVT_GRID_LABEL_RIGHT_DCLICK` event type.
- EVT_GRID_SELECT_CELL(func): The given cell was made current, either by user or by the program via a call to SetGridCursor() or GoToCell(). Processes a `wxEVT_GRID_SELECT_CELL` event type.
- EVT_GRID_COL_MOVE(func): The user tries to change the order of the columns in the grid by dragging the column specified by GetCol(). This event can be vetoed to either prevent the user from reordering the column change completely (but notice that if you don't want to allow it at all, you simply shouldn't call wxGrid::EnableDragColMove() in the first place), vetoed but handled in some way in the handler, e.g. by really moving the column to the new position at the associated table level, or allowed to proceed in which case wxGrid::SetColPos() is used to reorder the columns display order without affecting the use of the column indices otherwise. This event macro corresponds to `wxEVT_GRID_COL_MOVE` event type.

- EVT_GRID_COL_SORT(func): This event is generated when a column is clicked by the user and its name is explained by the fact that the custom reaction to a click on a column is to sort the grid contents by this column. However the grid itself has no special support for sorting and it's up to the handler of this event to update the associated table. But if the event is handled (and not vetoed) the grid supposes that the table was indeed resorted and updates the column to indicate the new sort order and refreshes itself. This event macro corresponds to `wxEVT_GRID_COL_SORT` event type.
- EVT_GRID_TABBING(func): This event is generated when the user presses TAB or Shift-TAB in the grid. It can be used to customize the simple default TAB handling logic, e.g. to go to the next non-empty cell instead of just the next cell. See also wxGrid::SetTabBehaviour(). This event is new since wxWidgets 2.9.5.

Library: wxAdvanced

Category: Grid Related Classes, Events

Public Member Functions

- wxGridEvent ()

  Default constructor.
- wxGridEvent (int id, wxEventType type, wxObject *obj, int row=-1, int col=-1, int x=-1, int y=-1, bool sel=true, const wxKeyboardState &kbd=wxKeyboardState())

  Constructor for initializing all event attributes.
- bool AltDown () const

  Returns true if the Alt key was down at the time of the event.
- bool ControlDown () const

  Returns true if the Control key was down at the time of the event.
- virtual int GetCol ()

  Column at which the event occurred.
- wxPoint GetPosition ()

  Position in pixels at which the event occurred.
- virtual int GetRow ()

  Row at which the event occurred.
- bool MetaDown () const

  Returns true if the Meta key was down at the time of the event.
- bool Selecting ()

  Returns true if the user is selecting grid cells, or false if deselecting.
- bool ShiftDown () const

  Returns true if the Shift key was down at the time of the event.

Additional Inherited Members

21.324.2 Constructor & Destructor Documentation

wxGridEvent::wxGridEvent ( )

Default constructor.

wxGridEvent::wxGridEvent (int *id*, wxEventType *type*, wxObject * *obj*, int *row* = -1, int *col* = -1, int *x* = -1, int *y* = -1, bool *sel* = true, const wxKeyboardState & *kbd* = wxKeyboardState())

Constructor for initializing all event attributes.

21.324.3 Member Function Documentation

bool wxGridEvent::AltDown ( ) const

Returns true if the Alt key was down at the time of the event.

bool wxGridEvent::ControlDown ( ) const

Returns true if the Control key was down at the time of the event.

virtual int wxGridEvent::GetCol ( ) [virtual]

Column at which the event occurred.

Notice that for a `wxEVT_GRID_SELECT_CELL` event this column is the column of the newly selected cell while the previously selected cell can be retrieved using wxGrid::GetGridCursorCol().

wxPoint wxGridEvent::GetPosition ( )

Position in pixels at which the event occurred.

virtual int wxGridEvent::GetRow ( ) [virtual]

Row at which the event occurred.

Notice that for a `wxEVT_GRID_SELECT_CELL` event this row is the row of the newly selected cell while the previously selected cell can be retrieved using wxGrid::GetGridCursorRow().

bool wxGridEvent::MetaDown ( ) const

Returns true if the Meta key was down at the time of the event.

bool wxGridEvent::Selecting ( )

Returns true if the user is selecting grid cells, or false if deselecting.

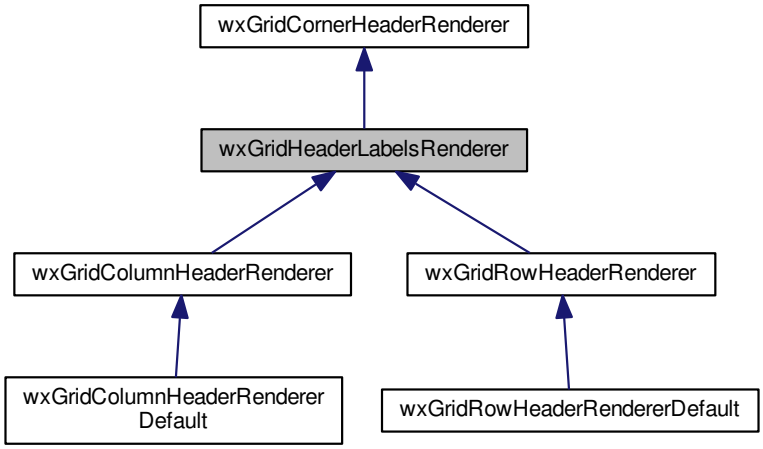
bool wxGridEvent::ShiftDown ( ) const

Returns true if the Shift key was down at the time of the event.

21.325 wxGridHeaderLabelsRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridHeaderLabelsRenderer:



21.325.1 Detailed Description

Common base class for row and column headers renderers.

See also

wxGridColumnHeaderRenderer, wxGridRowHeaderRenderer

Since

2.9.1

Public Member Functions

- virtual void DrawLabel (const wxGrid &grid, wxDC &dc, const wxString &value, const wxRect &rect, int horiz↩ Align, int vertAlign, int textOrientation) const

  Called by the grid to draw the specified label.

21.325.2 Member Function Documentation

virtual void wxGridHeaderLabelsRenderer::DrawLabel (const wxGrid & *grid,* wxDC & *dc,* const wxString & *value,* const wxRect & *rect,* int *horizAlign,* int *vertAlign,* int *textOrientation*) const `[virtual]`

Called by the grid to draw the specified label.

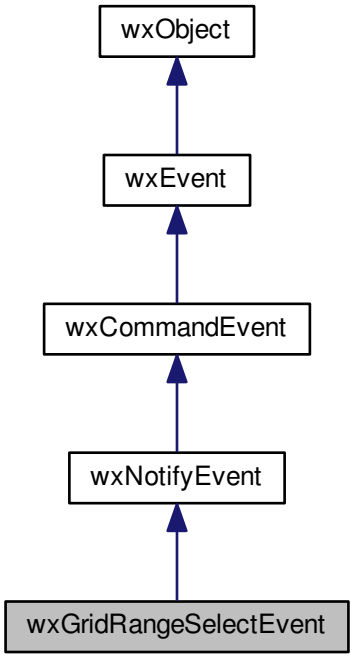
Notice that the base class DrawBorder() method is called before this one.

The default implementation uses wxGrid::GetLabelTextColour() and wxGrid::GetLabelFont() to draw the label.

21.326 wxGridRangeSelectEvent Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridRangeSelectEvent:



21.326.1 Detailed Description

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxGridRangeSelectEvent& event)

Event macros:

- EVT_GRID_RANGE_SELECT(func): The user selected a group of contiguous cells. Processes a `wxEVT_GRID_RANGE_SELECT` event type.
- EVT_GRID_CMD_RANGE_SELECT(id, func): The user selected a group of contiguous cells; variant taking a window identifier. Processes a `wxEVT_GRID_RANGE_SELECT` event type.

Library: wxAdvanced

Category: Grid Related Classes, Events

Public Member Functions

- wxGridRangeSelectEvent ()

  Default constructor.

- wxGridRangeSelectEvent (int id, wxEventType type, wxObject ∗obj, const wxGridCellCoords &topLeft, const wxGridCellCoords &bottomRight, bool sel=true, const wxKeyboardState &kbd=wxKeyboardState())

  Constructor for initializing all event attributes.
- bool AltDown () const

  Returns true if the Alt key was down at the time of the event.
- bool ControlDown () const

  Returns true if the Control key was down at the time of the event.
- wxGridCellCoords GetBottomRightCoords ()

  Top left corner of the rectangular area that was (de)selected.
- int GetBottomRow ()

  Bottom row of the rectangular area that was (de)selected.
- int GetLeftCol ()

  Left column of the rectangular area that was (de)selected.
- int GetRightCol ()

  Right column of the rectangular area that was (de)selected.
- wxGridCellCoords GetTopLeftCoords ()

  Top left corner of the rectangular area that was (de)selected.
- int GetTopRow ()

  Top row of the rectangular area that was (de)selected.
- bool MetaDown () const

  Returns true if the Meta key was down at the time of the event.
- bool Selecting ()

  Returns true if the area was selected, false otherwise.
- bool ShiftDown () const

  Returns true if the Shift key was down at the time of the event.

Additional Inherited Members

21.326.2 Constructor & Destructor Documentation

wxGridRangeSelectEvent::wxGridRangeSelectEvent ( )

Default constructor.

wxGridRangeSelectEvent::wxGridRangeSelectEvent (int *id*, wxEventType *type*, wxObject ∗ *obj*, const wxGridCellCoords & *topLeft*, const wxGridCellCoords & *bottomRight*, bool *sel* = `true`, const wxKeyboardState & *kbd* = wxKeyboardState `()`)

Constructor for initializing all event attributes.

21.326.3 Member Function Documentation

bool wxGridRangeSelectEvent::AltDown ( ) const

Returns true if the Alt key was down at the time of the event.

bool wxGridRangeSelectEvent::ControlDown ( ) const

Returns true if the Control key was down at the time of the event.

wxGridCellCoords wxGridRangeSelectEvent::GetBottomRightCoords ( )

Top left corner of the rectangular area that was (de)selected.

int wxGridRangeSelectEvent::GetBottomRow ( )

Bottom row of the rectangular area that was (de)selected.

int wxGridRangeSelectEvent::GetLeftCol ( )

Left column of the rectangular area that was (de)selected.

int wxGridRangeSelectEvent::GetRightCol ( )

Right column of the rectangular area that was (de)selected.

wxGridCellCoords wxGridRangeSelectEvent::GetTopLeftCoords ( )

Top left corner of the rectangular area that was (de)selected.

int wxGridRangeSelectEvent::GetTopRow ( )

Top row of the rectangular area that was (de)selected.

bool wxGridRangeSelectEvent::MetaDown ( ) const

Returns true if the Meta key was down at the time of the event.

bool wxGridRangeSelectEvent::Selecting ( )

Returns true if the area was selected, false otherwise.

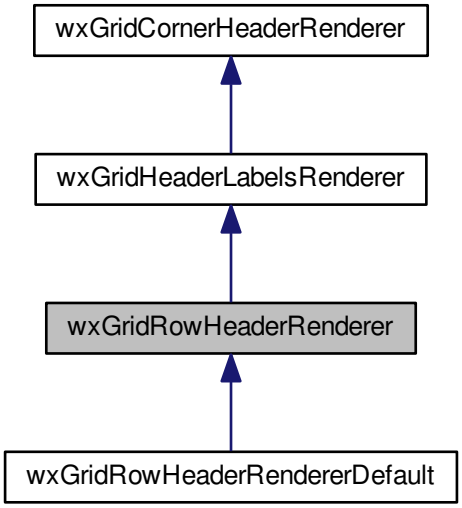
bool wxGridRangeSelectEvent::ShiftDown ( ) const

Returns true if the Shift key was down at the time of the event.

21.327 wxGridRowHeaderRenderer Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridRowHeaderRenderer:



21.327.1 Detailed Description

Base class for row headers renderer.

This is the same as wxGridHeaderLabelsRenderer currently but we still use a separate class for it to distinguish it from wxGridColumnHeaderRenderer.

See also

wxGridRowHeaderRendererDefault
wxGridCellAttrProvider::GetRowHeaderRenderer()

Since

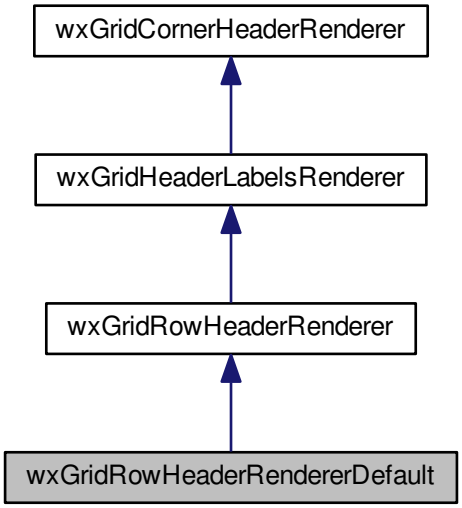
2.9.1

Additional Inherited Members

21.328 wxGridRowHeaderRendererDefault Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridRowHeaderRendererDefault:



21.328.1 Detailed Description

Default row header renderer.

You may derive from this class if you need to only override one of its methods (i.e. either DrawLabel() or DrawBorder()) but continue to use the default implementation for the other one.

See also

wxGridColumnHeaderRendererDefault

Since

2.9.1

Public Member Functions

- virtual void DrawBorder (const wxGrid &grid, wxDC &dc, wxRect &rect) const

  Implement border drawing for the row labels.

21.328.2 Member Function Documentation

virtual void wxGridRowHeaderRendererDefault::DrawBorder (const wxGrid & *grid,* wxDC & *dc,* wxRect & *rect*) const `[virtual]`

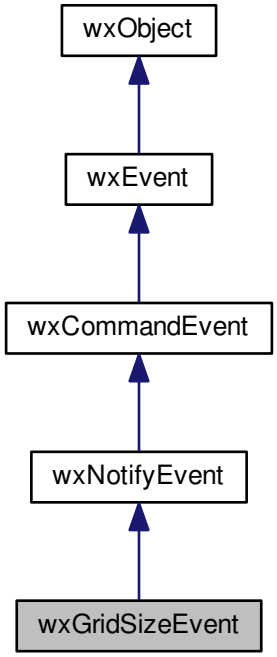
Implement border drawing for the row labels.

Implements wxGridCornerHeaderRenderer.

21.329 wxGridSizeEvent Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridSizeEvent:



21.329.1 Detailed Description

This event class contains information about a row/column resize event.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxGridSizeEvent& event)

Event macros:

- EVT_GRID_CMD_COL_SIZE(id, func): The user resized a column, corresponds to `wxEVT_GRID_COL_SIZE` event type.

- EVT_GRID_CMD_ROW_SIZE(id, func): The user resized a row, corresponds to `wxEVT_GRID_ROW_SIZE` event type.
- EVT_GRID_COL_SIZE(func): Same as EVT_GRID_CMD_COL_SIZE() but uses `wxID_ANY` id.
- EVT_GRID_COL_AUTO_SIZE(func): This event is sent when a column must be resized to its best size, e.g. when the user double clicks the column divider. The default implementation simply resizes the column to fit the column label (but not its contents as this could be too slow for big grids). This macro corresponds to `wxEVT_GRID_COL_AUTO_SIZE` event type and is new since wxWidgets 2.9.5.
- EVT_GRID_ROW_SIZE(func): Same as EVT_GRID_CMD_ROW_SIZE() but uses `wxID_ANY` id.

Library: wxAdvanced

Category: Grid Related Classes, Events

Public Member Functions

- wxGridSizeEvent ()

  Default constructor.
- wxGridSizeEvent (int id, wxEventType type, wxObject *obj, int rowOrCol=-1, int x=-1, int y=-1, const wxKeyboardState &kbd=wxKeyboardState())

  Constructor for initializing all event attributes.
- bool AltDown () const

  Returns true if the Alt key was down at the time of the event.
- bool ControlDown () const

  Returns true if the Control key was down at the time of the event.
- wxPoint GetPosition ()

  Position in pixels at which the event occurred.
- int GetRowOrCol ()

  Row or column at that was resized.
- bool MetaDown () const

  Returns true if the Meta key was down at the time of the event.
- bool ShiftDown () const

  Returns true if the Shift key was down at the time of the event.

Additional Inherited Members

21.329.2 Constructor & Destructor Documentation

wxGridSizeEvent::wxGridSizeEvent ( )

Default constructor.

wxGridSizeEvent::wxGridSizeEvent (int *id*, wxEventType *type*, wxObject * *obj*, int *rowOrCol* = −1, int *x* = −1, int *y* = −1, const wxKeyboardState & *kbd* = wxKeyboardState())

Constructor for initializing all event attributes.

21.329.3 Member Function Documentation

bool wxGridSizeEvent::AltDown ( ) const

Returns true if the Alt key was down at the time of the event.

bool wxGridSizeEvent::ControlDown ( ) const

Returns true if the Control key was down at the time of the event.

wxPoint wxGridSizeEvent::GetPosition ( )

Position in pixels at which the event occurred.

int wxGridSizeEvent::GetRowOrCol ( )

Row or column at that was resized.

bool wxGridSizeEvent::MetaDown ( ) const

Returns true if the Meta key was down at the time of the event.

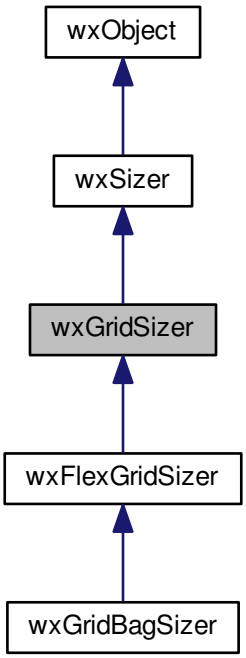
bool wxGridSizeEvent::ShiftDown ( ) const

Returns true if the Shift key was down at the time of the event.

21.330 wxGridSizer Class Reference

```
#include <wx/sizer.h>
```

Inheritance diagram for wxGridSizer:



21.330.1 Detailed Description

A grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields having the same size, i.e.

the width of each field is the width of the widest child, the height of each field is the height of the tallest child.

Library: wxCore

Category: Window Layout

See also

wxSizer, Sizers Overview

Public Member Functions

- int GetCols () const

  Returns the number of columns that has been specified for the sizer.
- int GetRows () const

  Returns the number of rows that has been specified for the sizer.
- int GetEffectiveColsCount () const

  Returns the number of columns currently used by the sizer.

- int GetEffectiveRowsCount () const

  Returns the number of rows currently used by the sizer.
- int GetHGap () const

  Returns the horizontal gap (in pixels) between cells in the sizer.
- int GetVGap () const

  Returns the vertical gap (in pixels) between the cells in the sizer.
- void SetCols (int cols)

  Sets the number of columns in the sizer.
- void SetHGap (int gap)

  Sets the horizontal gap (in pixels) between cells in the sizer.
- void SetRows (int rows)

  Sets the number of rows in the sizer.
- void SetVGap (int gap)

  Sets the vertical gap (in pixels) between the cells in the sizer.
- virtual wxSize CalcMin ()

  This method is abstract and has to be overwritten by any derived class.
- virtual void RecalcSizes ()

  This method is abstract and has to be overwritten by any derived class.

- wxGridSizer (int cols, int vgap, int hgap)

  wxGridSizer constructors.
- wxGridSizer (int cols, const wxSize &gap=wxSize(0, 0))

  wxGridSizer constructors.
- wxGridSizer (int rows, int cols, int vgap, int hgap)

  wxGridSizer constructors.
- wxGridSizer (int rows, int cols, const wxSize &gap)

  wxGridSizer constructors.

Additional Inherited Members

21.330.2 Constructor & Destructor Documentation

wxGridSizer::wxGridSizer (int *cols*, int *vgap*, int *hgap*)

wxGridSizer constructors.

Usually only the number of columns in the flex grid sizer needs to be specified using *cols* argument. The number of rows will be deduced automatically depending on the number of the elements added to the sizer.

If a constructor form with *rows* parameter is used (and the value of *rows* argument is not zero, meaning "unspecified") the sizer will check that no more than `cols*rows` elements are added to it, i.e. that no more than the given number of *rows* is used. Adding less than maximally allowed number of items is not an error however.

Finally, it is also possible to specify the number of rows and use 0 for *cols*. In this case, the sizer will use the given fixed number of rows and as many columns as necessary.

The *gap* (or *vgap* and *hgap*, which correspond to the height and width of the wxSize object) argument defines the size of the padding between the rows (its vertical component, or *vgap*) and columns (its horizontal component, or *hgap*), in pixels.

Since

2.9.1 (except for the four argument overload)

wxGridSizer::wxGridSizer (int *cols*, const wxSize & *gap* = wxSize(0, 0))

wxGridSizer constructors.

Usually only the number of columns in the flex grid sizer needs to be specified using *cols* argument. The number of rows will be deduced automatically depending on the number of the elements added to the sizer.

If a constructor form with *rows* parameter is used (and the value of *rows* argument is not zero, meaning "unspecified") the sizer will check that no more than `cols*rows` elements are added to it, i.e. that no more than the given number of *rows* is used. Adding less than maximally allowed number of items is not an error however.

Finally, it is also possible to specify the number of rows and use 0 for *cols*. In this case, the sizer will use the given fixed number of rows and as many columns as necessary.

The *gap* (or *vgap* and *hgap*, which correspond to the height and width of the wxSize object) argument defines the size of the padding between the rows (its vertical component, or *vgap*) and columns (its horizontal component, or *hgap*), in pixels.

Since

2.9.1 (except for the four argument overload)

wxGridSizer::wxGridSizer (int *rows*, int *cols*, int *vgap*, int *hgap*)

wxGridSizer constructors.

Usually only the number of columns in the flex grid sizer needs to be specified using *cols* argument. The number of rows will be deduced automatically depending on the number of the elements added to the sizer.

If a constructor form with *rows* parameter is used (and the value of *rows* argument is not zero, meaning "unspecified") the sizer will check that no more than `cols*rows` elements are added to it, i.e. that no more than the given number of *rows* is used. Adding less than maximally allowed number of items is not an error however.

Finally, it is also possible to specify the number of rows and use 0 for *cols*. In this case, the sizer will use the given fixed number of rows and as many columns as necessary.

The *gap* (or *vgap* and *hgap*, which correspond to the height and width of the wxSize object) argument defines the size of the padding between the rows (its vertical component, or *vgap*) and columns (its horizontal component, or *hgap*), in pixels.

Since

2.9.1 (except for the four argument overload)

wxGridSizer::wxGridSizer (int *rows*, int *cols*, const wxSize & *gap*)

wxGridSizer constructors.

Usually only the number of columns in the flex grid sizer needs to be specified using *cols* argument. The number of rows will be deduced automatically depending on the number of the elements added to the sizer.

If a constructor form with *rows* parameter is used (and the value of *rows* argument is not zero, meaning "unspecified") the sizer will check that no more than `cols*rows` elements are added to it, i.e. that no more than the given number of *rows* is used. Adding less than maximally allowed number of items is not an error however.

Finally, it is also possible to specify the number of rows and use 0 for *cols*. In this case, the sizer will use the given fixed number of rows and as many columns as necessary.

The *gap* (or *vgap* and *hgap*, which correspond to the height and width of the wxSize object) argument defines the size of the padding between the rows (its vertical component, or *vgap*) and columns (its horizontal component, or *hgap*), in pixels.

Since

2.9.1 (except for the four argument overload)

21.330.3 Member Function Documentation

virtual wxSize wxGridSizer::CalcMin ( ) [virtual]

This method is abstract and has to be overwritten by any derived class.

Here, the sizer will do the actual calculation of its children's minimal sizes.

Implements wxSizer.

Reimplemented in wxFlexGridSizer, and wxGridBagSizer.

int wxGridSizer::GetCols ( ) const

Returns the number of columns that has been specified for the sizer.

Returns zero if the sizer is automatically adjusting the number of columns depending on number of its children. To get the effective number of columns or rows being currently used, see GetEffectiveColsCount()

int wxGridSizer::GetEffectiveColsCount ( ) const

Returns the number of columns currently used by the sizer.

This will depend on the number of children the sizer has if the sizer is automatically adjusting the number of columns/rows.

Since

2.9.1

int wxGridSizer::GetEffectiveRowsCount ( ) const

Returns the number of rows currently used by the sizer.

This will depend on the number of children the sizer has if the sizer is automatically adjusting the number of columns/rows.

Since

2.9.1

int wxGridSizer::GetHGap ( ) const

Returns the horizontal gap (in pixels) between cells in the sizer.

int wxGridSizer::GetRows ( ) const

Returns the number of rows that has been specified for the sizer.

Returns zero if the sizer is automatically adjusting the number of rows depending on number of its children. To get the effective number of columns or rows being currently used, see GetEffectiveRowsCount().

int wxGridSizer::GetVGap ( ) const

Returns the vertical gap (in pixels) between the cells in the sizer.

virtual void wxGridSizer::RecalcSizes ( ) [virtual]

This method is abstract and has to be overwritten by any derived class.

Here, the sizer will do the actual calculation of its children's positions and sizes.

Implements wxSizer.

Reimplemented in wxFlexGridSizer, and wxGridBagSizer.

void wxGridSizer::SetCols (int *cols*)

Sets the number of columns in the sizer.

void wxGridSizer::SetHGap (int *gap*)

Sets the horizontal gap (in pixels) between cells in the sizer.

void wxGridSizer::SetRows (int *rows*)

Sets the number of rows in the sizer.

void wxGridSizer::SetVGap (int *gap*)

Sets the vertical gap (in pixels) between the cells in the sizer.

21.331 wxGridSizesInfo Class Reference

```
#include <wx/grid.h>
```

21.331.1 Detailed Description

wxGridSizesInfo stores information about sizes of all wxGrid rows or columns.

It assumes that most of the rows or columns (which are both called elements here as the difference between them doesn't matter at this class level) have the default size and so stores it separately. And it uses a wxHashMap to store the sizes of all elements which have the non-default size.

This structure is particularly useful for serializing the sizes of all wxGrid elements at once.

Library: wxAdvanced

Category: Grid Related Classes

Public Member Functions

- wxGridSizesInfo ()

Default constructor.

- wxGridSizesInfo (int defSize, const wxArrayInt &allSizes)

  Constructor.

- int GetSize (unsigned pos) const

  Get the element size.

Public Attributes

- int m_sizeDefault

  Default size.

- wxUnsignedToIntHashMap m_customSizes

  Map with element indices as keys and their sizes as values.

21.331.2 Constructor & Destructor Documentation

wxGridSizesInfo::wxGridSizesInfo ( )

Default constructor.

m_sizeDefault and m_customSizes must be initialized later.

wxGridSizesInfo::wxGridSizesInfo (int *defSize*, const wxArrayInt & *allSizes*)

Constructor.

This constructor is used by wxGrid::GetRowSizes() and GetColSizes() methods. User code will usually use the default constructor instead.

Parameters

defSize	The default element size.
allSizes	Array containing the sizes of *all* elements, including those which have the default size.

21.331.3 Member Function Documentation

int wxGridSizesInfo::GetSize (unsigned *pos*) const

Get the element size.

Parameters

pos	The index of the element.

Returns

The size for this element, using m_customSizes if *pos* is in it or m_sizeDefault otherwise.

21.331.4 Member Data Documentation

wxUnsignedToIntHashMap wxGridSizesInfo::m_customSizes

Map with element indices as keys and their sizes as values.

This map only contains the elements with non-default size.

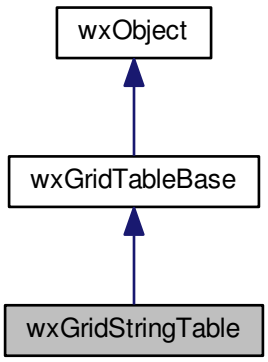
int wxGridSizesInfo::m_sizeDefault

Default size.

21.332 wxGridStringTable Class Reference

`#include <wx/grid.h>`

Inheritance diagram for wxGridStringTable:



21.332.1 Detailed Description

Simplest type of data table for a grid for small tables of strings that are stored in memory.

Public Member Functions

- wxGridStringTable ()
- wxGridStringTable (int numRows, int numCols)
- virtual int GetNumberRows ()

  Must be overridden to return the number of rows in the table.
- virtual int GetNumberCols ()

  Must be overridden to return the number of columns in the table.
- virtual wxString GetValue (int row, int col)

  Must be overridden to implement accessing the table values as text.
- virtual void SetValue (int row, int col, const wxString &value)

  Must be overridden to implement setting the table values as text.
- void Clear ()

  Clear the table contents.
- bool InsertRows (size_t pos=0, size_t numRows=1)

  Insert additional rows into the table.
- bool AppendRows (size_t numRows=1)

  Append additional rows at the end of the table.
- bool DeleteRows (size_t pos=0, size_t numRows=1)

Delete rows from the table.

- bool InsertCols (size_t pos=0, size_t numCols=1)

  Exactly the same as InsertRows() but for columns.
- bool AppendCols (size_t numCols=1)

  Exactly the same as AppendRows() but for columns.
- bool DeleteCols (size_t pos=0, size_t numCols=1)

  Exactly the same as DeleteRows() but for columns.
- void SetRowLabelValue (int row, const wxString &)

  Set the given label for the specified row.
- void SetColLabelValue (int col, const wxString &)

  Exactly the same as SetRowLabelValue() but for columns.
- wxString GetRowLabelValue (int row)

  Return the label of the specified row.
- wxString GetColLabelValue (int col)

  Return the label of the specified column.

Additional Inherited Members

21.332.2 Constructor & Destructor Documentation

wxGridStringTable::wxGridStringTable ( )

wxGridStringTable::wxGridStringTable (int *numRows*, int *numCols*)

21.332.3 Member Function Documentation

bool wxGridStringTable::AppendCols (size_t *numCols* = `1`) `[virtual]`

Exactly the same as AppendRows() but for columns.

Reimplemented from wxGridTableBase.

bool wxGridStringTable::AppendRows (size_t *numRows* = `1`) `[virtual]`

Append additional rows at the end of the table.

This method is provided in addition to InsertRows() as some data models may only support appending rows to them but not inserting them at arbitrary locations. In such case you may implement this method only and leave InsertRows() unimplemented.

Parameters

numRows	The number of rows to add.

Reimplemented from wxGridTableBase.

void wxGridStringTable::Clear ( ) `[virtual]`

Clear the table contents.

This method is used by wxGrid::ClearGrid().

Reimplemented from wxGridTableBase.

bool wxGridStringTable::DeleteCols (size_t *pos* = 0, size_t *numCols* = 1) [virtual]

Exactly the same as DeleteRows() but for columns.

Reimplemented from wxGridTableBase.

bool wxGridStringTable::DeleteRows (size_t *pos* = 0, size_t *numRows* = 1) [virtual]

Delete rows from the table.

Notice that currently deleting a row intersecting a multi-cell (see SetCellSize()) is not supported and will result in a crash.

Parameters

pos	The first row to delete.
numRows	The number of rows to delete.

Reimplemented from wxGridTableBase.

wxString wxGridStringTable::GetColLabelValue (int *col*) [virtual]

Return the label of the specified column.

Reimplemented from wxGridTableBase.

virtual int wxGridStringTable::GetNumberCols ( ) [virtual]

Must be overridden to return the number of columns in the table.

For backwards compatibility reasons, this method is not const. Use GetColsCount() instead of it in const methods of derived table classes,

Implements wxGridTableBase.

virtual int wxGridStringTable::GetNumberRows ( ) [virtual]

Must be overridden to return the number of rows in the table.

For backwards compatibility reasons, this method is not const. Use GetRowsCount() instead of it in const methods of derived table classes.

Implements wxGridTableBase.

wxString wxGridStringTable::GetRowLabelValue (int *row*) [virtual]

Return the label of the specified row.

Reimplemented from wxGridTableBase.

virtual wxString wxGridStringTable::GetValue (int *row*, int *col*) [virtual]

Must be overridden to implement accessing the table values as text.

Implements wxGridTableBase.

bool wxGridStringTable::InsertCols (size_t *pos* = 0, size_t *numCols* = 1) [virtual]

Exactly the same as InsertRows() but for columns.

Reimplemented from wxGridTableBase.

bool wxGridStringTable::InsertRows (size_t *pos* = 0, size_t *numRows* = 1) [virtual]

Insert additional rows into the table.

Parameters

pos	The position of the first new row.
numRows	The number of rows to insert.

Reimplemented from wxGridTableBase.

void wxGridStringTable::SetColLabelValue (int *col*, const wxString & *label*) [virtual]

Exactly the same as SetRowLabelValue() but for columns.

Reimplemented from wxGridTableBase.

void wxGridStringTable::SetRowLabelValue (int *row*, const wxString & *label*) [virtual]

Set the given label for the specified row.

The default version does nothing, i.e. the label is not stored. You must override this method in your derived class if you wish wxGrid::SetRowLabelValue() to work.

Reimplemented from wxGridTableBase.

virtual void wxGridStringTable::SetValue (int *row*, int *col*, const wxString & *value*) [virtual]

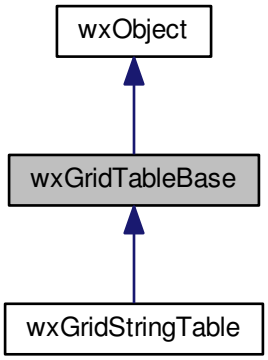
Must be overridden to implement setting the table values as text.

Implements wxGridTableBase.

21.333 wxGridTableBase Class Reference

```
#include <wx/grid.h>
```

Inheritance diagram for wxGridTableBase:



21.333.1 Detailed Description

The almost abstract base class for grid tables.

A grid table is responsible for storing the grid data and, indirectly, grid cell attributes. The data can be stored in the way most convenient for the application but has to be provided in string form to wxGrid. It is also possible to provide cells values in other formats if appropriate, e.g. as numbers.

This base class is not quite abstract as it implements a trivial strategy for storing the attributes by forwarding it to wxGridCellAttrProvider and also provides stubs for some other functions. However it does have a number of pure virtual methods which must be implemented in the derived classes.

See also

wxGridStringTable

Library: wxAdvanced

Category: Grid Related Classes

Public Member Functions

- wxGridTableBase ()

  Default constructor.
- virtual ~wxGridTableBase ()

  Destructor frees the attribute provider if it was created.
- virtual int GetNumberRows ()=0

  Must be overridden to return the number of rows in the table.
- virtual int GetNumberCols ()=0

  Must be overridden to return the number of columns in the table.
- int GetRowsCount () const

  Return the number of rows in the table.
- int GetColsCount () const

Return the number of columns in the table.

- virtual void SetView (wxGrid *grid)

  Called by the grid when the table is associated with it.

- virtual wxGrid * GetView () const

  Returns the last grid passed to SetView().

- virtual bool CanHaveAttributes ()

  Returns true if this table supports attributes or false otherwise.

Table Cell Accessors

- virtual bool IsEmptyCell (int row, int col)

  May be overridden to implement testing for empty cells.

- bool IsEmpty (const wxGridCellCoords &coords)

  Same as IsEmptyCell() but taking wxGridCellCoords.

- virtual wxString GetValue (int row, int col)=0

  Must be overridden to implement accessing the table values as text.

- virtual void SetValue (int row, int col, const wxString &value)=0

  Must be overridden to implement setting the table values as text.

- virtual wxString GetTypeName (int row, int col)

  Returns the type of the value in the given cell.

- virtual bool CanGetValueAs (int row, int col, const wxString &typeName)

  Returns true if the value of the given cell can be accessed as if it were of the specified type.

- virtual bool CanSetValueAs (int row, int col, const wxString &typeName)

  Returns true if the value of the given cell can be set as if it were of the specified type.

- virtual long GetValueAsLong (int row, int col)

  Returns the value of the given cell as a long.

- virtual double GetValueAsDouble (int row, int col)

  Returns the value of the given cell as a double.

- virtual bool GetValueAsBool (int row, int col)

  Returns the value of the given cell as a boolean.

- virtual void * GetValueAsCustom (int row, int col, const wxString &typeName)

  Returns the value of the given cell as a user-defined type.

- virtual void SetValueAsLong (int row, int col, long value)

  Sets the value of the given cell as a long.

- virtual void SetValueAsDouble (int row, int col, double value)

  Sets the value of the given cell as a double.

- virtual void SetValueAsBool (int row, int col, bool value)

  Sets the value of the given cell as a boolean.

- virtual void SetValueAsCustom (int row, int col, const wxString &typeName, void *value)

  Sets the value of the given cell as a user-defined type.

Table Structure Modifiers

Notice that none of these functions are pure virtual as they don't have to be implemented if the table structure is never modified after creation, i.e.

neither rows nor columns are never added or deleted but that you do need to implement them if they are called, i.e. if your code either calls them directly or uses the matching wxGrid methods, as by default they simply do nothing which is definitely inappropriate.

- virtual void Clear ()

  Clear the table contents.

- virtual bool InsertRows (size_t pos=0, size_t numRows=1)

  Insert additional rows into the table.

- virtual bool AppendRows (size_t numRows=1)

  Append additional rows at the end of the table.

- virtual bool DeleteRows (size_t pos=0, size_t numRows=1)

  Delete rows from the table.

- virtual bool InsertCols (size_t pos=0, size_t numCols=1)

Exactly the same as InsertRows() but for columns.

- virtual bool AppendCols (size_t numCols=1)
  Exactly the same as AppendRows() but for columns.
- virtual bool DeleteCols (size_t pos=0, size_t numCols=1)
  Exactly the same as DeleteRows() but for columns.

Table Row and Column Labels

By default the numbers are used for labeling rows and Latin letters for labeling columns.

If the table has more than 26 columns, the pairs of letters are used starting from the 27-th one and so on, i.e. the sequence of labels is A, B, ..., Z, AA, AB, ..., AZ, BA, ..., ..., ZZ, AAA, ...

- virtual wxString GetRowLabelValue (int row)
  Return the label of the specified row.
- virtual wxString GetColLabelValue (int col)
  Return the label of the specified column.
- virtual void SetRowLabelValue (int row, const wxString &label)
  Set the given label for the specified row.
- virtual void SetColLabelValue (int col, const wxString &label)
  Exactly the same as SetRowLabelValue() but for columns.

Attributes Management

By default the attributes management is delegated to wxGridCellAttrProvider class.

You may override the methods in this section to handle the attributes directly if, for example, they can be computed from the cell values.

- void SetAttrProvider (wxGridCellAttrProvider *attrProvider)
  Associate this attributes provider with the table.
- wxGridCellAttrProvider * GetAttrProvider () const
  Returns the attribute provider currently being used.
- virtual wxGridCellAttr * GetAttr (int row, int col, wxGridCellAttr::wxAttrKind kind)
  Return the attribute for the given cell.
- virtual void SetAttr (wxGridCellAttr *attr, int row, int col)
  Set attribute of the specified cell.
- virtual void SetRowAttr (wxGridCellAttr *attr, int row)
  Set attribute of the specified row.
- virtual void SetColAttr (wxGridCellAttr *attr, int col)
  Set attribute of the specified column.

Additional Inherited Members

21.333.2 Constructor & Destructor Documentation

wxGridTableBase::wxGridTableBase ( )

Default constructor.

virtual wxGridTableBase::~wxGridTableBase ( ) [virtual]

Destructor frees the attribute provider if it was created.

21.333.3 Member Function Documentation

virtual bool wxGridTableBase::AppendCols (size_t *numCols* = 1) [virtual]

Exactly the same as AppendRows() but for columns.

Reimplemented in wxGridStringTable.

virtual bool wxGridTableBase::AppendRows (size_t *numRows* = 1) [virtual]

Append additional rows at the end of the table.

This method is provided in addition to InsertRows() as some data models may only support appending rows to them but not inserting them at arbitrary locations. In such case you may implement this method only and leave InsertRows() unimplemented.

Parameters

numRows	The number of rows to add.

Reimplemented in wxGridStringTable.

virtual bool wxGridTableBase::CanGetValueAs (int *row*, int *col*, const wxString & *typeName*) [virtual]

Returns true if the value of the given cell can be accessed as if it were of the specified type.

By default the cells can only be accessed as strings. Note that a cell could be accessible in different ways, e.g. a numeric cell may return true for `wxGRID_VALUE_NUMBER` but also for `wxGRID_VALUE_STRING` indicating that the value can be coerced to a string form.

virtual bool wxGridTableBase::CanHaveAttributes ( ) [virtual]

Returns true if this table supports attributes or false otherwise.

By default, the table automatically creates a wxGridCellAttrProvider when this function is called if it had no attribute provider before and returns true.

virtual bool wxGridTableBase::CanSetValueAs (int *row*, int *col*, const wxString & *typeName*) [virtual]

Returns true if the value of the given cell can be set as if it were of the specified type.

See also

CanGetValueAs()

virtual void wxGridTableBase::Clear ( ) [virtual]

Clear the table contents.

This method is used by wxGrid::ClearGrid().

Reimplemented in wxGridStringTable.

virtual bool wxGridTableBase::DeleteCols (size_t *pos* = 0, size_t *numCols* = 1) [virtual]

Exactly the same as DeleteRows() but for columns.

Reimplemented in wxGridStringTable.

virtual bool wxGridTableBase::DeleteRows (size_t *pos* = 0, size_t *numRows* = 1) [virtual]

Delete rows from the table.

Notice that currently deleting a row intersecting a multi-cell (see SetCellSize()) is not supported and will result in a crash.

Parameters

pos	The first row to delete.
numRows	The number of rows to delete.

Reimplemented in wxGridStringTable.

virtual wxGridCellAttr* wxGridTableBase::GetAttr (int *row*, int *col*, wxGridCellAttr::wxAttrKind *kind*) [virtual]

Return the attribute for the given cell.

By default this function is simply forwarded to wxGridCellAttrProvider::GetAttr() but it may be overridden to handle attributes directly in the table.

wxGridCellAttrProvider* wxGridTableBase::GetAttrProvider ( ) const

Returns the attribute provider currently being used.

This function may return NULL if the attribute provider hasn't been neither associated with this table by SetAttrProvider() nor created on demand by any other methods.

virtual wxString wxGridTableBase::GetColLabelValue (int *col*) [virtual]

Return the label of the specified column.

Reimplemented in wxGridStringTable.

int wxGridTableBase::GetColsCount ( ) const

Return the number of columns in the table.

This method is not virtual and is only provided as a convenience for the derived classes which can't call GetNumberCols() without a `const_cast` from their const methods.

virtual int wxGridTableBase::GetNumberCols ( ) [pure virtual]

Must be overridden to return the number of columns in the table.

For backwards compatibility reasons, this method is not const. Use GetColsCount() instead of it in const methods of derived table classes,

Implemented in wxGridStringTable.

virtual int wxGridTableBase::GetNumberRows ( ) [pure virtual]

Must be overridden to return the number of rows in the table.

For backwards compatibility reasons, this method is not const. Use GetRowsCount() instead of it in const methods of derived table classes.

Implemented in wxGridStringTable.

virtual wxString wxGridTableBase::GetRowLabelValue (int *row*) [virtual]

Return the label of the specified row.

Reimplemented in wxGridStringTable.

int wxGridTableBase::GetRowsCount ( ) const

Return the number of rows in the table.

This method is not virtual and is only provided as a convenience for the derived classes which can't call GetNumberRows() without a `const_cast` from their const methods.

virtual wxString wxGridTableBase::GetTypeName (int *row*, int *col*) [virtual]

Returns the type of the value in the given cell.

By default all cells are strings and this method returns `wxGRID_VALUE_STRING`.

virtual wxString wxGridTableBase::GetValue (int *row*, int *col*) [pure virtual]

Must be overridden to implement accessing the table values as text.

Implemented in wxGridStringTable.

virtual bool wxGridTableBase::GetValueAsBool (int *row*, int *col*) [virtual]

Returns the value of the given cell as a boolean.

This should only be called if CanGetValueAs() returns true when called with `wxGRID_VALUE_BOOL` argument. Default implementation always return false.

virtual void* wxGridTableBase::GetValueAsCustom (int *row*, int *col*, const wxString & *typeName*) [virtual]

Returns the value of the given cell as a user-defined type.

This should only be called if CanGetValueAs() returns true when called with *typeName*. Default implementation always return NULL.

virtual double wxGridTableBase::GetValueAsDouble (int *row*, int *col*) [virtual]

Returns the value of the given cell as a double.

This should only be called if CanGetValueAs() returns true when called with `wxGRID_VALUE_FLOAT` argument. Default implementation always return 0.0.

virtual long wxGridTableBase::GetValueAsLong (int *row*, int *col*) [virtual]

Returns the value of the given cell as a long.

This should only be called if CanGetValueAs() returns true when called with `wxGRID_VALUE_NUMBER` argument. Default implementation always return 0.

virtual wxGrid* wxGridTableBase::GetView ( ) const [virtual]

Returns the last grid passed to SetView().

virtual bool wxGridTableBase::InsertCols (size_t *pos* = 0, size_t *numCols* = 1) [virtual]

Exactly the same as InsertRows() but for columns.

Reimplemented in wxGridStringTable.

virtual bool wxGridTableBase::InsertRows (size_t *pos* = 0, size_t *numRows* = 1) [virtual]

Insert additional rows into the table.

Parameters

pos	The position of the first new row.
numRows	The number of rows to insert.

Reimplemented in wxGridStringTable.

bool wxGridTableBase::IsEmpty (const wxGridCellCoords & *coords*)

Same as IsEmptyCell() but taking wxGridCellCoords.

Notice that this method is not virtual, only IsEmptyCell() should be overridden.

virtual bool wxGridTableBase::IsEmptyCell (int *row*, int *col*) [virtual]

May be overridden to implement testing for empty cells.

This method is used by the grid to test if the given cell is not used and so whether a neighbouring cell may overflow into it. By default it only returns true if the value of the given cell, as returned by GetValue(), is empty.

virtual void wxGridTableBase::SetAttr (wxGridCellAttr * *attr*, int *row*, int *col*) [virtual]

Set attribute of the specified cell.

By default this function is simply forwarded to wxGridCellAttrProvider::SetAttr().

The table takes ownership of *attr*, i.e. will call DecRef() on it.

void wxGridTableBase::SetAttrProvider (wxGridCellAttrProvider * *attrProvider*)

Associate this attributes provider with the table.

The table takes ownership of *attrProvider* pointer and will delete it when it doesn't need it any more. The pointer can be NULL, however this won't disable attributes management in the table but will just result in a default attributes being recreated the next time any of the other functions in this section is called. To completely disable the attributes support, should this be needed, you need to override CanHaveAttributes() to return false.

virtual void wxGridTableBase::SetColAttr (wxGridCellAttr * *attr*, int *col*) [virtual]

Set attribute of the specified column.

By default this function is simply forwarded to wxGridCellAttrProvider::SetColAttr().

The table takes ownership of *attr*, i.e. will call DecRef() on it.

virtual void wxGridTableBase::SetColLabelValue (int *col*, const wxString & *label*) [virtual]

Exactly the same as SetRowLabelValue() but for columns.

Reimplemented in wxGridStringTable.

virtual void wxGridTableBase::SetRowAttr (wxGridCellAttr * *attr*, int *row*) [virtual]

Set attribute of the specified row.

By default this function is simply forwarded to wxGridCellAttrProvider::SetRowAttr().

The table takes ownership of *attr*, i.e. will call DecRef() on it.

virtual void wxGridTableBase::SetRowLabelValue (int *row,* const wxString & *label*) `[virtual]`

Set the given label for the specified row.

The default version does nothing, i.e. the label is not stored. You must override this method in your derived class if you wish wxGrid::SetRowLabelValue() to work.

Reimplemented in wxGridStringTable.

virtual void wxGridTableBase::SetValue (int *row,* int *col,* const wxString & *value*) `[pure virtual]`

Must be overridden to implement setting the table values as text.

Implemented in wxGridStringTable.

virtual void wxGridTableBase::SetValueAsBool (int *row,* int *col,* bool *value*) `[virtual]`

Sets the value of the given cell as a boolean.

This should only be called if CanSetValueAs() returns true when called with `wxGRID_VALUE_BOOL` argument. Default implementation doesn't do anything.

virtual void wxGridTableBase::SetValueAsCustom (int *row,* int *col,* const wxString & *typeName,* void * *value*) `[virtual]`

Sets the value of the given cell as a user-defined type.

This should only be called if CanSetValueAs() returns true when called with *typeName*. Default implementation doesn't do anything.

virtual void wxGridTableBase::SetValueAsDouble (int *row,* int *col,* double *value*) `[virtual]`

Sets the value of the given cell as a double.

This should only be called if CanSetValueAs() returns true when called with `wxGRID_VALUE_FLOAT` argument. Default implementation doesn't do anything.

virtual void wxGridTableBase::SetValueAsLong (int *row,* int *col,* long *value*) `[virtual]`

Sets the value of the given cell as a long.

This should only be called if CanSetValueAs() returns true when called with `wxGRID_VALUE_NUMBER` argument. Default implementation doesn't do anything.

virtual void wxGridTableBase::SetView (wxGrid * *grid*) `[virtual]`

Called by the grid when the table is associated with it.

The default implementation stores the pointer and returns it from its GetView() and so only makes sense if the table cannot be associated with more than one grid at a time.

21.334 wxGridTableMessage Class Reference

`#include <wx/grid.h>`

21.334.1 Detailed Description

A simple class used to pass messages from the table to the grid.

Library: wxAdvanced

Category: Grid Related Classes

Public Member Functions

- wxGridTableMessage ()
- wxGridTableMessage (wxGridTableBase *table, int id, int comInt1=-1, int comInt2=-1)
- void SetTableObject (wxGridTableBase *table)
- wxGridTableBase * GetTableObject () const
- void SetId (int id)
- int GetId ()
- void SetCommandInt (int comInt1)
- int GetCommandInt ()
- void SetCommandInt2 (int comInt2)
- int GetCommandInt2 ()

21.334.2 Constructor & Destructor Documentation

wxGridTableMessage::wxGridTableMessage ( )

wxGridTableMessage::wxGridTableMessage (wxGridTableBase * *table,* int *id,* int *comInt1* = -1, int *comInt2* = -1)

21.334.3 Member Function Documentation

int wxGridTableMessage::GetCommandInt ( )

int wxGridTableMessage::GetCommandInt2 ( )

int wxGridTableMessage::GetId ( )

wxGridTableBase* wxGridTableMessage::GetTableObject ( ) const

void wxGridTableMessage::SetCommandInt (int *comInt1*)

void wxGridTableMessage::SetCommandInt2 (int *comInt2*)

void wxGridTableMessage::SetId (int *id*)

void wxGridTableMessage::SetTableObject (wxGridTableBase * *table*)

21.335 wxGridUpdateLocker Class Reference

```
#include <wx/grid.h>
```

21.335.1 Detailed Description

This small class can be used to prevent wxGrid from redrawing during its lifetime by calling wxGrid::BeginBatch() in its constructor and wxGrid::EndBatch() in its destructor.

It is typically used in a function performing several operations with a grid which would otherwise result in flicker. For example:

```
void MyFrame::Foo()
{
    m_grid = new wxGrid(this, ...);

    wxGridUpdateLocker noUpdates(m_grid);
    m_grid-AppendColumn();
    // ... many other operations with m_grid ...
    m_grid-AppendRow();

    // destructor called, grid refreshed
}
```

Using this class is easier and safer than calling wxGrid::BeginBatch() and wxGrid::EndBatch() because you don't risk missing the call the latter (due to an exception for example).

Library: wxAdvanced

Category: Grid Related Classes

Public Member Functions

- wxGridUpdateLocker (wxGrid *grid=NULL)

  Creates an object preventing the updates of the specified grid.
- ~wxGridUpdateLocker ()

  Destructor reenables updates for the grid this object is associated with.
- void Create (wxGrid *grid)

  This method can be called if the object had been constructed using the default constructor.

21.335.2 Constructor & Destructor Documentation

wxGridUpdateLocker::wxGridUpdateLocker (wxGrid * *grid* = NULL)

Creates an object preventing the updates of the specified *grid*.

The parameter could be NULL in which case nothing is done. If *grid* is non-NULL then the grid must exist for longer than this wxGridUpdateLocker object itself.

The default constructor could be followed by a call to Create() to set the grid object later.

wxGridUpdateLocker::~wxGridUpdateLocker ( )

Destructor reenables updates for the grid this object is associated with.

21.335.3 Member Function Documentation

void wxGridUpdateLocker::Create (wxGrid * *grid*)

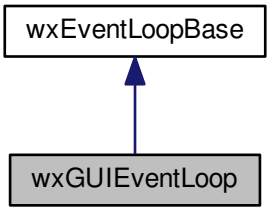
This method can be called if the object had been constructed using the default constructor.

It must not be called more than once.

21.336 wxGUIEventLoop Class Reference

`#include <wx/evtloop.h>`

Inheritance diagram for wxGUIEventLoop:



21.336.1 Detailed Description

A generic implementation of the GUI event loop.

Library: wxBase

Category: Application and Process Management

Public Member Functions

- wxGUIEventLoop ()
- virtual ~wxGUIEventLoop ()

Additional Inherited Members

21.336.2 Constructor & Destructor Documentation

wxGUIEventLoop::wxGUIEventLoop ( )

virtual wxGUIEventLoop::~wxGUIEventLoop ( ) [virtual]

21.337 wxHashMap Class Reference

`#include <wx/hashmap.h>`

21.337.1 Detailed Description

This is a simple, type-safe, and reasonably efficient hash map class, whose interface is a subset of the interface of STL containers.

In particular, the interface is modelled after std::map, and the various, non-standard, std::hash_map (http://www.cppreference.com/wiki/stl/map/start).

Example:

```
class MyClass { ... };

// declare a hash map with string keys and int values
WX_DECLARE_STRING_HASH_MAP( int, MyHash5 );
// same, with int keys and MyClass* values
WX_DECLARE_HASH_MAP( int, MyClass*, wxIntegerHash, wxIntegerEqual, MyHash1 );
// same, with wxString keys and int values
WX_DECLARE_STRING_HASH_MAP( int, MyHash3 );
// same, with wxString keys and values
WX_DECLARE_STRING_HASH_MAP( wxString, MyHash2 );

MyHash1 h1;
MyHash2 h2;

// store and retrieve values
h1[1] = new MyClass( 1 );
h1[10000000] = NULL;
h1[50000] = new MyClass( 2 );
h2["Bill"] = "ABC";
wxString tmp = h2["Bill"];
// since element with key "Joe" is not present, this will return
// the default value, which is an empty string in the case of wxString
MyClass tmp2 = h2["Joe"];

// iterate over all the elements in the class
MyHash2::iterator it;
for( it = h2.begin(); it != h2.end(); ++it )
{
    wxString key = it->first, value = it->second;
    // do something useful with key and value
}
```

21.337.2 Declaring new hash table types

```
WX_DECLARE_STRING_HASH_MAP( VALUE_T,     // type of the values
                            CLASSNAME ); // name of the class
```

Declares a hash map class named CLASSNAME, with wxString keys and VALUE_T values.

```
WX_DECLARE_VOIDPTR_HASH_MAP( VALUE_T,     // type of the values
                             CLASSNAME ); // name of the class
```

Declares a hash map class named CLASSNAME, with void* keys and VALUE_T values.

```
WX_DECLARE_HASH_MAP( KEY_T,      // type of the keys
                     VALUE_T,    // type of the values
                     HASH_T,     // hasher
                     KEY_EQ_T,   // key equality predicate
                     CLASSNAME); // name of the class
```

The HASH_T and KEY_EQ_T are the types used for the hashing function and key comparison. wxWidgets provides three predefined hashing functions: `wxIntegerHash` for integer types (int, long, short, and their unsigned counterparts), `wxStringHash` for strings (wxString, wxChar*, char*), and `wxPointerHash` for any kind of pointer. Similarly three equality predicates: `wxIntegerEqual`, `wxStringEqual`, `wxPointerEqual` are provided. Using this you could declare a hash map mapping int values to wxString like this:

```
WX_DECLARE_HASH_MAP( int,
                     wxString,
                     wxIntegerHash,
                     wxIntegerEqual,
                     MyHash );

// using an user-defined class for keys
class MyKey { ... };

// hashing function
class MyKeyHash
{
public:
    MyKeyHash() { }

    unsigned long operator()( const MyKey& k ) const
        {
            // compute the hash
        }

    MyKeyHash& operator=(const MyKeyHash&) { return *this; }
};

// comparison operator
class MyKeyEqual
{
public:
    MyKeyEqual() { }
    bool operator()( const MyKey& a, const MyKey& b ) const
        {
           // compare for equality
        }

    MyKeyEqual& operator=(const MyKeyEqual&) { return *this; }
};

WX_DECLARE_HASH_MAP( MyKey,      // type of the keys
                     SOME_TYPE,  // any type you like
                     MyKeyHash,  // hasher
                     MyKeyEqual, // key equality predicate
                     CLASSNAME); // name of the class
```

21.337.3 Types

In the documentation below you should replace wxHashMap with the name you used in the class declaration.

- wxHashMap::key_type: Type of the hash keys.
- wxHashMap::mapped_type: Type of the values stored in the hash map.
- wxHashMap::value_type: Equivalent to struct { key_type first; mapped_type second }.
- wxHashMap::iterator: Used to enumerate all the elements in a hash map; it is similar to a value_type*.
- wxHashMap::const_iterator: Used to enumerate all the elements in a constant hash map; it is similar to a const value_type*.
- wxHashMap::size_type: Used for sizes.
- wxHashMap::Insert_Result: The return value for insert().

21.337.4 Iterators

An iterator is similar to a pointer, and so you can use the usual pointer operations: ++it (and it++) to move to the next element, *it to access the element pointed to, it->first (it->second) to access the key (value) of the element pointed to.

Hash maps provide forward only iterators, this means that you can't use –it, it + 3, it1 - it2.

21.337.5 Predefined hashmap types

wxWidgets defines the following hashmap types:

- wxLongToLongHashMap (uses long both for keys and values)
- wxStringToStringHashMap (uses wxString both for keys and values)

Library: wxBase

Category: Containers

Public Member Functions

- wxHashMap (size_type size=10)
  The size parameter is just a hint, the table will resize automatically to preserve performance.
- wxHashMap (const wxHashMap &map)
  Copy constructor.
- void clear ()
  Removes all elements from the hash map.
- size_type count (const key_type &key) const
  Counts the number of elements with the given key present in the map.
- bool empty () const
  Returns true if the hash map does not contain any elements, false otherwise.
- Insert_Result insert (const value_type &v)
  Inserts the given value in the hash map.
- mapped_type operator[] (const key_type &key)
  Use the key as an array subscript.
- size_type size () const
  Returns the number of elements in the map.

- const_iterator begin () const
  Returns an iterator pointing at the first element of the hash map.
- iterator begin ()
  Returns an iterator pointing at the first element of the hash map.

- const_iterator end () const
  Returns an iterator pointing at the one-after-the-last element of the hash map.
- iterator end ()
  Returns an iterator pointing at the one-after-the-last element of the hash map.

- size_type erase (const key_type &key)
  Erases the element with the given key, and returns the number of elements erased (either 0 or 1).
- void erase (iterator it)
  Erases the element pointed to by the iterator.
- void erase (const_iterator it)
  Erases the element with the given key, and returns the number of elements erased (either 0 or 1).

- iterator find (const key_type &key) const
  If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.
- const_iterator find (const key_type &key) const
  If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.

21.337.6 Constructor & Destructor Documentation

wxHashMap::wxHashMap (size_type *size* = 10)

The size parameter is just a hint, the table will resize automatically to preserve performance.

wxHashMap::wxHashMap (const wxHashMap & *map*)

Copy constructor.

21.337.7 Member Function Documentation

const_iterator wxHashMap::begin ( ) const

Returns an iterator pointing at the first element of the hash map.

Please remember that hash maps do not guarantee ordering.

iterator wxHashMap::begin ( )

Returns an iterator pointing at the first element of the hash map.

Please remember that hash maps do not guarantee ordering.

void wxHashMap::clear ( )

Removes all elements from the hash map.

size_type wxHashMap::count (const key_type & *key*) const

Counts the number of elements with the given key present in the map.

This function returns only 0 or 1.

bool wxHashMap::empty ( ) const

Returns true if the hash map does not contain any elements, false otherwise.

const_iterator wxHashMap::end ( ) const

Returns an iterator pointing at the one-after-the-last element of the hash map.

Please remember that hash maps do not guarantee ordering.

iterator wxHashMap::end ( )

Returns an iterator pointing at the one-after-the-last element of the hash map.

Please remember that hash maps do not guarantee ordering.

size_type wxHashMap::erase (const key_type & *key*)

Erases the element with the given key, and returns the number of elements erased (either 0 or 1).

void wxHashMap::erase (iterator *it*)

Erases the element pointed to by the iterator.

After the deletion the iterator is no longer valid and must not be used.

void wxHashMap::erase (const_iterator *it*)

Erases the element with the given key, and returns the number of elements erased (either 0 or 1).

iterator wxHashMap::find (const key_type & *key*) const

If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.

```
hashmap.find( non_existent_key ) == hashmap.end()
```

const_iterator wxHashMap::find (const key_type & *key*) const

If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.

```
hashmap.find( non_existent_key ) == hashmap.end()
```

Insert_Result wxHashMap::insert (const value_type & *v*)

Inserts the given value in the hash map.

The return value is equivalent to a

```
std::pair<wxHashMap::iterator, bool>
```

The iterator points to the inserted element, the boolean value is true if *v* was actually inserted.

mapped_type wxHashMap::operator[] (const key_type & *key*)

Use the key as an array subscript.

The only difference is that if the given key is not present in the hash map, an element with the default `value_type()` is inserted in the table.

size_type wxHashMap::size ( ) const

Returns the number of elements in the map.

21.338 wxHashSet Class Reference

```
#include <wx/hashset.h>
```

21.338.1 Detailed Description

This is a simple, type-safe, and reasonably efficient hash set class, whose interface is a subset of the interface of STL containers.

The interface is similar to std::tr1::hash_set or std::set classes but notice that, unlike std::set, the contents of a hash set is not sorted.

Example:

```
class MyClass { ... };

// same, with MyClass* keys (only uses pointer equality!)
WX_DECLARE_HASH_SET( MyClass*, wxPointerHash, wxPointerEqual, MySet1 );
// same, with int keys
WX_DECLARE_HASH_SET( int, wxIntegerHash, wxIntegerEqual, MySet2 );
// declare a hash set with string keys
WX_DECLARE_HASH_SET( wxString, wxStringHash, wxStringEqual, MySet3 );

MySet1 h1;
MySet2 h1;
MySet3 h3;

// store and retrieve values
h1.insert( new MyClass( 1 ) );

h3.insert( "foo" );
h3.insert( "bar" );
h3.insert( "baz" );

int size = h3.size(); // now is three
bool has_foo = h3.find( "foo" ) != h3.end();

h3.insert( "bar" ); // still has size three

// iterate over all the elements in the class
MySet3::iterator it;
for( it = h3.begin(); it != h3.end(); ++it )
{
    wxString key = *it;
    // do something useful with key
}
```

21.338.2 Declaring new hash set types

```
WX_DECLARE_HASH_SET( KEY_T,      // type of the keys
                     HASH_T,     // hasher
                     KEY_EQ_T,   // key equality predicate
                     CLASSNAME); // name of the class
```

The HASH_T and KEY_EQ_T are the types used for the hashing function and key comparison. wxWidgets provides three predefined hashing functions: wxIntegerHash for integer types (int, long, short, and their unsigned counterparts), wxStringHash for strings (wxString, wxChar*, char*), and wxPointerHash for any kind of pointer. Similarly three equality predicates: wxIntegerEqual, wxStringEqual, wxPointerEqual are provided. Using this you could declare a hash set using int values like this:

```
WX_DECLARE_HASH_SET( int,
                     wxIntegerHash,
                     wxIntegerEqual,
                     MySet );

// using an user-defined class for keys
class MyKey { ... };

// hashing function
class MyKeyHash
{
public:
    MyKeyHash() { }

    unsigned long operator()( const MyKey& k ) const
        {
            // compute the hash
        }

    MyKeyHash& operator=(const MyKeyHash&) { return *this; }
};
```

```
// comparison operator
class MyKeyEqual
{
public:
    MyKeyEqual() { }
    bool operator()( const MyKey& a, const MyKey& b ) const
        {
            // compare for equality
        }

    MyKeyEqual& operator=(const MyKeyEqual&) { return *this; }
};

WX_DECLARE_HASH_SET( MyKey,      // type of the keys
                     MyKeyHash,  // hasher
                     MyKeyEqual, // key equality predicate
                     CLASSNAME); // name of the class
```

21.338.3 Types

In the documentation below you should replace wxHashSet with the name you used in the class declaration.

- wxHashSet::key_type: Type of the hash keys
- wxHashSet::mapped_type: Type of hash keys
- wxHashSet::value_type: Type of hash keys
- wxHashSet::iterator: Used to enumerate all the elements in a hash set; it is similar to a value_type*
- wxHashSet::const_iterator: Used to enumerate all the elements in a constant hash set; it is similar to a const value_type*
- wxHashSet::size_type: Used for sizes
- wxHashSet::Insert_Result: The return value for insert()

21.338.4 Iterators

An iterator is similar to a pointer, and so you can use the usual pointer operations: ++it (and it++) to move to the next element, *it to access the element pointed to, *it to access the value of the element pointed to. Hash sets provide forward only iterators, this means that you can't use –it, it + 3, it1 - it2.

Library: wxBase

Category: Containers

Public Member Functions

- wxHashSet (size_type size=10)

  The size parameter is just a hint, the table will resize automatically to preserve performance.
- wxHashSet (const wxHashSet &set)

  Copy constructor.
- void clear ()

  Removes all elements from the hash set.
- size_type count (const key_type &key) const

  Counts the number of elements with the given key present in the set.
- bool empty () const

Returns true if the hash set does not contain any elements, false otherwise.

- size_type erase (const key_type &key)

  Erases the element with the given key, and returns the number of elements erased (either 0 or 1).
- Insert_Result insert (const value_type &v)

  Inserts the given value in the hash set.
- size_type size () const

  Returns the number of elements in the set.

- const_iterator begin () const

  Returns an iterator pointing at the first element of the hash set.
- iterator begin ()

  Returns an iterator pointing at the first element of the hash set.

- const_iterator end () const

  Returns an iterator pointing at the one-after-the-last element of the hash set.
- iterator end ()

  Returns an iterator pointing at the one-after-the-last element of the hash set.

- void erase (iterator it)

  Erases the element pointed to by the iterator.
- void erase (const_iterator it)

  Erases the element pointed to by the iterator.

- iterator find (const key_type &key) const

  If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.
- const_iterator find (const key_type &key) const

  If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.

21.338.5 Constructor & Destructor Documentation

wxHashSet::wxHashSet (size_type *size* = `10`)

The size parameter is just a hint, the table will resize automatically to preserve performance.

wxHashSet::wxHashSet (const wxHashSet & *set*)

Copy constructor.

21.338.6 Member Function Documentation

const_iterator wxHashSet::begin ( ) const

Returns an iterator pointing at the first element of the hash set.

Please remember that hash sets do not guarantee ordering.

iterator wxHashSet::begin ( )

Returns an iterator pointing at the first element of the hash set.

Please remember that hash sets do not guarantee ordering.

void wxHashSet::clear ( )

Removes all elements from the hash set.

size_type wxHashSet::count (const key_type & *key*) const

Counts the number of elements with the given key present in the set.

This function returns only 0 or 1.

bool wxHashSet::empty ( ) const

Returns true if the hash set does not contain any elements, false otherwise.

const_iterator wxHashSet::end ( ) const

Returns an iterator pointing at the one-after-the-last element of the hash set.

Please remember that hash sets do not guarantee ordering.

iterator wxHashSet::end ( )

Returns an iterator pointing at the one-after-the-last element of the hash set.

Please remember that hash sets do not guarantee ordering.

size_type wxHashSet::erase (const key_type & *key*)

Erases the element with the given key, and returns the number of elements erased (either 0 or 1).

void wxHashSet::erase (iterator *it*)

Erases the element pointed to by the iterator.

After the deletion the iterator is no longer valid and must not be used.

void wxHashSet::erase (const_iterator *it*)

Erases the element pointed to by the iterator.

After the deletion the iterator is no longer valid and must not be used.

iterator wxHashSet::find (const key_type & *key*) const

If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.

i.e.

```
hashset.find( non_existent_key ) == hashset.end()
```

const_iterator wxHashSet::find (const key_type & *key*) const

If an element with the given key is present, the functions returns an iterator pointing at that element, otherwise an invalid iterator is returned.

i.e.

```
hashset.find( non_existent_key ) == hashset.end()
```

Insert_Result wxHashSet::insert (const value_type & *v*)

Inserts the given value in the hash set.

The return value is equivalent to a

```
std::pair<wxHashMap::iterator, bool>
```

The iterator points to the inserted element, the boolean value is true if *v* was actually inserted.

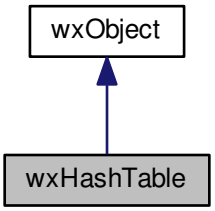
size_type wxHashSet::size ( ) const

Returns the number of elements in the set.

21.339 wxHashTable Class Reference

`#include <wx/hash.h>`

Inheritance diagram for wxHashTable:



21.339.1 Detailed Description

Deprecated Please note that this class is retained for backward compatibility reasons; you should use wxHashMap.

This class provides hash table functionality for wxWidgets, and for an application if it wishes. Data can be hashed on an integer or string key.

Example:

```
wxHashTable table(wxKEY_STRING);

wxPoint *point = new wxPoint(100, 200);
```

```
table.Put("point 1", point);

....

wxPoint *found_point = (wxPoint *)table.Get("point 1");
```

A hash table is implemented as an array of pointers to lists. When no data has been stored, the hash table takes only a little more space than this array (default size is 1000). When a data item is added, an integer is constructed from the integer or string key that is within the bounds of the array. If the array element is NULL, a new (keyed) list is created for the element. Then the data object is appended to the list, storing the key in case other data objects need to be stored in the list also (when a 'collision' occurs).

Retrieval involves recalculating the array index from the key, and searching along the keyed list for the data object whose stored key matches the passed key. Obviously this is quicker when there are fewer collisions, so hashing will become inefficient if the number of items to be stored greatly exceeds the size of the hash table.

Library: wxBase

Category: Containers

See also

wxList

Public Member Functions

- wxHashTable (wxKeyType key_type=wxKEY_INTEGER, size_t size=1000)

  Constructor.
- virtual ~wxHashTable ()

  Destroys the hash table.
- void BeginFind ()

  The counterpart of Next().
- void Clear ()

  Clears the hash table of all nodes (but as usual, doesn't delete user data).
- void DeleteContents (bool flag)

  If set to true data stored in hash table will be deleted when hash table object is destroyed.
- size_t GetCount () const

  Returns the number of elements in the hash table.
- wxHashTable::Node * Next ()

  If the application wishes to iterate through all the data in the hash table, it can call BeginFind() and then loop on Next().
- wxObject * Delete (long key)

  Deletes entry in hash table and returns the user's data (if found).
- wxObject * Delete (const wxString &key)

  Deletes entry in hash table and returns the user's data (if found).
- wxObject * Get (long key)

  Gets data from the hash table, using an integer or string key (depending on which has table constructor was used).
- wxObject * Get (const char *key)

  Gets data from the hash table, using an integer or string key (depending on which has table constructor was used).
- void Put (long key, wxObject *object)

  Inserts data into the hash table, using an integer or string key (depending on which has table constructor was used).
- void Put (const char *key, wxObject *object)

  Inserts data into the hash table, using an integer or string key (depending on which has table constructor was used).

Static Public Member Functions

- static long MakeKey (const wxString &string)
  Makes an integer key out of a string.

Additional Inherited Members

21.339.2 Constructor & Destructor Documentation

wxHashTable::wxHashTable (wxKeyType *key_type* = `wxKEY_INTEGER`, size_t *size* = `1000`)

Constructor.

key_type is one of wxKEY_INTEGER, or wxKEY_STRING, and indicates what sort of keying is required. *size* is optional.

virtual wxHashTable::~wxHashTable ( ) `[virtual]`

Destroys the hash table.

21.339.3 Member Function Documentation

void wxHashTable::BeginFind ( )

The counterpart of Next().

If the application wishes to iterate through all the data in the hash table, it can call BeginFind() and then loop on Next().

void wxHashTable::Clear ( )

Clears the hash table of all nodes (but as usual, doesn't delete user data).

wxObject* wxHashTable::Delete (long *key*)

Deletes entry in hash table and returns the user's data (if found).

wxObject* wxHashTable::Delete (const wxString & *key*)

Deletes entry in hash table and returns the user's data (if found).

void wxHashTable::DeleteContents (bool *flag*)

If set to true data stored in hash table will be deleted when hash table object is destroyed.

wxObject* wxHashTable::Get (long *key*)

Gets data from the hash table, using an integer or string key (depending on which has table constructor was used).

wxObject* wxHashTable::Get (const char * *key*)

Gets data from the hash table, using an integer or string key (depending on which has table constructor was used).

size_t wxHashTable::GetCount ( ) const

Returns the number of elements in the hash table.

static long wxHashTable::MakeKey (const wxString & *string*) [static]

Makes an integer key out of a string.

An application may wish to make a key explicitly (for instance when combining two data values to form a key).

wxHashTable::Node* wxHashTable::Next ( )

If the application wishes to iterate through all the data in the hash table, it can call BeginFind() and then loop on Next().

This function returns a **wxHashTable::Node** pointer (or NULL if there are no more nodes).

The return value is functionally equivalent to **wxNode** but might not be implemented as a **wxNode**. The user will probably only wish to use the wxNode::GetData() method to retrieve the data; the node may also be deleted.

void wxHashTable::Put (long *key,* wxObject * *object*)

Inserts data into the hash table, using an integer or string key (depending on which has table constructor was used).

The key string is copied and stored by the hash table implementation.

void wxHashTable::Put (const char * *key,* wxObject * *object*)

Inserts data into the hash table, using an integer or string key (depending on which has table constructor was used).

The key string is copied and stored by the hash table implementation.

21.340 wxHeaderButtonParams Struct Reference

`#include <wx/renderer.h>`

21.340.1 Detailed Description

This `struct` can optionally be used with wxRendererNative::DrawHeaderButton() to specify custom values used to draw the text or bitmap label.

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- wxHeaderButtonParams ()

Public Attributes

- wxColour m_arrowColour
- wxColour m_selectionColour
- wxString m_labelText
- wxFont m_labelFont
- wxColour m_labelColour
- wxBitmap m_labelBitmap
- int m_labelAlignment

21.340.2 Constructor & Destructor Documentation

wxHeaderButtonParams::wxHeaderButtonParams ( )

21.340.3 Member Data Documentation

wxColour wxHeaderButtonParams::m_arrowColour

int wxHeaderButtonParams::m_labelAlignment

wxBitmap wxHeaderButtonParams::m_labelBitmap

wxColour wxHeaderButtonParams::m_labelColour

wxFont wxHeaderButtonParams::m_labelFont

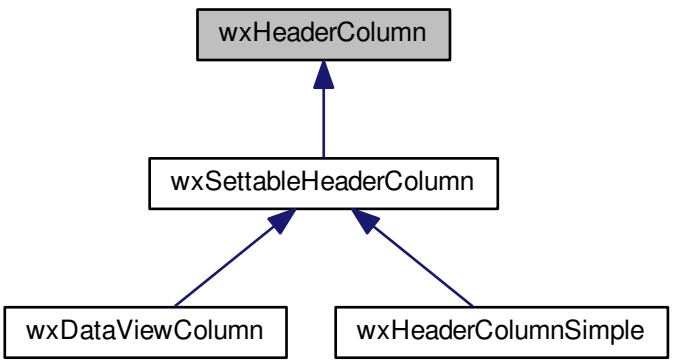
wxString wxHeaderButtonParams::m_labelText

wxColour wxHeaderButtonParams::m_selectionColour

21.341 wxHeaderColumn Class Reference

`#include <wx/headercol.h>`

Inheritance diagram for wxHeaderColumn:



21.341.1 Detailed Description

Represents a column header in controls displaying tabular data such as wxDataViewCtrl or wxGrid.

Notice that this is an abstract base class which is implemented (usually using the information stored in the associated control) by the different controls using wxHeaderCtrl. As the control only needs to retrieve the information about the column, this class defines only the methods for accessing the various column properties but not for changing them as the setters might not be needed at all, e.g. if the column attributes can only be changed via the methods of the main associated control (this is the case for wxGrid for example). If you do want to allow changing them directly using the column itself, you should inherit from wxSettableHeaderColumn instead of this class.

Finally, if you don't already store the column information at all anywhere, you should use the concrete wxHeaderColumnSimple class and wxHeaderCtrlSimple.

Library: wxCore

Category: Controls

Public Member Functions

- virtual wxString GetTitle () const =0

  Get the text shown in the column header.
- virtual wxBitmap GetBitmap () const =0

  Returns the bitmap in the header of the column, if any.
- virtual int GetWidth () const =0

  Returns the current width of the column.
- virtual int GetMinWidth () const =0

  Return the minimal column width.
- virtual wxAlignment GetAlignment () const =0

  Returns the current column alignment.
- virtual int GetFlags () const =0

  Get the column flags.
- bool HasFlag (int flag) const

  Return true if the specified flag is currently set for this column.
- virtual bool IsResizeable () const

  Return true if the column can be resized by the user.
- virtual bool IsSortable () const

  Returns true if the column can be clicked by user to sort the control contents by the field in this column.
- virtual bool IsReorderable () const

  Returns true if the column can be dragged by user to change its order.
- virtual bool IsHidden () const

  Returns true if the column is currently hidden.
- bool IsShown () const

  Returns true if the column is currently shown.
- virtual bool IsSortKey () const =0

  Returns true if the column is currently used for sorting.
- virtual bool IsSortOrderAscending () const =0

  Returns true, if the sort order is ascending.

21.341.2 Member Function Documentation

virtual wxAlignment wxHeaderColumn::GetAlignment ( ) const [pure virtual]

Returns the current column alignment.

Returns

One of wxALIGN_CENTRE, wxALIGN_LEFT or wxALIGN_RIGHT.

Implemented in wxHeaderColumnSimple.

virtual wxBitmap wxHeaderColumn::GetBitmap ( ) const [pure virtual]

Returns the bitmap in the header of the column, if any.

If the column has no associated bitmap, wxNullBitmap should be returned.

Implemented in wxHeaderColumnSimple.

virtual int wxHeaderColumn::GetFlags ( ) const [pure virtual]

Get the column flags.

This method retrieves all the flags at once, you can also use HasFlag() to test for any individual flag or IsResizeable(), IsSortable(), IsReorderable() and IsHidden() to test for particular flags.

Implemented in wxHeaderColumnSimple.

virtual int wxHeaderColumn::GetMinWidth ( ) const [pure virtual]

Return the minimal column width.

Returns

The minimal width such that the user can't resize the column to lesser size (notice that it is still possible to set the column width to smaller value from the program code). Return 0 from here to allow resizing the column to arbitrarily small size.

Implemented in wxHeaderColumnSimple.

virtual wxString wxHeaderColumn::GetTitle ( ) const [pure virtual]

Get the text shown in the column header.

Implemented in wxHeaderColumnSimple.

virtual int wxHeaderColumn::GetWidth ( ) const [pure virtual]

Returns the current width of the column.

Returns

Width of the column in pixels, never wxCOL_WIDTH_DEFAULT or wxCOL_WIDTH_AUTOSIZE.

Implemented in wxHeaderColumnSimple.

bool wxHeaderColumn::HasFlag (int *flag*) const

Return true if the specified flag is currently set for this column.

virtual bool wxHeaderColumn::IsHidden ( ) const `[virtual]`

Returns true if the column is currently hidden.

This corresponds to wxCOL_HIDDEN flag which is off by default.

virtual bool wxHeaderColumn::IsReorderable ( ) const `[virtual]`

Returns true if the column can be dragged by user to change its order.

This corresponds to wxCOL_REORDERABLE flag which is on by default.

virtual bool wxHeaderColumn::IsResizeable ( ) const `[virtual]`

Return true if the column can be resized by the user.

Equivalent to HasFlag(wxCOL_RESIZABLE).

bool wxHeaderColumn::IsShown ( ) const

Returns true if the column is currently shown.

This corresponds to the absence of wxCOL_HIDDEN flag.

virtual bool wxHeaderColumn::IsSortable ( ) const `[virtual]`

Returns true if the column can be clicked by user to sort the control contents by the field in this column.

This corresponds to wxCOL_SORTABLE flag which is off by default.

virtual bool wxHeaderColumn::IsSortKey ( ) const `[pure virtual]`

Returns true if the column is currently used for sorting.

Implemented in wxHeaderColumnSimple.

virtual bool wxHeaderColumn::IsSortOrderAscending ( ) const `[pure virtual]`

Returns true, if the sort order is ascending.

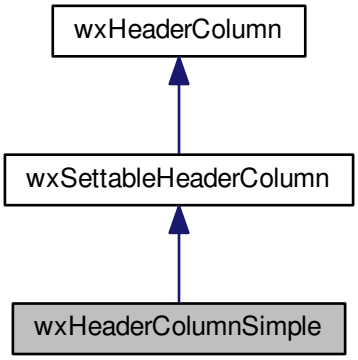
Notice that it only makes sense to call this function if the column is used for sorting at all, i.e. if IsSortKey() returns true.

Implemented in wxHeaderColumnSimple.

21.342 wxHeaderColumnSimple Class Reference

`#include <wx/headercol.h>`

Inheritance diagram for wxHeaderColumnSimple:



21.342.1 Detailed Description

Simple container for the information about the column.

This is a concrete class implementing all wxSettableHeaderColumn class methods in a trivial way, i.e. by just storing the information in the object itself. It is used by and with wxHeaderCtrlSimple, e.g.

```
wxHeaderCtrlSimple * header = new wxHeaderCtrlSimple(...);
wxHeaderColumnSimple col("Title");
col.SetWidth(100);
col.SetSortable(100);
header->AppendColumn(col);
```

Library: wxCore

Category: Controls

Public Member Functions

- wxHeaderColumnSimple (const wxString &title, int width=wxCOL_WIDTH_DEFAULT, wxAlignment align=wxALIGN_NOT, int flags=wxCOL_DEFAULT_FLAGS)

  Constructor for a column header.
- wxHeaderColumnSimple (const wxBitmap &bitmap, int width=wxCOL_WIDTH_DEFAULT, wxAlignment align=wxALIGN_CENTER, int flags=wxCOL_DEFAULT_FLAGS)

  Constructor for a column header.

- virtual void SetTitle (const wxString &title)

  Trivial implementations of the base class pure virtual functions.
- virtual wxString GetTitle () const

  Trivial implementations of the base class pure virtual functions.
- virtual void SetBitmap (const wxBitmap &bitmap)

  Trivial implementations of the base class pure virtual functions.
- virtual wxBitmap GetBitmap () const

Trivial implementations of the base class pure virtual functions.

- virtual void SetWidth (int width)

  Trivial implementations of the base class pure virtual functions.
- virtual int GetWidth () const

  Trivial implementations of the base class pure virtual functions.
- virtual void SetMinWidth (int minWidth)

  Trivial implementations of the base class pure virtual functions.
- virtual int GetMinWidth () const

  Trivial implementations of the base class pure virtual functions.
- virtual void SetAlignment (wxAlignment align)

  Trivial implementations of the base class pure virtual functions.
- virtual wxAlignment GetAlignment () const

  Trivial implementations of the base class pure virtual functions.
- virtual void SetFlags (int flags)

  Trivial implementations of the base class pure virtual functions.
- virtual int GetFlags () const

  Trivial implementations of the base class pure virtual functions.
- virtual bool IsSortKey () const

  Trivial implementations of the base class pure virtual functions.
- virtual void SetSortOrder (bool ascending)

  Trivial implementations of the base class pure virtual functions.
- virtual bool IsSortOrderAscending () const

  Trivial implementations of the base class pure virtual functions.

21.342.2 Constructor & Destructor Documentation

wxHeaderColumnSimple::wxHeaderColumnSimple (const wxString & *title*, int *width* = wxCOL_WIDTH_DEFAULT, wxAlignment *align* = wxALIGN_NOT, int *flags* = wxCOL_DEFAULT_FLAGS)

Constructor for a column header.

The first constructor creates a header showing the given text *title* while the second one creates one showing the specified *bitmap* image.

wxHeaderColumnSimple::wxHeaderColumnSimple (const wxBitmap & *bitmap*, int *width* = wxCOL_WIDTH_DEFAULT, wxAlignment *align* = wxALIGN_CENTER, int *flags* = wxCOL_DEFAULT_FLAGS)

Constructor for a column header.

The first constructor creates a header showing the given text *title* while the second one creates one showing the specified *bitmap* image.

21.342.3 Member Function Documentation

virtual wxAlignment wxHeaderColumnSimple::GetAlignment ( ) const `[virtual]`

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual wxBitmap wxHeaderColumnSimple::GetBitmap ( ) const `[virtual]`

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual int wxHeaderColumnSimple::GetFlags ( ) const [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual int wxHeaderColumnSimple::GetMinWidth ( ) const [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual wxString wxHeaderColumnSimple::GetTitle ( ) const [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual int wxHeaderColumnSimple::GetWidth ( ) const [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual bool wxHeaderColumnSimple::IsSortKey ( ) const [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual bool wxHeaderColumnSimple::IsSortOrderAscending ( ) const [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxHeaderColumn.

virtual void wxHeaderColumnSimple::SetAlignment (wxAlignment *align*) [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

virtual void wxHeaderColumnSimple::SetBitmap (const wxBitmap & *bitmap*) [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

virtual void wxHeaderColumnSimple::SetFlags (int *flags*) [virtual]

Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

virtual void wxHeaderColumnSimple::SetMinWidth (int *minWidth*) `[virtual]`

Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

virtual void wxHeaderColumnSimple::SetSortOrder (bool *ascending*) `[virtual]`

Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

virtual void wxHeaderColumnSimple::SetTitle (const wxString & *title*) `[virtual]`

Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

virtual void wxHeaderColumnSimple::SetWidth (int *width*) `[virtual]`

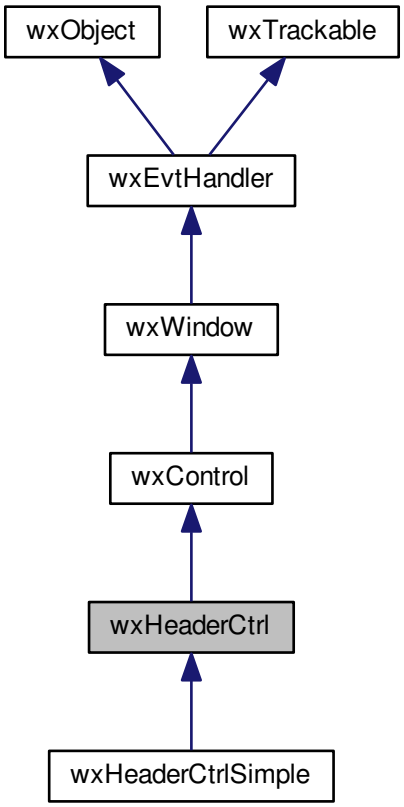
Trivial implementations of the base class pure virtual functions.

Implements wxSettableHeaderColumn.

21.343 wxHeaderCtrl Class Reference

`#include <wx/headerctrl.h>`

Inheritance diagram for wxHeaderCtrl:



21.343.1 Detailed Description

wxHeaderCtrl is the control containing the column headings which is usually used for display of tabular data.

It is used as part of wxGrid, in generic version wxDataViewCtrl and report view of wxListCtrl but can be also used independently. In general this class is meant to be used as part of another control which already stores the column information somewhere as it can't be used directly: instead you need to inherit from it and implement the GetColumn() method to provide column information. See wxHeaderCtrlSimple for a concrete control class which can be used directly.

In addition to labeling the columns, the control has the following features:

- Column reordering support, either by explicitly configuring the columns order and calling SetColumnsOrder() or by dragging the columns interactively (if enabled).
- Display of the icons in the header: this is often used to display a sort or reverse sort indicator when the column header is clicked.

Notice that this control itself doesn't do anything other than displaying the column headers. In particular column reordering and sorting must still be supported by the associated control displaying the real data under the header. Also remember to call ScrollWindow() method of the control if the associated data display window has a horizontal scrollbar, otherwise the headers wouldn't align with the data when the window is scrolled.

This control is implemented using the native header control under MSW systems and a generic implementation elsewhere.

21.343.2 Future Improvements

Some features are supported by the native MSW control and so could be easily implemented in this version of wxHeaderCtrl but need to be implemented in the generic version as well to be really useful. Please let us know if you need or, better, plan to work on implementing, any of them:

- Displaying bitmaps instead of or together with the text
- Custom drawn headers
- Filters associated with a column.

Styles

This class supports the following styles:

- wxHD_ALLOW_REORDER: If this style is specified (it is by default), the user can reorder the control columns by dragging them.
- wxHD_ALLOW_HIDE: If this style is specified, the control shows a popup menu allowing the user to change the columns visibility on right mouse click. Notice that the program can always hide or show the columns, this style only affects the users capability to do it.
- wxHD_DEFAULT_STYLE: Symbolic name for the default control style, currently equal to `wxHD_ALLOW_REORDER`.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHeaderCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_HEADER_CLICK(id, func): A column heading was clicked.
- EVT_HEADER_RIGHT_CLICK(id, func): A column heading was right clicked.
- EVT_HEADER_MIDDLE_CLICK(id, func): A column heading was clicked with the middle mouse button.
- EVT_HEADER_DCLICK(id, func): A column heading was double clicked.
- EVT_HEADER_RIGHT_DCLICK(id, func): A column heading was right double clicked.
- EVT_HEADER_MIDDLE_DCLICK(id, func): A column heading was double clicked with the middle mouse button.
- EVT_HEADER_SEPARATOR_DCLICK(id, func): Separator to the right of the specified column was double clicked (this action is commonly used to resize the column to fit its contents width and the control provides UpdateColumnWidthToFit() method to make implementing this easier).
- EVT_HEADER_BEGIN_RESIZE(id, func): The user started to drag the separator to the right of the column with the specified index (this can only happen for the columns for which wxHeaderColumn::IsResizeable() returns true). The event can be vetoed to prevent the column from being resized. If it isn't, the resizing and end resize (or dragging cancelled) events will be generated later.
- EVT_HEADER_RESIZING(id, func): The user is dragging the column with the specified index resizing it and its current width is wxHeaderCtrlEvent::GetWidth(). The event can be vetoed to stop the dragging operation completely at any time.

- EVT_HEADER_END_RESIZE(id, func): The user stopped dragging the column by releasing the mouse. The column should normally be resized to the value of wxHeaderCtrlEvent::GetWidth().
- EVT_HEADER_BEGIN_REORDER(id, func): The user started to drag the column with the specified index (this can only happen for the controls with wxHD_ALLOW_REORDER style). This event can be vetoed to prevent the column from being reordered, otherwise the end reorder message will be generated later.
- EVT_HEADER_END_REORDER(id, func): The user dropped the column in its new location. The event can be vetoed to prevent the column from being placed at the new position or handled to update the display of the data in the associated control to match the new column location (available from wxHeaderCtrlEvent::GetNewOrder()).
- EVT_HEADER_DRAGGING_CANCELLED(id, func): The resizing or reordering operation currently in progress was cancelled. This can happen if the user pressed Esc key while dragging the mouse or the mouse capture was lost for some other reason. You only need to handle this event if your application entered into some modal mode when resizing or reordering began, in which case it should handle this event in addition to the matching end resizing or reordering ones.

Library: wxCore

Category: Controls

See also

wxGrid, wxListCtrl, wxDataViewCtrl

Public Member Functions

- wxHeaderCtrl ()

  Default constructor not creating the underlying window.
- wxHeaderCtrl (wxWindow *parent, wxWindowID winid=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHD_DEFAULT_STYLE, const wxString &name=wxHeaderCtrlNameStr)

  Constructor creating the window.
- bool Create (wxWindow *parent, wxWindowID winid=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHD_DEFAULT_STYLE, const wxString &name=wxHeaderCtrlNameStr)

  Create the header control window.
- void SetColumnCount (unsigned int count)

  Set the number of columns in the control.
- unsigned int GetColumnCount () const

  Return the number of columns in the control.
- bool IsEmpty () const

  Return whether the control has any columns.
- void UpdateColumn (unsigned int idx)

  Update the column with the given index.
- void SetColumnsOrder (const wxArrayInt &order)

  Change the columns display order.
- wxArrayInt GetColumnsOrder () const

  Return the array describing the columns display order.
- unsigned int GetColumnAt (unsigned int pos) const

  Return the index of the column displayed at the given position.
- unsigned int GetColumnPos (unsigned int idx) const

Get the position at which this column is currently displayed.

- void ResetColumnsOrder ()

  Reset the columns order to the natural one.

- bool ShowColumnsMenu (const wxPoint &pt, const wxString &title=wxString())

  Show the popup menu allowing the user to show or hide the columns.

- void AddColumnsItems (wxMenu &menu, int idColumnsBase=0)

  Helper function appending the checkable items corresponding to all the columns to the given menu.

- bool ShowCustomizeDialog ()

  Show the column customization dialog.

- int GetColumnTitleWidth (const wxHeaderColumn &col)

  Returns width needed for given column's title.

Static Public Member Functions

- static void MoveColumnInOrderArray (wxArrayInt &order, unsigned int idx, unsigned int pos)

  Helper function to manipulate the array of column indices.

Protected Member Functions

- virtual const wxHeaderColumn & GetColumn (unsigned int idx) const =0

  Method to be implemented by the derived classes to return the information for the given column.

- virtual void UpdateColumnVisibility (unsigned int idx, bool show)

  Method called when the column visibility is changed by the user.

- virtual void UpdateColumnsOrder (const wxArrayInt &order)

  Method called when the columns order is changed in the customization dialog.

- virtual bool UpdateColumnWidthToFit (unsigned int idx, int widthTitle)

  Method which may be implemented by the derived classes to allow double clicking the column separator to resize the column to fit its contents.

- virtual void OnColumnCountChanging (unsigned int count)

  Can be overridden in the derived class to update internal data structures when the number of the columns in the control changes.

Additional Inherited Members

21.343.3 Constructor & Destructor Documentation

wxHeaderCtrl::wxHeaderCtrl ( )

Default constructor not creating the underlying window.

You must use Create() after creating the object using this constructor.

wxHeaderCtrl::wxHeaderCtrl (wxWindow * *parent*, wxWindowID *winid* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHD_DEFAULT_STYLE, const wxString & *name* = `wxHeaderCtrlNameStr`)

Constructor creating the window.

Please see Create() for the parameters documentation.

21.343.4 Member Function Documentation

void wxHeaderCtrl::AddColumnsItems (wxMenu & *menu*, int *idColumnsBase* = 0)

Helper function appending the checkable items corresponding to all the columns to the given menu.

This function is used by ShowColumnsMenu() but can also be used if you show your own custom columns menu and still want all the columns shown in it. It appends menu items with column labels as their text and consecutive ids starting from *idColumnsBase* to the menu and checks the items corresponding to the currently visible columns.

Example of use:

```
wxMenu menu;
menu.Append(100, "Some custom command");
menu.AppendSeparator();
AddColumnsItems(menu, 200);
const int rc = GetPopupMenuSelectionFromUser(menu, pt);
if ( rc >= 200 )
    ... toggle visibility of the column rc-200 ...
```

Parameters

menu	The menu to append the items to. It may be currently empty or not.
idColumnsBase	The id for the menu item corresponding to the first column, the other ones are consecutive starting from it. It should be positive.

bool wxHeaderCtrl::Create (wxWindow * *parent*, wxWindowID *winid* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHD_DEFAULT_STYLE, const wxString & *name* = wxHeaderCtrlNameStr)

Create the header control window.

Parameters

parent	The parent window. The header control should be typically positioned along the top edge of this window.
winid	Id of the control or wxID_ANY if you don't care.
pos	The initial position of the control.
size	The initial size of the control (usually not very useful as this control will typically be resized to have the same width as the associated data display control).
style	The control style, wxHD_DEFAULT_STYLE by default. Notice that the default style allows the user to reorder the columns by dragging them and you need to explicitly turn this feature off by using `wxHD_DEFAULT_STYLE & ~wxHD_ALLOW_REORDER` if this is undesirable.
name	The name of the control.

virtual const wxHeaderColumn& wxHeaderCtrl::GetColumn (unsigned int *idx*) const [protected], [pure virtual]

Method to be implemented by the derived classes to return the information for the given column.

Parameters

idx	The column index, between 0 and the value last passed to SetColumnCount().

unsigned int wxHeaderCtrl::GetColumnAt (unsigned int *pos*) const

Return the index of the column displayed at the given position.

Parameters

pos	The display position, e.g. 0 for the left-most column, 1 for the next one and so on until GetColumnCount() - 1.

See also

GetColumnPos()

unsigned int wxHeaderCtrl::GetColumnCount ( ) const

Return the number of columns in the control.

Returns

Number of columns as previously set by SetColumnCount().

See also

IsEmpty()

unsigned int wxHeaderCtrl::GetColumnPos (unsigned int *idx*) const

Get the position at which this column is currently displayed.

Notice that a valid position is returned even for the hidden columns currently.

Parameters

idx	The column index, must be less than GetColumnCount().

See also

GetColumnAt()

wxArrayInt wxHeaderCtrl::GetColumnsOrder ( ) const

Return the array describing the columns display order.

For the controls without wxHD_ALLOW_REORDER style the returned array will be the same as was passed to SetColumnsOrder() previously or define the default order (with n-th element being n) if it hadn't been called. But for the controls with wxHD_ALLOW_REORDER style, the columns can be also reordered by user.

int wxHeaderCtrl::GetColumnTitleWidth (const wxHeaderColumn & *col*)

Returns width needed for given column's title.

Since

2.9.4

bool wxHeaderCtrl::IsEmpty ( ) const

Return whether the control has any columns.

See also

GetColumnCount()

static void wxHeaderCtrl::MoveColumnInOrderArray (wxArrayInt & *order*, unsigned int *idx*, unsigned int *pos*) [static]

Helper function to manipulate the array of column indices.

This function reshuffles the array of column indices indexed by positions (i.e. using the same convention as for SetColumnsOrder()) so that the column with the given index is found at the specified position.

Parameters

order	Array containing the indices of columns in order of their positions.
idx	The index of the column to move.
pos	The new position for the column *idx*.

virtual void wxHeaderCtrl::OnColumnCountChanging (unsigned int *count*) [protected], [virtual]

Can be overridden in the derived class to update internal data structures when the number of the columns in the control changes.

This method is called by SetColumnCount() before effectively changing the number of columns.

The base class version does nothing but it is good practice to still call it from the overridden version in the derived class.

void wxHeaderCtrl::ResetColumnsOrder ( )

Reset the columns order to the natural one.

After calling this function, the column with index `idx` appears at position `idx` in the control.

void wxHeaderCtrl::SetColumnCount (unsigned int *count*)

Set the number of columns in the control.

The control will use GetColumn() to get information about all the new columns and refresh itself, i.e. this method also has the same effect as calling UpdateColumn() for all columns but it should only be used if the number of columns really changed.

void wxHeaderCtrl::SetColumnsOrder (const wxArrayInt & *order*)

Change the columns display order.

The display order defines the order in which the columns appear on the screen and does *not* affect the interpretation of indices by all the other class methods.

The *order* array specifies the column indices corresponding to the display positions.

Parameters

order	A permutation of all column indices, i.e. an array of size GetColumnsOrder() containing all column indices exactly once. The n-th element of this array defines the index of the column shown at the n-th position from left (for the default left-to-right writing direction).

See also

wxListCtrl::SetColumnsOrder()

bool wxHeaderCtrl::ShowColumnsMenu (const wxPoint & *pt*, const wxString & *title* = wxString())

Show the popup menu allowing the user to show or hide the columns.

This functions shows the popup menu containing all columns with check marks for the ones which are currently shown and allows the user to check or uncheck them to toggle their visibility. It is called from the default EVT_HEADER_RIGHT_CLICK handler for the controls which have wxHD_ALLOW_HIDE style. And if the column has wxHD_ALLOW_REORDER style as well, the menu also contains an item to customize the columns shown using which results in ShowCustomizeDialog() being called, please see its description for more details.

If a column was toggled, UpdateColumnVisibility() virtual function is called so it must be implemented for the controls with wxHD_ALLOW_HIDE style or if you call this function explicitly.

Parameters

pt	The position of the menu, in the header window coordinates.
title	The title for the menu if not empty.

Returns

true if a column was shown or hidden or false if nothing was done, e.g. because the menu was cancelled.

bool wxHeaderCtrl::ShowCustomizeDialog ( )

Show the column customization dialog.

This function displays a modal dialog containing the list of all columns which the user can use to reorder them as well as show or hide individual columns.

If the user accepts the changes done in the dialog, the virtual methods UpdateColumnVisibility() and UpdateColumnsOrder() will be called so they must be overridden in the derived class if this method is ever called. Please notice that the user will be able to invoke it interactively from the header popup menu if the control has both wxHD_ALLOW_HIDE and wxHD_ALLOW_REORDER styles.

See also

wxRearrangeDialog

void wxHeaderCtrl::UpdateColumn (unsigned int *idx*)

Update the column with the given index.

When the value returned by GetColumn() changes, this method must be called to notify the control about the change and update the visual display to match the new column data.

Parameters

idx	The column index, must be less than GetColumnCount().

virtual void wxHeaderCtrl::UpdateColumnsOrder (const wxArrayInt & *order*) [protected], [virtual]

Method called when the columns order is changed in the customization dialog.

This method is only called from ShowCustomizeDialog() when the user changes the order of columns. In particular it is *not* called if a single column changes place because the user dragged it to the new location, the EVT_HEADER_END_REORDER event handler should be used to react to this.

A typical implementation in a derived class will update the display order of the columns in the associated control, if any. Notice that there is no need to call SetColumnsOrder() from it as wxHeaderCtrl does it itself.

The base class version doesn't do anything and must be overridden if this method is called.

Parameters

order	The new column order. This array uses the same convention as SetColumnsOrder().

virtual void wxHeaderCtrl::UpdateColumnVisibility (unsigned int *idx*, bool *show*) [protected], [virtual]

Method called when the column visibility is changed by the user.

This method is called from ShowColumnsMenu() or ShowCustomizeDialog() when the user interactively hides or shows a column. A typical implementation will simply update the internally stored column state. Notice that there is no need to call UpdateColumn() from this method as it is already done by wxHeaderCtrl itself.

The base class version doesn't do anything and must be overridden if this method is called.

Parameters

idx	The index of the column whose visibility was toggled.
show	The new visibility value, true if the column is now shown or false if it is not hidden.

virtual bool wxHeaderCtrl::UpdateColumnWidthToFit (unsigned int *idx*, int *widthTitle*) [protected], [virtual]

Method which may be implemented by the derived classes to allow double clicking the column separator to resize the column to fit its contents.

When a separator is double clicked, the default handler of EVT_HEADER_SEPARATOR_DCLICK event calls this function and refreshes the column if it returns true so to implement the resizing of the column to fit its width on header double click you need to implement this method using logic similar to this example:

```
class MyHeaderColumn : public wxHeaderColumn
{
public:
    ...

    void SetWidth(int width) { m_width = width; }
    virtual int GetWidth() const { return m_width; }

private:
    int m_width;
};

class MyHeaderCtrl : public wxHeaderCtrl
{
public:
protected:
    virtual wxHeaderColumn& GetColumn(unsigned int idx) const
    {
        return m_cols[idx];
    }

    virtual bool UpdateColumnWidthToFit(unsigned int idx, int widthTitle)
    {
        int widthContents = ... compute minimal width for column idx ...
        m_cols[idx].SetWidth(wxMax(widthContents, widthTitle));
        return true;
    }

    wxVector<MyHeaderColumn> m_cols;
};
```

Base class version simply returns false.

Parameters

idx	The zero-based index of the column to update.
widthTitle	Contains minimal width needed to display the column header itself and will usually be used as a starting point for the fitting width calculation.

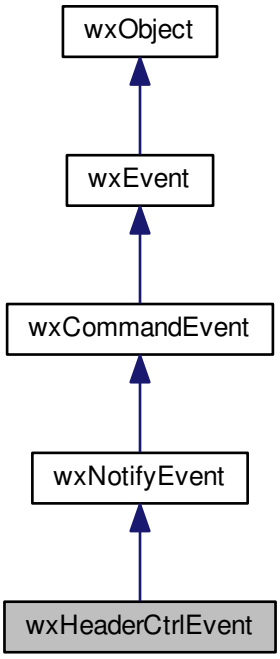
Returns

true to indicate that the column was resized, i.e. GetColumn() now returns the new width value, and so must be refreshed or false meaning that the control didn't reach to the separator double click.

21.344 wxHeaderCtrlEvent Class Reference

`#include <wx/headerctrl.h>`

Inheritance diagram for wxHeaderCtrlEvent:



21.344.1 Detailed Description

Event class representing the events generated by wxHeaderCtrl.

Library: wxCore

Category: Events

See also

wxHeaderCtrl

Public Member Functions

- wxHeaderCtrlEvent (wxEventType commandType=wxEVT_NULL, int winid=0)

- wxHeaderCtrlEvent (const wxHeaderCtrlEvent &event)
- int GetColumn () const

  Return the index of the column affected by this event.
- void SetColumn (int col)
- int GetWidth () const

  Return the current width of the column.
- void SetWidth (int width)
- unsigned int GetNewOrder () const

  Return the new order of the column.
- void SetNewOrder (unsigned int order)

Additional Inherited Members

21.344.2 Constructor & Destructor Documentation

wxHeaderCtrlEvent::wxHeaderCtrlEvent (wxEventType *commandType* = wxEVT_NULL, int *winid* = 0)

wxHeaderCtrlEvent::wxHeaderCtrlEvent (const wxHeaderCtrlEvent & *event*)

21.344.3 Member Function Documentation

int wxHeaderCtrlEvent::GetColumn ( ) const

Return the index of the column affected by this event.

This method can be called for all header control events.

unsigned int wxHeaderCtrlEvent::GetNewOrder ( ) const

Return the new order of the column.

This method can only be called for a reorder event for which it indicates the tentative new position for the column GetColumn() selected by the user. If the event is not vetoed, this will become the new column position in wxHeaderCtrl::GetColumnsOrder().

int wxHeaderCtrlEvent::GetWidth ( ) const

Return the current width of the column.

This method can only be called for the dragging events.

void wxHeaderCtrlEvent::SetColumn (int *col*)

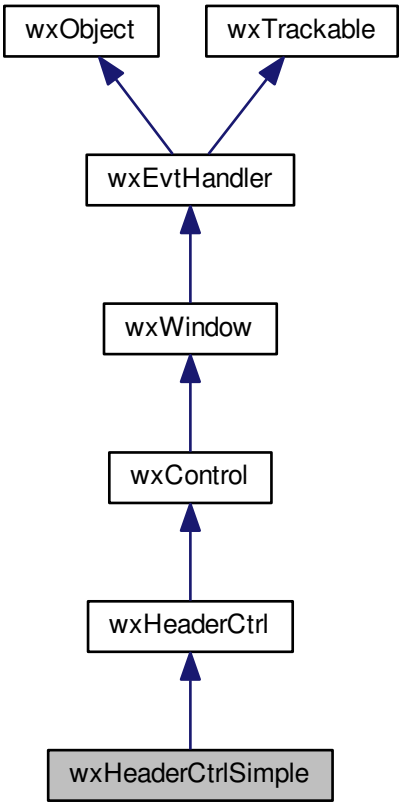
void wxHeaderCtrlEvent::SetNewOrder (unsigned int *order*)

void wxHeaderCtrlEvent::SetWidth (int *width*)

21.345 wxHeaderCtrlSimple Class Reference

```
#include <wx/headerctrl.h>
```

Inheritance diagram for wxHeaderCtrlSimple:



21.345.1 Detailed Description

wxHeaderCtrlSimple is a concrete header control which can be used directly, without inheriting from it as you need to do when using wxHeaderCtrl itself.

When using it, you need to use simple AppendColumn(), InsertColumn() and DeleteColumn() methods instead of setting the number of columns with SetColumnCount() and returning the information about them from the overridden GetColumn().

Library: wxCore

Category: Controls

See also

wxHeaderCtrl

Public Member Functions

- wxHeaderCtrlSimple ()

Default constructor not creating the underlying window.

- wxHeaderCtrlSimple (wxWindow *parent, wxWindowID winid=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHD_DEFAULT_STYLE, const wxString &name=wxHeaderCtrlNameStr)

  Constructor creating the window.
- void InsertColumn (const wxHeaderColumnSimple &col, unsigned int idx)

  Insert the column at the given position.
- void AppendColumn (const wxHeaderColumnSimple &col)

  Append the column to the end of the control.
- void DeleteColumn (unsigned int idx)

  Delete the column at the given position.
- void ShowColumn (unsigned int idx, bool show=true)

  Show or hide the column.
- void HideColumn (unsigned int idx)

  Hide the column with the given index.
- void ShowSortIndicator (unsigned int idx, bool sortOrder=true)

  Update the column sort indicator.
- void RemoveSortIndicator ()

  Remove the sort indicator from the column being used as sort key.

Protected Member Functions

- virtual int GetBestFittingWidth (unsigned int idx) const

  This function can be overridden in the classes deriving from this control instead of overriding UpdateColumnWidthToFit().

Additional Inherited Members

21.345.2 Constructor & Destructor Documentation

wxHeaderCtrlSimple::wxHeaderCtrlSimple ( )

Default constructor not creating the underlying window.

You must use Create() after creating the object using this constructor.

wxHeaderCtrlSimple::wxHeaderCtrlSimple (wxWindow * *parent*, wxWindowID *winid* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHD_DEFAULT_STYLE, const wxString & *name* = `wxHeaderCtrlNameStr`)

Constructor creating the window.

Please see the base class wxHeaderCtrl::Create() method for the parameters description.

21.345.3 Member Function Documentation

void wxHeaderCtrlSimple::AppendColumn (const wxHeaderColumnSimple & *col*)

Append the column to the end of the control.

See also

InsertColumn()

void wxHeaderCtrlSimple::DeleteColumn (unsigned int *idx*)

Delete the column at the given position.

See also

InsertColumn(), AppendColumn()

virtual int wxHeaderCtrlSimple::GetBestFittingWidth (unsigned int *idx*) const `[protected], [virtual]`

This function can be overridden in the classes deriving from this control instead of overriding UpdateColumnWidthToFit().

To implement automatic column resizing to fit its contents width when the column divider is double clicked, you need to simply return the fitting width for the given column *idx* from this method, the control will automatically use the biggest value between the one returned from here and the one needed for the display of the column title itself.

The base class version returns -1 indicating that this function is not implemented.

void wxHeaderCtrlSimple::HideColumn (unsigned int *idx*)

Hide the column with the given index.

This is the same as calling

```
ShowColumn(idx, false)
```

.

Parameters

idx	The index of the column to show or hide, from 0 to GetColumnCount().

void wxHeaderCtrlSimple::InsertColumn (const wxHeaderColumnSimple & *col*, unsigned int *idx*)

Insert the column at the given position.

Parameters

col	The column to insert. Notice that because of the existence of implicit conversion from wxString to wxHeaderColumn a string can be passed directly here.
idx	The position of the new column, from 0 to GetColumnCount(). Using GetColumnCount() means to append the column to the end.

See also

AppendColumn()

void wxHeaderCtrlSimple::RemoveSortIndicator ( )

Remove the sort indicator from the column being used as sort key.

See also

ShowSortIndicator

void wxHeaderCtrlSimple::ShowColumn (unsigned int *idx,* bool *show* = `true`)

Show or hide the column.

Initially the column is shown by default or hidden if it was added with wxCOL_HIDDEN flag set.

When a column is hidden, it doesn't appear at all on the screen but its index is still taken into account when working with other columns. E.g. if there are three columns 0, 1 and 2 and the column 1 is hidden you still need to use index 2 to refer to the last visible column.

Parameters

idx	The index of the column to show or hide, from 0 to GetColumnCount().
show	Indicates whether the column should be shown (default) or hidden.

void wxHeaderCtrlSimple::ShowSortIndicator (unsigned int *idx,* bool *sortOrder* = `true`)

Update the column sort indicator.

The sort indicator, if shown, is typically an arrow pointing upwards or downwards depending on whether the control contents is sorted in ascending or descending order.

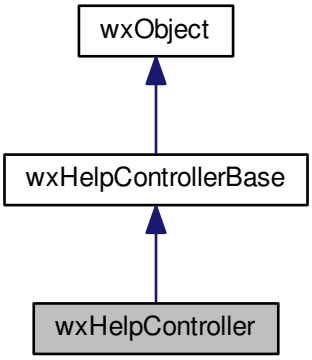
Parameters

idx	The column to set the sort indicator for. If `-1` is given, then the currently shown sort indicator will be removed.
sortOrder	If true or false show the sort indicator corresponding to ascending or descending sort order respectively.

21.346 wxHelpController Class Reference

`#include <wx/help.h>`

Inheritance diagram for wxHelpController:



21.346.1 Detailed Description

This is an alias for one of a family of help controller classes which is most appropriate for the current platform.

A help controller allows an application to display help, at the contents or at a particular topic, and shut the help program down on termination. This avoids proliferation of many instances of the help viewer whenever the user requests a different topic via the application's menus or buttons.

Typically, an application will create a help controller instance when it starts, and immediately call wxHelpController::Initialize to associate a filename with it. The help viewer will only get run, however, just before the first call to display something.

Most help controller classes actually derive from wxHelpControllerBase and have names of the form wxXXXHelpController or wxHelpControllerXXX. An appropriate class is aliased to the name wxHelpController for each platform, as follows:

- On desktop Windows, wxCHMHelpController is used (MS HTML Help).
- On Windows CE, wxWinceHelpController is used.
- On all other platforms, wxHtmlHelpController is used if wxHTML is compiled into wxWidgets; otherwise wxExtHelpController is used (for invoking an external browser).

The remaining help controller classes need to be named explicitly by an application that wishes to make use of them.

The following help controller classes are defined:

- wxWinHelpController, for controlling Windows Help.
- wxCHMHelpController, for controlling MS HTML Help. To use this, you need to set wxUSE_MS_HTML_HELP to 1 in setup.h and have the htmlhelp.h header from Microsoft's HTML Help kit. (You don't need the VC++-specific htmlhelp.lib because wxWidgets loads necessary DLL at runtime and so it works with all compilers.)
- wxBestHelpController, for controlling MS HTML Help or, if Microsoft's runtime is not available, wxHtmlHelpController. You need to provide **both** CHM and HTB versions of the help file. For wxMSW only.
- wxExtHelpController, for controlling external browsers under Unix. The default browser is Netscape Navigator. The 'help' sample shows its use.
- wxWinceHelpController, for controlling a simple .htm help controller for Windows CE applications.
- wxHtmlHelpController, a sophisticated help controller using wxHTML, in a similar style to the Microsoft HTML Help viewer and using some of the same files. Although it has an API compatible with other help controllers, it has more advanced features, so it is recommended that you use the specific API for this class instead. Note that if you use .zip or .htb formats for your books, you must add this line to your application initialization:

  ```
  wxFileSystem::AddHandler(new wxArchiveFSHandler);
  ```

  or nothing will be shown in your help window.

Library: wxCore

Category: Help

See also

wxHtmlHelpController, wxHTML Overview

Public Member Functions

- wxHelpController (wxWindow *parentWindow=NULL)

  Constructs a help instance object, but does not invoke the help viewer.

Additional Inherited Members

21.346.2 Constructor & Destructor Documentation

wxHelpController::wxHelpController (wxWindow * *parentWindow* = NULL)

Constructs a help instance object, but does not invoke the help viewer.

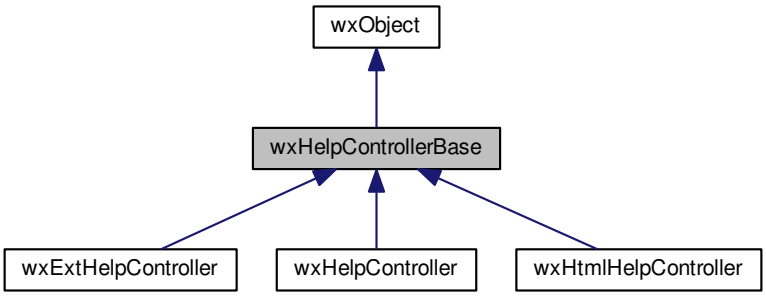
If you provide a window, it will be used by some help controller classes, such as wxCHMHelpController, wxWinHelpController and wxHtmlHelpController, as the parent for the help window instead of the value of wxApp::GetTopWindow.

You can also change the parent window later with SetParentWindow().

21.347 wxHelpControllerBase Class Reference

`#include <wx/help.h>`

Inheritance diagram for wxHelpControllerBase:



21.347.1 Detailed Description

This is the abstract base class a family of classes by which applications may invoke a help viewer to provide on-line help.

A help controller allows an application to display help, at the contents or at a particular topic, and shut the help program down on termination. This avoids proliferation of many instances of the help viewer whenever the user requests a different topic via the application's menus or buttons.

Typically, an application will create a help controller instance when it starts, and immediately call wxHelpController::Initialize to associate a filename with it. The help viewer will only get run, however, just before the first call to display something.

Library: wxCore

Category: Help

See also

wxHelpController, wxHtmlHelpController, wxHTML Overview

Public Member Functions

- wxHelpControllerBase (wxWindow *parentWindow=NULL)

  Constructs a help instance object, but does not invoke the help viewer.
- ~wxHelpControllerBase ()

  Destroys the help instance, closing down the viewer if it is running.
- virtual bool DisplayBlock (long blockNo)=0

  If the help viewer is not running, runs it and displays the file at the given block number.
- virtual bool DisplayContents ()=0

  If the help viewer is not running, runs it and displays the contents.
- virtual bool DisplayContextPopup (int contextId)

  Displays the section as a popup window using a context id.
- virtual bool DisplaySection (const wxString &section)

  If the help viewer is not running, runs it and displays the given section.
- virtual bool DisplaySection (int sectionNo)=0

  If the help viewer is not running, runs it and displays the given section.
- virtual bool DisplayTextPopup (const wxString &text, const wxPoint &pos)

  Displays the text in a popup window, if possible.
- virtual wxFrame * GetFrameParameters (wxSize *size=NULL, wxPoint *pos=NULL, bool *newFrameEachTime=NULL)

  For wxHtmlHelpController, returns the latest frame size and position settings and whether a new frame is drawn with each invocation.
- virtual wxWindow * GetParentWindow () const

  Returns the window to be used as the parent for the help window.
- virtual bool KeywordSearch (const wxString &keyWord, wxHelpSearchMode mode=wxHELP_SEARCH_ALL)=0

  If the help viewer is not running, runs it, and searches for sections matching the given keyword.
- virtual bool LoadFile (const wxString &file=wxEmptyString)=0

  If the help viewer is not running, runs it and loads the given file.
- virtual void OnQuit ()

  Overridable member called when this application's viewer is quit by the user.
- virtual bool Quit ()=0

  If the viewer is running, quits it by disconnecting.
- virtual void SetFrameParameters (const wxString &titleFormat, const wxSize &size, const wxPoint &pos=wxDefaultPosition, bool newFrameEachTime=false)

  Set the parameters of the frame window.
- virtual void SetParentWindow (wxWindow *parentWindow)

  Sets the window to be used as the parent for the help window.
- virtual void SetViewer (const wxString &viewer, long flags=wxHELP_NETSCAPE)

  Sets detailed viewer information.

- virtual bool Initialize (const wxString &file)

  Initializes the help instance with a help filename, and optionally a server socket number if using wxHelp (now obsolete).
- virtual bool Initialize (const wxString &file, int server)

  Initializes the help instance with a help filename, and optionally a server socket number if using wxHelp (now obsolete).

Additional Inherited Members

21.347.2 Constructor & Destructor Documentation

wxHelpControllerBase::wxHelpControllerBase (wxWindow * *parentWindow* = NULL)

Constructs a help instance object, but does not invoke the help viewer.

If you provide a window, it will be used by some help controller classes, such as wxCHMHelpController, wxWinHelpController and wxHtmlHelpController, as the parent for the help window instead of the value of wxApp::GetTopWindow.

You can also change the parent window later with SetParentWindow().

wxHelpControllerBase::~wxHelpControllerBase ( )

Destroys the help instance, closing down the viewer if it is running.

21.347.3 Member Function Documentation

virtual bool wxHelpControllerBase::DisplayBlock (long *blockNo*) [pure virtual]

If the help viewer is not running, runs it and displays the file at the given block number.

- *WinHelp:* Refers to the context number.
- *MS* HTML Help: Refers to the context number.
- *External* HTML help: the same as for DisplaySection().
- *wxHtmlHelpController: sectionNo* is an identifier as specified in the .hhc file. See Help Files Format.

Deprecated This function is for backward compatibility only, and applications should use DisplaySection() instead.

Implemented in wxExtHelpController.

virtual bool wxHelpControllerBase::DisplayContents ( ) [pure virtual]

If the help viewer is not running, runs it and displays the contents.

Implemented in wxHtmlHelpController, and wxExtHelpController.

virtual bool wxHelpControllerBase::DisplayContextPopup (int *contextId*) [virtual]

Displays the section as a popup window using a context id.

Returns false if unsuccessful or not implemented.

virtual bool wxHelpControllerBase::DisplaySection (const wxString & *section*) [virtual]

If the help viewer is not running, runs it and displays the given section.

The interpretation of section differs between help viewers. For most viewers, this call is equivalent to KeywordSearch. For MS HTML Help, wxHTML help and external HTML help, if section has a .htm or .html extension, that HTML file will be displayed; otherwise a keyword search is done.

Reimplemented in wxExtHelpController.

virtual bool wxHelpControllerBase::DisplaySection (int *sectionNo*) `[pure virtual]`

If the help viewer is not running, runs it and displays the given section.

- *WinHelp*, MS HTML Help *sectionNo* is a context id.
- *External* HTML help: wxExtHelpController implements *sectionNo* as an id in a map file, which is of the form:
- *wxHtmlHelpController: sectionNo* is an identifier as specified in the .hhc file. See Help Files Format. See also the help sample for notes on how to specify section numbers for various help file formats.

Implemented in wxExtHelpController.

virtual bool wxHelpControllerBase::DisplayTextPopup (const wxString & *text*, const wxPoint & *pos*) `[virtual]`

Displays the text in a popup window, if possible.

Returns false if unsuccessful or not implemented.

virtual wxFrame* wxHelpControllerBase::GetFrameParameters (wxSize * *size* = NULL, wxPoint * *pos* = NULL, bool * *newFrameEachTime* = NULL) `[virtual]`

For wxHtmlHelpController, returns the latest frame size and position settings and whether a new frame is drawn with each invocation.

For all other help controllers, this function does nothing and just returns NULL.

Parameters

size	The most recent frame size.
pos	The most recent frame position.
newFrameEachTime	true if a new frame is drawn with each invocation.

Reimplemented in wxExtHelpController.

virtual wxWindow* wxHelpControllerBase::GetParentWindow ( ) const `[virtual]`

Returns the window to be used as the parent for the help window.

This window is used by wxCHMHelpController, wxWinHelpController and wxHtmlHelpController.

virtual bool wxHelpControllerBase::Initialize (const wxString & *file*) `[virtual]`

Initializes the help instance with a help filename, and optionally a server socket number if using wxHelp (now obsolete).

Does not invoke the help viewer. This must be called directly after the help instance object is created and before any attempts to communicate with the viewer.

You may omit the file extension and a suitable one will be chosen. For wxHtmlHelpController, the extensions zip, htb and hhp will be appended while searching for a suitable file. For WinHelp, the hlp extension is appended.

Reimplemented in wxExtHelpController.

virtual bool wxHelpControllerBase::Initialize (const wxString & *file*, int *server*) `[virtual]`

Initializes the help instance with a help filename, and optionally a server socket number if using wxHelp (now obsolete).

Does not invoke the help viewer. This must be called directly after the help instance object is created and before any attempts to communicate with the viewer.

You may omit the file extension and a suitable one will be chosen. For wxHtmlHelpController, the extensions zip, htb and hhp will be appended while searching for a suitable file. For WinHelp, the hlp extension is appended.

virtual bool wxHelpControllerBase::KeywordSearch (const wxString & *keyWord,* wxHelpSearchMode *mode* = wxHELP_SEARCH_ALL) [pure virtual]

If the help viewer is not running, runs it, and searches for sections matching the given keyword.

If one match is found, the file is displayed at this section. The optional parameter allows to search the index (wxH↩ ELP_SEARCH_INDEX) but this currently is only supported by the wxHtmlHelpController.

- *WinHelp*, MS HTML Help: If more than one match is found, the first topic is displayed.
- *External* HTML help, simple wxHTML help: If more than one match is found, a choice of topics is displayed.
- *wxHtmlHelpController:* see wxHtmlHelpController::KeywordSearch.

Implemented in wxHtmlHelpController, and wxExtHelpController.

virtual bool wxHelpControllerBase::LoadFile (const wxString & *file* = wxEmptyString) [pure virtual]

If the help viewer is not running, runs it and loads the given file.

If the filename is not supplied or is empty, the file specified in Initialize() is used.

If the viewer is already displaying the specified file, it will not be reloaded. This member function may be used before each display call in case the user has opened another file.

wxHtmlHelpController ignores this call.

Implemented in wxExtHelpController.

virtual void wxHelpControllerBase::OnQuit ( ) [virtual]

Overridable member called when this application's viewer is quit by the user.

This does not work for all help controllers.

Reimplemented in wxExtHelpController.

virtual bool wxHelpControllerBase::Quit ( ) [pure virtual]

If the viewer is running, quits it by disconnecting.

For Windows Help, the viewer will only close if no other application is using it.

Implemented in wxExtHelpController.

virtual void wxHelpControllerBase::SetFrameParameters (const wxString & *titleFormat,* const wxSize & *size,* const wxPoint & *pos* = wxDefaultPosition, bool *newFrameEachTime* = false) [virtual]

Set the parameters of the frame window.

For wxHtmlHelpController, *titleFormat* specifies the title string format (with `%s` being replaced by the actual page title) and *size* and *position* specify the geometry of the frame.

For all other help controllers this function has no effect.

Finally, *newFrameEachTime* is always ignored currently.

Reimplemented in wxExtHelpController.

virtual void wxHelpControllerBase::SetParentWindow (wxWindow * *parentWindow*) [virtual]

Sets the window to be used as the parent for the help window.

This is used by wxCHMHelpController, wxWinHelpController and wxHtmlHelpController.

virtual void wxHelpControllerBase::SetViewer (const wxString & *viewer,* long *flags* = wxHELP_NETSCAPE) [virtual]

Sets detailed viewer information.

So far this is only relevant to wxExtHelpController. Some examples of usage:

```
m_help.SetViewer("kdehelp");
m_help.SetViewer("gnome-help-browser");
m_help.SetViewer("netscape", wxHELP_NETSCAPE);
```

Parameters

viewer	This defaults to "netscape" for wxExtHelpController.
flags	This defaults to wxHELP_NETSCAPE for wxExtHelpController, indicating that the viewer is a variant of Netscape Navigator.

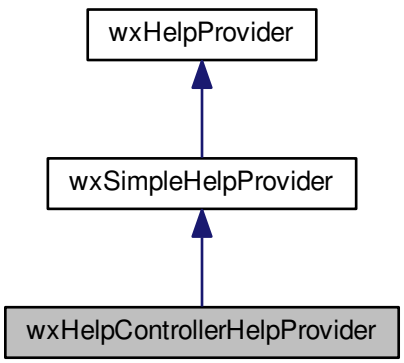
Todo modernize this function with wxLaunchDefaultBrowser

Reimplemented in wxExtHelpController.

21.348 wxHelpControllerHelpProvider Class Reference

`#include <wx/cshelp.h>`

Inheritance diagram for wxHelpControllerHelpProvider:



21.348.1 Detailed Description

wxHelpControllerHelpProvider is an implementation of wxHelpProvider which supports both context identifiers and plain text help strings.

If the help text is an integer, it is passed to wxHelpController::DisplayContextPopup(). Otherwise, it shows the string in a tooltip as per wxSimpleHelpProvider. If you use this with a wxCHMHelpController instance on windows, it will use the native style of tip window instead of wxTipWindow.

You can use the convenience function wxContextId() to convert an integer context id to a string for passing to wxWindow::SetHelpText().

Library: wxCore

Category: Help

See also

wxHelpProvider, wxSimpleHelpProvider, wxContextHelp, wxWindow::SetHelpText(), wxWindow::GetHelpTextAtPoint()

Public Member Functions

- wxHelpControllerHelpProvider (wxHelpControllerBase *hc=NULL)

  Note that the instance doesn't own the help controller.
- wxHelpControllerBase * GetHelpController () const

  Returns the help controller associated with this help provider.
- void SetHelpController (wxHelpControllerBase *hc)

  Sets the help controller associated with this help provider.

Additional Inherited Members

21.348.2 Constructor & Destructor Documentation

wxHelpControllerHelpProvider::wxHelpControllerHelpProvider (wxHelpControllerBase * *hc* = NULL)

Note that the instance doesn't own the help controller.

The help controller should be deleted separately.

21.348.3 Member Function Documentation

wxHelpControllerBase* wxHelpControllerHelpProvider::GetHelpController ( ) const

Returns the help controller associated with this help provider.

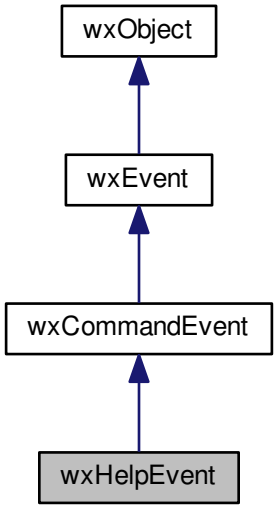
void wxHelpControllerHelpProvider::SetHelpController (wxHelpControllerBase * *hc*)

Sets the help controller associated with this help provider.

21.349 wxHelpEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxHelpEvent:



21.349.1 Detailed Description

A help event is sent when the user has requested context-sensitive help.

This can either be caused by the application requesting context-sensitive help mode via wxContextHelp, or (on MS Windows) by the system generating a WM_HELP message when the user pressed F1 or clicked on the query button in a dialog caption.

A help event is sent to the window that the user clicked on, and is propagated up the window hierarchy until the event is processed or there are no more event handlers.

The application should call wxEvent::GetId to check the identity of the clicked-on window, and then either show some suitable help or call wxEvent::Skip() if the identifier is unrecognised.

Calling Skip is important because it allows wxWidgets to generate further events for ancestors of the clicked-on window. Otherwise it would be impossible to show help for container windows, since processing would stop after the first window found.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHelpEvent& event)

Event macros:

- EVT_HELP(id, func): Process a `wxEVT_HELP` event.
- EVT_HELP_RANGE(id1, id2, func): Process a `wxEVT_HELP` event for a range of ids.

Library: wxCore

Category: Events

See also

wxContextHelp, wxDialog, Events and Event Handling

Public Types

- enum Origin {
  Origin_Unknown,
  Origin_Keyboard,
  Origin_HelpButton }

  Indicates how a wxHelpEvent was generated.

Public Member Functions

- wxHelpEvent (wxEventType type=wxEVT_NULL, wxWindowID winid=0, const wxPoint &pt=wxDefaultPosition, wxHelpEvent::Origin origin=Origin_Unknown)

  Constructor.
- wxHelpEvent::Origin GetOrigin () const

  Returns the origin of the help event which is one of the wxHelpEvent::Origin values.
- const wxPoint & GetPosition () const

  Returns the left-click position of the mouse, in screen coordinates.
- void SetOrigin (wxHelpEvent::Origin origin)

  Set the help event origin, only used internally by wxWidgets normally.
- void SetPosition (const wxPoint &pt)

  Sets the left-click position of the mouse, in screen coordinates.

Additional Inherited Members

21.349.2 Member Enumeration Documentation

enum wxHelpEvent::Origin

Indicates how a wxHelpEvent was generated.

Enumerator

Origin_Unknown unrecognized event source.

Origin_Keyboard event generated from F1 key press.

Origin_HelpButton event generated by wxContextHelp or from the [?] button on the title bar (Windows).

21.349.3 Constructor & Destructor Documentation

wxHelpEvent::wxHelpEvent (wxEventType *type* = wxEVT_NULL, wxWindowID *winid* = 0, const wxPoint & *pt* = wxDefaultPosition, wxHelpEvent::Origin *origin* = Origin_Unknown)

Constructor.

21.349.4 Member Function Documentation

wxHelpEvent::Origin wxHelpEvent::GetOrigin ( ) const

Returns the origin of the help event which is one of the wxHelpEvent::Origin values.

The application may handle events generated using the keyboard or mouse differently, e.g. by using wxGetMouse↩ Position() for the mouse events.

See also

SetOrigin()

const wxPoint& wxHelpEvent::GetPosition ( ) const

Returns the left-click position of the mouse, in screen coordinates.

This allows the application to position the help appropriately.

void wxHelpEvent::SetOrigin (wxHelpEvent::Origin *origin*)

Set the help event origin, only used internally by wxWidgets normally.

See also

GetOrigin()

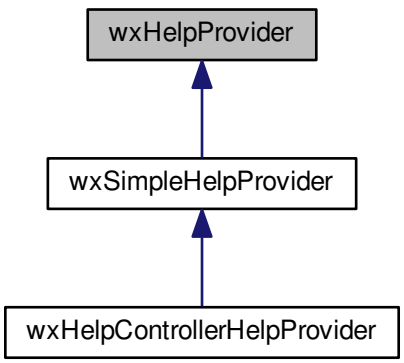
void wxHelpEvent::SetPosition (const wxPoint & *pt*)

Sets the left-click position of the mouse, in screen coordinates.

21.350 wxHelpProvider Class Reference

`#include <wx/cshelp.h>`

Inheritance diagram for wxHelpProvider:



21.350.1 Detailed Description

wxHelpProvider is an abstract class used by a program implementing context-sensitive help to show the help text for the given window.

The current help provider must be explicitly set by the application using Set().

Library: wxCore

Category: Help

See also

wxContextHelp, wxContextHelpButton, wxSimpleHelpProvider, wxHelpControllerHelpProvider, wxWindow::SetHelpText(), wxWindow::GetHelpTextAtPoint()

Public Member Functions

- virtual ~wxHelpProvider ()

  Virtual destructor for any base class.
- virtual void AddHelp (wxWindowBase *window, const wxString &text)

  Associates the text with the given window.
- virtual void AddHelp (wxWindowID id, const wxString &text)

  Associates the text with the given ID.
- virtual wxString GetHelp (const wxWindowBase *window)=0

  This version associates the given text with all windows with this id.
- virtual void RemoveHelp (wxWindowBase *window)

  Removes the association between the window pointer and the help text.
- virtual bool ShowHelp (wxWindowBase *window)

  Shows help for the given window.
- virtual bool ShowHelpAtPoint (wxWindowBase *window, const wxPoint &point, wxHelpEvent::Origin origin)

  This function may be overridden to show help for the window when it should depend on the position inside the window, By default this method forwards to ShowHelp(), so it is enough to only implement the latter if the help doesn't depend on the position.

Static Public Member Functions

- static wxHelpProvider * Get ()

  Returns pointer to help provider instance.
- static wxHelpProvider * Set (wxHelpProvider *helpProvider)

  Set the current, application-wide help provider.

21.350.2 Constructor & Destructor Documentation

virtual wxHelpProvider::~wxHelpProvider ( ) [virtual]

Virtual destructor for any base class.

21.350.3 Member Function Documentation

virtual void wxHelpProvider::AddHelp (wxWindowBase * *window,* const wxString & *text*) `[virtual]`

Associates the text with the given window.

Remarks

Although all help providers have these functions to allow making wxWindow::SetHelpText() work, not all of them implement the functions.

virtual void wxHelpProvider::AddHelp (wxWindowID *id,* const wxString & *text*) `[virtual]`

Associates the text with the given ID.

This help text will be shown for all windows with ID *id*, unless they have more specific help text associated using the other AddHelp() prototype. May be used to set the same help string for all Cancel buttons in the application, for example.

Remarks

Although all help providers have these functions to allow making wxWindow::SetHelpText() work, not all of them implement the functions.

static wxHelpProvider* wxHelpProvider::Get ( ) `[static]`

Returns pointer to help provider instance.

Unlike some other classes, the help provider is not created on demand. This must be explicitly done by the application using Set().

virtual wxString wxHelpProvider::GetHelp (const wxWindowBase * *window*) `[pure virtual]`

This version associates the given text with all windows with this id.

May be used to set the same help string for all Cancel buttons in the application, for example.

virtual void wxHelpProvider::RemoveHelp (wxWindowBase * *window*) `[virtual]`

Removes the association between the window pointer and the help text.

This is called by the wxWindow destructor. Without this, the table of help strings will fill up and when window pointers are reused, the wrong help string will be found.

static wxHelpProvider* wxHelpProvider::Set (wxHelpProvider * *helpProvider*) `[static]`

Set the current, application-wide help provider.

Returns

Pointer to previous help provider or NULL if there wasn't any.

virtual bool wxHelpProvider::ShowHelp (wxWindowBase * *window*) `[virtual]`

Shows help for the given window.

Override this function if the help doesn't depend on the exact position inside the window, otherwise you need to override ShowHelpAtPoint(). Returns true if help was shown, or false if no help was available for this window.

virtual bool wxHelpProvider::ShowHelpAtPoint (wxWindowBase * *window,* const wxPoint & *point,* wxHelpEvent::Origin *origin*) `[virtual]`

This function may be overridden to show help for the window when it should depend on the position inside the window, By default this method forwards to ShowHelp(), so it is enough to only implement the latter if the help doesn't depend on the position.

Parameters

window	Window to show help text for.
point	Coordinates of the mouse at the moment of help event emission.
origin	Help event origin, see wxHelpEvent::GetOrigin.

Returns

true if help was shown, or false if no help was available for this window.

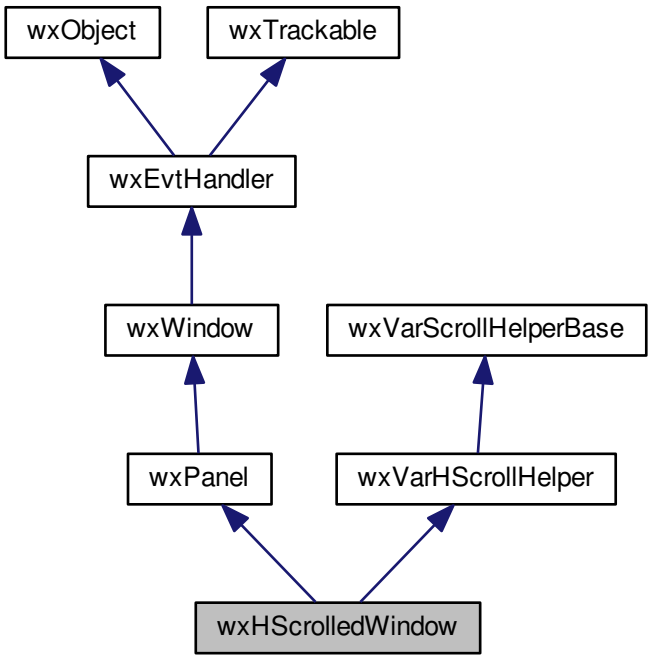
Since

2.7.0

21.351 wxHScrolledWindow Class Reference

`#include <wx/vscroll.h>`

Inheritance diagram for wxHScrolledWindow:



21.351.1 Detailed Description

In the name of this class, "H" stands for "horizontal" because it can be used for scrolling columns of variable widths.

It is not necessary to know the widths of all columns in advance – only those which are shown on the screen need to be measured.

In any case, this is a generalization of wxScrolled which can be only used when all columns have the same widths. It lacks some other wxScrolled features however, notably it can't scroll specific pixel sizes of the window or its exact client area size.

To use this class, you need to derive from it and implement the OnGetColumnWidth() pure virtual method. You also must call SetColumnCount() to let the base class know how many columns it should display, but from that moment on the scrolling is handled entirely by wxHScrolledWindow. You only need to draw the visible part of contents in your `OnPaint()` method as usual. You should use GetVisibleColumnsBegin() and GetVisibleColumnsEnd() to select the lines to display. Note that the device context origin is not shifted so the first visible column always appears at the point (0, 0) in physical as well as logical coordinates.

Library: wxCore

Category: Miscellaneous Windows

See also

wxHVScrolledWindow, wxVScrolledWindow

Public Member Functions

- wxHScrolledWindow ()

  Default constructor, you must call Create() later.
- wxHScrolledWindow (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  This is the normal constructor, no need to call Create() after using this constructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  Same as the non-default constructor, but returns a status code: true if ok, false if the window couldn't be created.

Additional Inherited Members

21.351.2 Constructor & Destructor Documentation

wxHScrolledWindow::wxHScrolledWindow ( )

Default constructor, you must call Create() later.

wxHScrolledWindow::wxHScrolledWindow (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxPanelNameStr`)

This is the normal constructor, no need to call Create() after using this constructor.

Note

`wxHSCROLL` is always automatically added to the style, there is no need to specify it explicitly.

Parameters

parent	The parent window, must not be NULL.
id	The identifier of this window, wxID_ANY by default.
pos	The initial window position.
size	The initial window size.
style	The window style. There are no special style bits defined for this class.
name	The name for this window; usually not used.

21.351.3 Member Function Documentation

bool wxHScrolledWindow::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxPanelNameStr)

Same as the non-default constructor, but returns a status code: true if ok, false if the window couldn't be created.

Just as with the constructor, the wxHSCROLL style is always used, there is no need to specify it explicitly.

21.352 wxHtmlBookRecord Class Reference

`#include <wx/html/helpdata.h>`

21.352.1 Detailed Description

Helper class for wxHtmlHelpData.

Public Member Functions

- wxHtmlBookRecord (const wxString &bookfile, const wxString &basepath, const wxString &title, const wxString &start)
- wxString GetBookFile () const
- wxString GetTitle () const
- wxString GetStart () const
- wxString GetBasePath () const
- void SetContentsRange (int start, int end)
- int GetContentsStart () const
- int GetContentsEnd () const
- void SetTitle (const wxString &title)
- void SetBasePath (const wxString &path)
- void SetStart (const wxString &start)
- wxString GetFullPath (const wxString &page) const

21.352.2 Constructor & Destructor Documentation

wxHtmlBookRecord::wxHtmlBookRecord (const wxString & *bookfile*, const wxString & *basepath*, const wxString & *title*, const wxString & *start*)

21.352.3 Member Function Documentation

wxString wxHtmlBookRecord::GetBasePath ( ) const

wxString wxHtmlBookRecord::GetBookFile ( ) const

int wxHtmlBookRecord::GetContentsEnd ( ) const

int wxHtmlBookRecord::GetContentsStart ( ) const

wxString wxHtmlBookRecord::GetFullPath (const wxString & *page*) const

wxString wxHtmlBookRecord::GetStart ( ) const

wxString wxHtmlBookRecord::GetTitle ( ) const

void wxHtmlBookRecord::SetBasePath (const wxString & *path*)

void wxHtmlBookRecord::SetContentsRange (int *start,* int *end*)

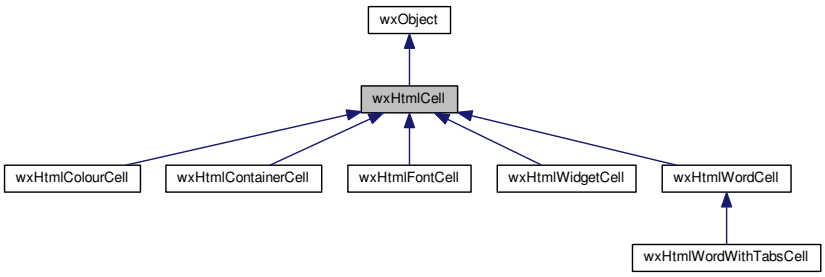
void wxHtmlBookRecord::SetStart (const wxString & *start*)

void wxHtmlBookRecord::SetTitle (const wxString & *title*)

21.353 wxHtmlCell Class Reference

`#include <wx/html/htmlcell.h>`

Inheritance diagram for wxHtmlCell:



21.353.1 Detailed Description

Internal data structure.

It represents fragments of parsed HTML page, the so-called **cell** - a word, picture, table, horizontal line and so on. It is used by wxHtmlWindow and wxHtmlWinParser to represent HTML page in memory.

You can divide cells into two groups : *visible* cells with non-zero width and height and *helper* cells (usually with zero width and height) that perform special actions such as color or font change.

Library: wxHTML

Category: HTML

See also

Cells and Containers, wxHtmlContainerCell

Public Member Functions

- wxHtmlCell ()

  Constructor.
- virtual bool AdjustPagebreak (int *pagebreak, const wxArrayInt &known_pagebreaks, int pageHeight) const

  This method is used to adjust pagebreak position.
- virtual void Draw (wxDC &dc, int x, int y, int view_y1, int view_y2, wxHtmlRenderingInfo &info)

  Renders the cell.
- virtual void DrawInvisible (wxDC &dc, int x, int y, wxHtmlRenderingInfo &info)

  This method is called instead of Draw() when the cell is certainly out of the screen (and thus invisible).
- virtual const wxHtmlCell * Find (int condition, const void *param) const

  Returns pointer to itself if this cell matches condition (or if any of the cells following in the list matches), NULL otherwise.
- int GetDescent () const

  Returns descent value of the cell (m_Descent member).
- virtual wxHtmlCell * GetFirstChild () const

  Returns pointer to the first cell in the list.
- int GetHeight () const

  Returns height of the cell (m_Height member).
- const wxString & GetId () const

  Returns unique cell identifier if there is any, the empty string otherwise.
- virtual wxHtmlLinkInfo * GetLink (int x=0, int y=0) const

  Returns hypertext link if associated with this cell or NULL otherwise.
- virtual wxCursor GetMouseCursor (wxHtmlWindowInterface *window) const

  Returns cursor to show when mouse pointer is over the cell.
- virtual wxCursor GetMouseCursorAt (wxHtmlWindowInterface *window, const wxPoint &rePos) const

  Returns cursor to show when mouse pointer is over the specified point.
- wxHtmlCell * GetNext () const

  Returns pointer to the next cell in list (see htmlcell.h if you're interested in details).
- wxHtmlContainerCell * GetParent () const

  Returns pointer to parent container.
- int GetPosX () const

  Returns X position within parent (the value is relative to parent's upper left corner).
- int GetPosY () const

  Returns Y position within parent (the value is relative to parent's upper left corner).
- int GetWidth () const

  Returns width of the cell (m_Width member).
- virtual void Layout (int w)

  Layouts the cell.
- virtual bool ProcessMouseClick (wxHtmlWindowInterface *window, const wxPoint &pos, const wxMouseEvent &event)

  This function is simple event handler.
- void SetId (const wxString &id)

  Sets unique cell identifier.
- void SetLink (const wxHtmlLinkInfo &link)

  Sets the hypertext link associated with this cell.
- void SetNext (wxHtmlCell *cell)

  Sets the next cell in the list.

- void SetParent (wxHtmlContainerCell *p)

  Sets parent container of this cell.
- virtual void SetPos (int x, int y)

  Sets the cell's position within parent container.

Additional Inherited Members

21.353.2 Constructor & Destructor Documentation

wxHtmlCell::wxHtmlCell ( )

Constructor.

21.353.3 Member Function Documentation

virtual bool wxHtmlCell::AdjustPagebreak (int * *pagebreak,* const wxArrayInt & *known_pagebreaks,* int *pageHeight*) const `[virtual]`

This method is used to adjust pagebreak position.

The first parameter is a variable that contains the y-coordinate of the page break (= horizontal line that should not be crossed by words, images etc.). If this cell cannot be divided into two pieces (each one on another page) then it either moves the pagebreak a few pixels up, if possible, or, if the cell cannot fit on the page at all, then the cell is forced to split unconditionally.

Returns true if pagebreak was modified, false otherwise.

Parameters

pagebreak	position in pixel of the pagebreak.
known_↩ pagebreaks	the list of the previous pagebreaks
pageHeight	the height in pixel of the page drawable area

Usage:

```
while (container->AdjustPagebreak(&p, kp, ph)) {}
```

virtual void wxHtmlCell::Draw (wxDC & *dc,* int *x,* int *y,* int *view_y1,* int *view_y2,* wxHtmlRenderingInfo & *info*) `[virtual]`

Renders the cell.

Parameters

dc	Device context to which the cell is to be drawn.
x,y	Coordinates of parent's upper left corner (origin). You must add this to m_PosX,m_PosY when passing coordinates to dc's methods Example: `dc->DrawText("hello", x + m_PosX, y + m_PosY)`

view_y1	y-coord of the first line visible in window. This is used to optimize rendering speed.
view_y2	y-coord of the last line visible in window. This is used to optimize rendering speed.
info	Additional information for the rendering of the cell.

virtual void wxHtmlCell::DrawInvisible (wxDC & *dc*, int *x*, int *y*, wxHtmlRenderingInfo & *info*) [virtual]

This method is called instead of Draw() when the cell is certainly out of the screen (and thus invisible).

This is not nonsense - some tags (like wxHtmlColourCell or font setter) must be drawn even if they are invisible!

Parameters

dc	Device context to which the cell is to be drawn.
x,y	Coordinates of parent's upper left corner. You must add this to m_PosX,m_PosY when passing coordinates to dc's methods Example: `dc->DrawText("hello", x + m_PosX, y + m_PosY)`
info	Additional information for the rendering of the cell.

virtual const wxHtmlCell* wxHtmlCell::Find (int *condition*, const void * *param*) const [virtual]

Returns pointer to itself if this cell matches condition (or if any of the cells following in the list matches), NULL otherwise.

(In other words if you call top-level container's Find() it will return pointer to the first cell that matches the condition)

It is recommended way how to obtain pointer to particular cell or to cell of some type (e.g. wxHtmlAnchorCell reacts on wxHTML_COND_ISANCHOR condition).

Parameters

condition	Unique integer identifier of condition
param	Optional parameters

int wxHtmlCell::GetDescent ( ) const

Returns descent value of the cell (m_Descent member).

See explanation:

virtual wxHtmlCell* wxHtmlCell::GetFirstChild ( ) const [virtual]

Returns pointer to the first cell in the list.

You can then use child's GetNext() method to obtain pointer to the next cell in list.

Note

This shouldn't be used by the end user. If you need some way of finding particular cell in the list, try Find() method instead.

int wxHtmlCell::GetHeight ( ) const

Returns height of the cell (m_Height member).

const wxString& wxHtmlCell::GetId ( ) const

Returns unique cell identifier if there is any, the empty string otherwise.

virtual wxHtmlLinkInfo* wxHtmlCell::GetLink (int *x* = 0, int *y* = 0) const [virtual]

Returns hypertext link if associated with this cell or NULL otherwise.

See wxHtmlLinkInfo. (Note: this makes sense only for visible tags).

Parameters

x,y	Coordinates of position where the user pressed mouse button. These coordinates are used e.g. by COLORMAP. Values are relative to the upper left corner of THIS cell (i.e. from 0 to m_Width or m_Height)

virtual wxCursor wxHtmlCell::GetMouseCursor (wxHtmlWindowInterface * *window*) const [virtual]

Returns cursor to show when mouse pointer is over the cell.

Parameters

window	interface to the parent HTML window

See also

GetMouseCursorAt()

virtual wxCursor wxHtmlCell::GetMouseCursorAt (wxHtmlWindowInterface * *window*, const wxPoint & *rePos*) const [virtual]

Returns cursor to show when mouse pointer is over the specified point.

This function should be overridden instead of GetMouseCursorAt() if the cursor should depend on the exact position of the mouse in the window.

Parameters

window	interface to the parent HTML window
rePos	Position to show cursor.

Since

3.0

wxHtmlCell* wxHtmlCell::GetNext ( ) const

Returns pointer to the next cell in list (see htmlcell.h if you're interested in details).

wxHtmlContainerCell* wxHtmlCell::GetParent ( ) const

Returns pointer to parent container.

int wxHtmlCell::GetPosX ( ) const

Returns X position within parent (the value is relative to parent's upper left corner).

The returned value is meaningful only if parent's Layout() was called before!

int wxHtmlCell::GetPosY ( ) const

Returns Y position within parent (the value is relative to parent's upper left corner).

The returned value is meaningful only if parent's Layout() was called before!

int wxHtmlCell::GetWidth ( ) const

Returns width of the cell (m_Width member).

virtual void wxHtmlCell::Layout (int *w*) [virtual]

Layouts the cell.

This method performs two actions:

1. adjusts the cell's width according to the fact that maximal possible width is *w* (this has sense when working with horizontal lines, tables etc.)
2. prepares layout (=fill-in m_PosX, m_PosY (and sometimes m_Height) members) based on actual width *w*

It must be called before displaying cells structure because m_PosX and m_PosY are undefined (or invalid) before calling Layout().

virtual bool wxHtmlCell::ProcessMouseClick (wxHtmlWindowInterface * *window*, const wxPoint & *pos*, const wxMouseEvent & *event*) [virtual]

This function is simple event handler.

Each time the user clicks mouse button over a cell within wxHtmlWindow this method of that cell is called. Default behaviour is to call wxHtmlWindow::LoadPage.

Parameters

window	interface to the parent HTML window
pos	coordinates of mouse click (this is relative to cell's origin
event	mouse event that triggered the call

Returns

true if a link was clicked, false otherwise.

Since

2.7.0 (before OnMouseClick() method served a similar purpose).

Note

If you need more "advanced" event handling you should use wxHtmlBinderCell instead.

void wxHtmlCell::SetId (const wxString & *id*)

Sets unique cell identifier.

Default value is no identifier, i.e. empty string.

void wxHtmlCell::SetLink (const wxHtmlLinkInfo & *link*)

Sets the hypertext link associated with this cell.

(Default value is wxHtmlLinkInfo("", "") (no link))

void wxHtmlCell::SetNext (wxHtmlCell * *cell*)

Sets the next cell in the list.

This shouldn't be called by user - it is to be used only by wxHtmlContainerCell::InsertCell.

void wxHtmlCell::SetParent (wxHtmlContainerCell * *p*)

Sets parent container of this cell.

This is called from wxHtmlContainerCell::InsertCell.

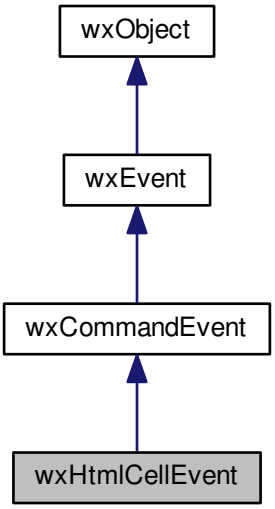
virtual void wxHtmlCell::SetPos (int *x*, int *y*) `[virtual]`

Sets the cell's position within parent container.

21.354 wxHtmlCellEvent Class Reference

`#include <wx/html/htmlwin.h>`

Inheritance diagram for wxHtmlCellEvent:



21.354.1 Detailed Description

This event class is used for the events generated by wxHtmlWindow.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHtmlCellEvent& event)

Event macros:

- EVT_HTML_CELL_HOVER(id, func): User moved the mouse over a wxHtmlCell.
- EVT_HTML_CELL_CLICKED(id, func): User clicked on a wxHtmlCell. When handling this event, remember to use wxHtmlCell::SetLinkClicked(true) if the cell contains a link.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlCellEvent (wxEventType commandType, int id, wxHtmlCell *cell, const wxPoint &point, const wxMouseEvent &ev)

  The constructor is not normally used by the user code.
- wxHtmlCell * GetCell () const

  Returns the wxHtmlCellEvent associated with the event.

- bool GetLinkClicked () const

  Returns true if SetLinkClicked(true) has previously been called; false otherwise.

- wxPoint GetPoint () const

  Returns the wxPoint associated with the event.

- void SetLinkClicked (bool linkclicked)

  Call this function with linkclicked set to true if the cell which has been clicked contained a link or false otherwise (which is the default).

Additional Inherited Members

21.354.2 Constructor & Destructor Documentation

wxHtmlCellEvent::wxHtmlCellEvent (wxEventType *commandType,* int *id,* wxHtmlCell * *cell,* const wxPoint & *point,* const wxMouseEvent & *ev*)

The constructor is not normally used by the user code.

21.354.3 Member Function Documentation

wxHtmlCell* wxHtmlCellEvent::GetCell ( ) const

Returns the wxHtmlCellEvent associated with the event.

bool wxHtmlCellEvent::GetLinkClicked ( ) const

Returns true if SetLinkClicked(true) has previously been called; false otherwise.

wxPoint wxHtmlCellEvent::GetPoint ( ) const

Returns the wxPoint associated with the event.

void wxHtmlCellEvent::SetLinkClicked (bool *linkclicked*)

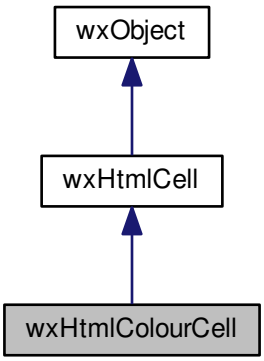
Call this function with *linkclicked* set to true if the cell which has been clicked contained a link or false otherwise (which is the default).

With this function the event handler can return info to the wxHtmlWindow which sent the event.

21.355 wxHtmlColourCell Class Reference

`#include <wx/html/htmlcell.h>`

Inheritance diagram for wxHtmlColourCell:



21.355.1 Detailed Description

This cell changes the colour of either the background or the foreground.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlColourCell (const wxColour &clr, int flags=wxHTML_CLR_FOREGROUND)

  Constructor.

Additional Inherited Members

21.355.2 Constructor & Destructor Documentation

wxHtmlColourCell::wxHtmlColourCell (const wxColour & *clr,* int *flags* = wxHTML_CLR_FOREGROUND)

Constructor.

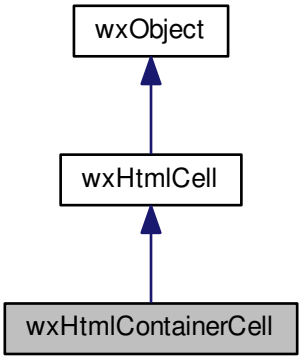
Parameters

clr	The color
flags	Can be one of following: • wxHTML_CLR_FOREGROUND: change color of text • wxHTML_CLR_BACKGROUND: change background color

21.356 wxHtmlContainerCell Class Reference

`#include <wx/html/htmlcell.h>`

Inheritance diagram for wxHtmlContainerCell:



21.356.1 Detailed Description

The wxHtmlContainerCell class is an implementation of a cell that may contain more cells in it.

It is heavily used in the wxHTML layout algorithm.

Library: wxHTML

Category: HTML

See also

Cells and Containers

Public Member Functions

- wxHtmlContainerCell (wxHtmlContainerCell *parent)

  Constructor.
- int GetAlignHor () const

  Returns container's horizontal alignment.
- int GetAlignVer () const

  Returns container's vertical alignment.
- wxColour GetBackgroundColour ()

  Returns the background colour of the container or `wxNullColour` if no background colour is set.
- int GetIndent (int ind) const

  Returns the indentation.
- int GetIndentUnits (int ind) const

  *Returns the units of indentation for ind where ind is one of the **wxHTML_INDENT_*** constants.*

- void InsertCell (wxHtmlCell *cell)

  Inserts a new cell into the container.
- void SetAlign (const wxHtmlTag &tag)

  Sets the container's alignment (both horizontal and vertical) according to the values stored in tag.
- void SetAlignHor (int al)

  Sets the container's horizontal alignment.
- void SetAlignVer (int al)

  Sets the container's vertical alignment.
- void SetBackgroundColour (const wxColour &clr)

  Sets the background colour for this container.
- void SetBorder (const wxColour &clr1, const wxColour &clr2, int border=1)

  Sets the border (frame) colours.
- void SetIndent (int i, int what, int units=wxHTML_UNITS_PIXELS)

  Sets the indentation (free space between borders of container and subcells).
- void SetMinHeight (int h, int align=wxHTML_ALIGN_TOP)

  Sets minimal height of the container.
- void SetWidthFloat (int w, int units)

  Sets floating width adjustment.
- void SetWidthFloat (const wxHtmlTag &tag, double pixel_scale=1.0)

  Sets floating width adjustment.

Additional Inherited Members

21.356.2 Constructor & Destructor Documentation

wxHtmlContainerCell::wxHtmlContainerCell (wxHtmlContainerCell * *parent*)

Constructor.

parent is pointer to parent container or NULL.

21.356.3 Member Function Documentation

int wxHtmlContainerCell::GetAlignHor ( ) const

Returns container's horizontal alignment.

int wxHtmlContainerCell::GetAlignVer ( ) const

Returns container's vertical alignment.

wxColour wxHtmlContainerCell::GetBackgroundColour ( )

Returns the background colour of the container or `wxNullColour` if no background colour is set.

int wxHtmlContainerCell::GetIndent (int *ind*) const

Returns the indentation.

ind is one of the **wxHTML_INDENT_*** constants.

Note

You must call GetIndentUnits() with same *ind* parameter in order to correctly interpret the returned integer value. It is NOT always in pixels!

int wxHtmlContainerCell::GetIndentUnits (int *ind*) const

Returns the units of indentation for *ind* where *ind* is one of the **wxHTML_INDENT**_∗ constants.

void wxHtmlContainerCell::InsertCell (wxHtmlCell ∗ *cell*)

Inserts a new cell into the container.

void wxHtmlContainerCell::SetAlign (const wxHtmlTag & *tag*)

Sets the container's alignment (both horizontal and vertical) according to the values stored in *tag*.

(Tags `ALIGN` parameter is extracted.) In fact it is only a front-end to SetAlignHor() and SetAlignVer().

void wxHtmlContainerCell::SetAlignHor (int *al*)

Sets the container's *horizontal* alignment.

During wxHtmlCell::Layout each line is aligned according to *al* value.

Parameters

al	new horizontal alignment. May be one of these values: • wxHTML_ALIGN_LEFT: lines are left-aligned (default) • wxHTML_ALIGN_JUSTIFY: lines are justified • wxHTML_ALIGN_CENTER: lines are centered • wxHTML_ALIGN_RIGHT: lines are right-aligned

void wxHtmlContainerCell::SetAlignVer (int *al*)

Sets the container's *vertical* alignment.

This is per-line alignment!

Parameters

al	new vertical alignment. May be one of these values: • wxHTML_ALIGN_BOTTOM: cells are over the line (default) • wxHTML_ALIGN_CENTER: cells are centered on line • wxHTML_ALIGN_TOP: cells are under the line

void wxHtmlContainerCell::SetBackgroundColour (const wxColour & *clr*)

Sets the background colour for this container.

void wxHtmlContainerCell::SetBorder (const wxColour & *clr1,* const wxColour & *clr2,* int *border* = 1)

Sets the border (frame) colours.

A border is a rectangle around the container.

Parameters

clr1	Colour of top and left lines
clr2	Colour of bottom and right lines
border	Size of the border in pixels

void wxHtmlContainerCell::SetIndent (int *i,* int *what,* int *units* = wxHTML_UNITS_PIXELS)

Sets the indentation (free space between borders of container and subcells).

Parameters

i	Indentation value.
what	Determines which of the four borders we're setting. It is OR combination of following constants: • wxHTML_INDENT_TOP: top border • wxHTML_INDENT_BOTTOM: bottom • wxHTML_INDENT_LEFT: left • wxHTML_INDENT_RIGHT: right • wxHTML_INDENT_HORIZONTAL: left and right • wxHTML_INDENT_VERTICAL: top and bottom • wxHTML_INDENT_ALL: all 4 borders
units	Units of i. This parameter affects interpretation of value. • wxHTML_UNITS_PIXELS: *i* is number of pixels • wxHTML_UNITS_PERCENT: *i* is interpreted as percents of width of parent container

void wxHtmlContainerCell::SetMinHeight (int *h,* int *align* = wxHTML_ALIGN_TOP)

Sets minimal height of the container.

When container's wxHtmlCell::Layout is called, m_Height is set depending on layout of subcells to the height of area covered by layed-out subcells. Calling this method guarantees you that the height of container is never smaller than *h* - even if the subcells cover much smaller area.

Parameters

h	The minimal height.
align	If height of the container is lower than the minimum height, empty space must be inserted somewhere in order to ensure minimal height. This parameter is one of wxHTML_ALIGN_TOP, wxHTML_ALIGN_BOTTOM, wxHTML_ALIGN_CENTER. It refers to the contents, not to the empty place.

void wxHtmlContainerCell::SetWidthFloat (int *w,* int *units*)

Sets floating width adjustment.

The normal behaviour of container is that its width is the same as the width of parent container (and thus you can have only one sub-container per line). You can change this by setting the floating width adjustment.

Parameters

w	Width of the container. If the value is negative it means complement to full width of parent container. E.g. `SetWidthFloat(-50, wxHTML_UNITS_PIXELS)` sets the width of container to parent's width minus 50 pixels. This is useful when creating tables - you can call SetWidthFloat(50) and SetWidthFloat(-50).
units	Units of w This parameter affects the interpretation of value. • wxHTML_UNITS_PIXELS: *w* is number of pixels • wxHTML_UNITS_PERCENT: *w* is interpreted as percents of width of parent container

void wxHtmlContainerCell::SetWidthFloat (const wxHtmlTag & *tag,* double *pixel_scale* = `1.0`)

Sets floating width adjustment.

The normal behaviour of container is that its width is the same as the width of parent container (and thus you can have only one sub-container per line). You can change this by setting the floating width adjustment.

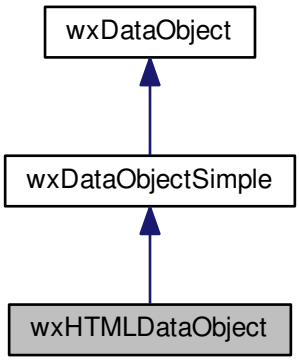
Parameters

tag	In the second version of method, *w* and *units* info is extracted from tag's WIDTH parameter.
pixel_scale	This is number of real pixels that equals to 1 HTML pixel.

21.357 wxHTMLDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxHTMLDataObject:



21.357.1 Detailed Description

wxHTMLDataObject is used for working with HTML-formatted text.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

wxDataObject, wxDataObjectSimple

Public Member Functions

- wxHTMLDataObject (const wxString &html=wxEmptyString)

  Constructor.
- virtual wxString GetHTML () const

  Returns the HTML string.
- virtual void SetHTML (const wxString &html)

  Sets the HTML string.

Additional Inherited Members

21.357.2 Constructor & Destructor Documentation

wxHTMLDataObject::wxHTMLDataObject (const wxString & *html* = wxEmptyString)

Constructor.

21.357.3 Member Function Documentation

virtual wxString wxHTMLDataObject::GetHTML ( ) const `[virtual]`

Returns the HTML string.

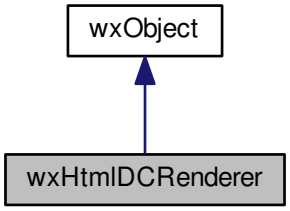
virtual void wxHTMLDataObject::SetHTML (const wxString & *html*) `[virtual]`

Sets the HTML string.

21.358 wxHtmlDCRenderer Class Reference

`#include <wx/html/htmprint.h>`

Inheritance diagram for wxHtmlDCRenderer:



21.358.1 Detailed Description

This class can render HTML document into a specified area of a DC.

You can use it in your own printing code, although use of wxHtmlEasyPrinting or wxHtmlPrintout is strongly recommended.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlDCRenderer ()

  Constructor.
- int GetTotalWidth () const

  Returns the width of the HTML text in pixels.
- int GetTotalHeight () const

  Returns the height of the HTML text in pixels.
- int Render (int x, int y, wxArrayInt &known_pagebreaks, int from=0, int dont_render=false, int to=INT_MAX)

Renders HTML text to the DC.

- void SetDC (wxDC *dc, double pixel_scale=1.0)

  Assign DC instance to the renderer.
- void SetFonts (const wxString &normal_face, const wxString &fixed_face, const int *sizes=NULL)

  This function sets font sizes and faces.
- void SetStandardFonts (int size=-1, const wxString &normal_face=wxEmptyString, const wxString &fixed_face=wxEmptyString)

  Sets font sizes to be relative to the given size or the system default size; use either specified or default font.
- void SetHtmlText (const wxString &html, const wxString &basepath=wxEmptyString, bool isdir=true)

  Assign text to the renderer.
- void SetSize (int width, int height)

  Set size of output rectangle, in pixels.

Additional Inherited Members

21.358.2 Constructor & Destructor Documentation

wxHtmlDCRenderer::wxHtmlDCRenderer ( )

Constructor.

21.358.3 Member Function Documentation

int wxHtmlDCRenderer::GetTotalHeight ( ) const

Returns the height of the HTML text in pixels.

This is important if area height (see wxHtmlDCRenderer::SetSize) is smaller that total height and thus the page cannot fit into it. In that case you're supposed to call Render() as long as its return value is smaller than GetTotalHeight()'s.

See also

GetTotalWidth()

int wxHtmlDCRenderer::GetTotalWidth ( ) const

Returns the width of the HTML text in pixels.

This can be compared with the width parameter of SetSize() to check if the document being printed fits into the page boundary.

See also

GetTotalHeight()

Since

2.9.0

int wxHtmlDCRenderer::Render (int *x*, int *y*, wxArrayInt & *known_pagebreaks*, int *from* = `0`, int *dont_render* = `false`, int *to* = `INT_MAX`)

Renders HTML text to the DC.

Parameters

x,y	position of upper-left corner of printing rectangle (see SetSize()).
known_↩ pagebreaks	

Todo docme

Parameters

from	y-coordinate of the very first visible cell.
dont_render	if true then this method only returns y coordinate of the next page and does not output anything.
to	y-coordinate of the last visible cell.

Returned value is y coordinate of first cell than didn't fit onto page. Use this value as from in next call to Render() in order to print multipages document.

Note

The following three methods **must** always be called before any call to Render(), in this order:

- SetDC()
- SetSize()
- SetHtmlText()

Render() changes the DC's user scale and does NOT restore it.

void wxHtmlDCRenderer::SetDC (wxDC * *dc*, double *pixel_scale* = 1.0)

Assign DC instance to the renderer.

pixel_scale can be used when rendering to high-resolution DCs (e.g. printer) to adjust size of pixel metrics. (Many dimensions in HTML are given in pixels – e.g. image sizes. 300x300 image would be only one inch wide on typical printer. With pixel_scale = 3.0 it would be 3 inches.)

void wxHtmlDCRenderer::SetFonts (const wxString & *normal_face*, const wxString & *fixed_face*, const int * *sizes* = NULL)

This function sets font sizes and faces.

Parameters

normal_face	This is face name for normal (i.e. non-fixed) font. It can be either empty string (then the default face is chosen) or platform-specific face name. Examples are "helvetica" under Unix or "Times New Roman" under Windows.
fixed_face	The same thing for fixed face (<TT>..</TT>)
sizes	This is an array of 7 items of int type. The values represent size of font with HTML size from -2 to +4 (<FONT SIZE=-2> to <FONT SIZE=+4>). Default sizes are used if sizes is NULL.

Default font sizes are defined by constants wxHTML_FONT_SIZE_1, wxHTML_FONT_SIZE_2, ..., wxHTML_FO↩ NT_SIZE_7. Note that they differ among platforms. Default face names are empty strings.

See also

SetSize()

void wxHtmlDCRenderer::SetHtmlText (const wxString & *html*, const wxString & *basepath* = wxEmptyString, bool *isdir* = true)

Assign text to the renderer.

Render() then draws the text onto DC.

Parameters

html	HTML text. This is not a filename.
basepath	base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.
isdir	false if basepath is filename, true if it is directory name (see wxFileSystem for detailed explanation).

void wxHtmlDCRenderer::SetSize (int *width*, int *height*)

Set size of output rectangle, in pixels.

Note that you **can't** change width of the rectangle between calls to Render() ! (You can freely change height.)

void wxHtmlDCRenderer::SetStandardFonts (int *size* = −1, const wxString & *normal_face* = wxEmptyString, const wxString & *fixed_face* = wxEmptyString)

Sets font sizes to be relative to the given size or the system default size; use either specified or default font.

Parameters

size	Point size of the default HTML text
normal_face	This is face name for normal (i.e. non-fixed) font. It can be either empty string (then the default face is chosen) or platform-specific face name. Examples are "helvetica" under Unix or "Times New Roman" under Windows.
fixed_face	The same thing for fixed face (<TT>..</TT>)

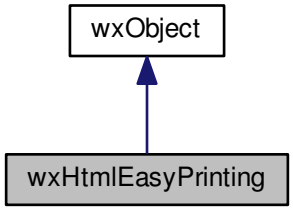
See also

SetSize()

21.359 wxHtmlEasyPrinting Class Reference

`#include <wx/html/htmprint.h>`

Inheritance diagram for wxHtmlEasyPrinting:



21.359.1 Detailed Description

This class provides very simple interface to printing architecture.

It allows you to print HTML documents using only a few commands.

Note

Do not create this class on the stack only. You should create an instance on app startup and use this instance for all printing operations. The reason is that this class stores various settings in it.

Library: wxHTML

Category: HTML, Printing Framework

Public Member Functions

- wxHtmlEasyPrinting (const wxString &name="Printing", wxWindow *parentWindow=NULL)

  Constructor.
- const wxString & GetName () const

  Returns the current name being used for preview frames and setup dialogs.
- wxPageSetupDialogData * GetPageSetupData ()

  Returns a pointer to wxPageSetupDialogData instance used by this class.
- wxWindow * GetParentWindow () const

  Gets the parent window for dialogs.
- wxPrintData * GetPrintData ()

  Returns pointer to wxPrintData instance used by this class.
- void PageSetup ()

  Display page setup dialog and allows the user to modify settings.
- bool PreviewFile (const wxString &htmlfile)

  Preview HTML file.
- bool PreviewText (const wxString &htmltext, const wxString &basepath=wxEmptyString)

  Preview HTML text (not file!).
- bool PrintFile (const wxString &htmlfile)

  Print HTML file.
- bool PrintText (const wxString &htmltext, const wxString &basepath=wxEmptyString)

  Print HTML text (not file!).
- void SetFonts (const wxString &normal_face, const wxString &fixed_face, const int *sizes=NULL)

  Sets fonts.
- void SetName (const wxString &name)

  Sets the name used for preview frames and setup dialogs.
- void SetStandardFonts (int size=-1, const wxString &normal_face=wxEmptyString, const wxString &fixed_face=wxEmptyString)

  Sets default font sizes and/or default font size.
- void SetFooter (const wxString &footer, int pg=wxPAGE_ALL)

  Set page footer.
- void SetHeader (const wxString &header, int pg=wxPAGE_ALL)

  Set page header.
- void SetParentWindow (wxWindow *window)

  Sets the parent window for dialogs.

Private Member Functions

- virtual bool CheckFit (const wxSize &pageArea, const wxSize &docArea) const

  Check whether the document fits into the page area.

Additional Inherited Members

21.359.2 Constructor & Destructor Documentation

wxHtmlEasyPrinting::wxHtmlEasyPrinting (const wxString & *name* = "Printing", wxWindow * *parentWindow* = NULL)

Constructor.

Parameters

name	Name of the printing object. Used by preview frames and setup dialogs.
parentWindow	pointer to the window that will own the preview frame and setup dialogs. May be NULL.

21.359.3 Member Function Documentation

virtual bool wxHtmlEasyPrinting::CheckFit (const wxSize & *pageArea*, const wxSize & *docArea*) const [private], [virtual]

Check whether the document fits into the page area.

This function is called by the base class OnPreparePrinting() implementation and by default checks whether the document fits into *pageArea* horizontally and warns the user if it does not, giving him the possibility to cancel printing in this case (presumably in order to change some layout options and retry it again).

You may override it to either suppress this check if truncation of the HTML being printed is acceptable or, on the contrary, add more checks to it, e.g. for the fit in the vertical direction if the document should always appear on a single page.

Returns

true if wxHtmlPrintout should continue or false to cancel printing.

Since

2.9.0

const wxString& wxHtmlEasyPrinting::GetName ( ) const

Returns the current name being used for preview frames and setup dialogs.

Since

2.8.11 / 2.9.1

wxPageSetupDialogData* wxHtmlEasyPrinting::GetPageSetupData ( )

Returns a pointer to wxPageSetupDialogData instance used by this class.

You can set its parameters (via SetXXXX methods).

wxWindow* wxHtmlEasyPrinting::GetParentWindow ( ) const

Gets the parent window for dialogs.

wxPrintData* wxHtmlEasyPrinting::GetPrintData ( )

Returns pointer to wxPrintData instance used by this class.

You can set its parameters (via SetXXXX methods).

void wxHtmlEasyPrinting::PageSetup ( )

Display page setup dialog and allows the user to modify settings.

bool wxHtmlEasyPrinting::PreviewFile (const wxString & *htmlfile*)

Preview HTML file.

Returns false in case of error – call wxPrinter::GetLastError to get detailed information about the kind of the error.

bool wxHtmlEasyPrinting::PreviewText (const wxString & *htmltext,* const wxString & *basepath* = wxEmptyString)

Preview HTML text (not file!).

Returns false in case of error – call wxPrinter::GetLastError to get detailed information about the kind of the error.

Parameters

htmltext	HTML text.
basepath	base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.

bool wxHtmlEasyPrinting::PrintFile (const wxString & *htmlfile*)

Print HTML file.

Returns false in case of error – call wxPrinter::GetLastError to get detailed information about the kind of the error.

bool wxHtmlEasyPrinting::PrintText (const wxString & *htmltext,* const wxString & *basepath* = wxEmptyString)

Print HTML text (not file!).

Returns false in case of error – call wxPrinter::GetLastError to get detailed information about the kind of the error.

Parameters

htmltext	HTML text.
basepath	base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.

void wxHtmlEasyPrinting::SetFonts (const wxString & *normal_face,* const wxString & *fixed_face,* const int * *sizes* = NULL)

Sets fonts.

See wxHtmlDCRenderer::SetFonts for detailed description.

void wxHtmlEasyPrinting::SetFooter (const wxString & *footer,* int *pg* = wxPAGE_ALL)

Set page footer.

The following macros can be used inside it: @DATE@ is replaced by the current date in default format @PAGEN↩ UM@ is replaced by page number @PAGESCNT@ is replaced by total number of pages @TIME@ is replaced by the current time in default format @TITLE@ is replaced with the title of the document

Parameters

footer	HTML text to be used as footer.
pg	one of wxPAGE_ODD, wxPAGE_EVEN and wxPAGE_ALL constants.

void wxHtmlEasyPrinting::SetHeader (const wxString & *header,* int *pg* = wxPAGE_ALL)

Set page header.

The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format
- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document

Parameters

header	HTML text to be used as header.
pg	one of wxPAGE_ODD, wxPAGE_EVEN and wxPAGE_ALL constants.

void wxHtmlEasyPrinting::SetName (const wxString & *name*)

Sets the name used for preview frames and setup dialogs.

Since

2.8.11 / 2.9.1

void wxHtmlEasyPrinting::SetParentWindow (wxWindow * *window*)

Sets the parent window for dialogs.

void wxHtmlEasyPrinting::SetStandardFonts (int *size* = -1, const wxString & *normal_face* = wxEmptyString, const wxString & *fixed_face* = wxEmptyString)

Sets default font sizes and/or default font size.

See wxHtmlDCRenderer::SetStandardFonts for detailed description.

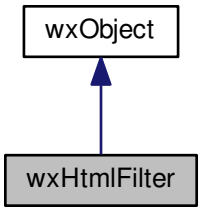
See also

SetFonts()

21.360 wxHtmlFilter Class Reference

`#include <wx/html/htmlfilt.h>`

Inheritance diagram for wxHtmlFilter:



21.360.1 Detailed Description

This class is the parent class of input filters for wxHtmlWindow.

It allows you to read and display files of different file formats.

Library: wxHTML

Category: HTML

See also

Input Filters

Public Member Functions

- wxHtmlFilter ()

  Constructor.
- virtual bool CanRead (const wxFSFile &file) const =0

  Returns true if this filter is capable of reading file file.
- virtual wxString ReadFile (const wxFSFile &file) const =0

  Reads the file and returns string with HTML document.

Additional Inherited Members

21.360.2 Constructor & Destructor Documentation

wxHtmlFilter::wxHtmlFilter ( )

Constructor.

21.360.3 Member Function Documentation

virtual bool wxHtmlFilter::CanRead (const wxFSFile & *file*) const `[pure virtual]`

Returns true if this filter is capable of reading file *file*.

Example:

```
bool MyFilter::CanRead(const wxFSFile& file)
{
    return (file.GetMimeType() == "application/x-ugh");
}
```

virtual wxString wxHtmlFilter::ReadFile (const wxFSFile & *file*) const `[pure virtual]`

Reads the file and returns string with HTML document.

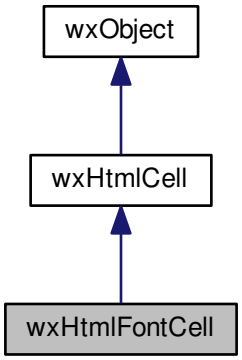
Example:

```
wxString MyImgFilter::ReadFile(const wxFSFile& file)
{
    return "<html><body><img src=\"" + file.GetLocation() +
           "\"></body></html>";
}
```

21.361 wxHtmlFontCell Class Reference

`#include <wx/html/htmlcell.h>`

Inheritance diagram for wxHtmlFontCell:



21.361.1 Detailed Description

This cell represents a font change in the document stream.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlFontCell (wxFont *font)

Additional Inherited Members

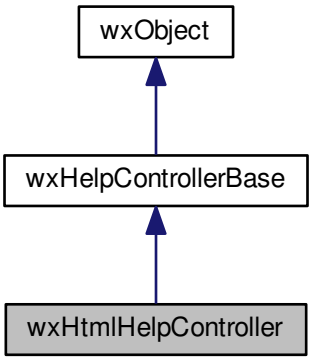
21.361.2 Constructor & Destructor Documentation

wxHtmlFontCell::wxHtmlFontCell (wxFont * *font*)

21.362 wxHtmlHelpController Class Reference

`#include <wx/html/helpctrl.h>`

Inheritance diagram for wxHtmlHelpController:



21.362.1 Detailed Description

This help controller provides an easy way of displaying HTML help in your application (see HTML Sample, test example).

The help system is based on **books** (see wxHtmlHelpController::AddBook). A book is a logical section of documentation (for example "User's Guide" or "Programmer's Guide" or "C++ Reference" or "wxWidgets Reference"). The help controller can handle as many books as you want.

Although this class has an API compatible with other wxWidgets help controllers as documented by wxHelpController, it is recommended that you use the enhanced capabilities of wxHtmlHelpController's API.

wxHTML uses Microsoft's HTML Help Workshop project files (.hhp, .hhk, .hhc) as its native format. The file format is described in Help Files Format. The directory `helpfiles` in the HTML Sample contains sample project files.

Note that the Microsoft's HTML Help Workshop (`http://www.microsoft.com/downloads/details.aspx?FamilyID=00535334-c8a6-452f-9aa0-d597d16580cc`) also runs on other platforms using WINE (`http://www.winehq.org/`) and it can be used to create the .hpp, .hhk and .hhc files through a friendly GUI. The commercial tool HelpBlocks (`http://www.helpblocks.com`) can also create these files.

Library: wxHTML

Category: Help, HTML

See also

wxBestHelpController, wxHtmlHelpFrame, wxHtmlHelpDialog, wxHtmlHelpWindow, wxHtmlModalHelp

Public Member Functions

- wxHtmlHelpController (int style=wxHF_DEFAULT_STYLE, wxWindow *parentWindow=NULL)

  Constructor.
- wxHtmlHelpController (wxWindow *parentWindow, int style=wxHF_DEFAULT_STYLE)
- bool AddBook (const wxFileName &bookFile, bool showWaitMsg=false)

  Adds a book (i.e.
- bool AddBook (const wxString &bookUrl, bool showWaitMsg=false)

  Adds a book (i.e.
- bool Display (const wxString &x)

  Displays page x.
- bool Display (int id)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
  This alternative form is used to search help contents by numeric IDs.
- virtual bool DisplayContents ()

  Displays help window and focuses contents panel.
- bool DisplayIndex ()

  Displays help window and focuses index panel.
- virtual bool KeywordSearch (const wxString &keyword, wxHelpSearchMode mode=wxHELP_SEARCH_ALL)

  Displays the help window, focuses search panel and starts searching.
- virtual void ReadCustomization (wxConfigBase *cfg, const wxString &path=wxEmptyString)

  Reads the controller's setting (position of window, etc.)
- void SetShouldPreventAppExit (bool enable)

  Sets whether the help frame should prevent application from exiting if it's the only remaining top level window.
- void SetTempDir (const wxString &path)

  Sets the path for storing temporary files - cached binary versions of index and contents files.
- void SetTitleFormat (const wxString &format)

  Sets format of title of the frame.
- void UseConfig (wxConfigBase *config, const wxString &rootpath=wxEmptyString)

  Associates the config object with the controller.
- virtual void WriteCustomization (wxConfigBase *cfg, const wxString &path=wxEmptyString)

  Stores controllers setting (position of window etc.)
- wxHtmlHelpWindow * GetHelpWindow ()

  Get the current help window.
- void SetHelpWindow (wxHtmlHelpWindow *helpWindow)

  Set the help window to be managed by this controller.
- wxHtmlHelpFrame * GetFrame ()

  Returns the current help frame.
- wxHtmlHelpDialog * GetDialog ()

  Returns the current help dialog.

Protected Member Functions

- virtual wxHtmlHelpDialog * CreateHelpDialog (wxHtmlHelpData *data)

  This protected virtual method may be overridden so that when specifying the `wxHF_DIALOG` *style, the controller uses a different dialog.*

- virtual wxHtmlHelpFrame * CreateHelpFrame (wxHtmlHelpData *data)

  This protected virtual method may be overridden so that the controller uses a different frame.

Additional Inherited Members

21.362.2 Constructor & Destructor Documentation

wxHtmlHelpController::wxHtmlHelpController (**int** ***style*** **= wxHF_DEFAULT_STYLE,** **wxWindow** * ***parentWindow*** **=** `NULL` **)**

Constructor.

Parameters

style	This is a combination of these flags: • wxHF_TOOLBAR: The help window has a toolbar. • wxHF_FLAT_TOOLBAR: The help window has a toolbar with flat buttons (aka coolbar). • wxHF_CONTENTS: The help window has a contents panel. • wxHF_INDEX: The help window has an index panel. • wxHF_SEARCH: The help window has a search panel. • wxHF_BOOKMARKS: The help window has bookmarks controls. • wxHF_OPEN_FILES: Allows user to open arbitrary HTML document. • wxHF_PRINT: The toolbar contains "print" button. • wxHF_MERGE_BOOKS: The contents pane does not show book nodes. All books are merged together and appear as single book to the user. • wxHF_ICONS_BOOK: All nodes in contents pane have a book icon. This is how Microsoft's HTML help viewer behaves. • wxHF_ICONS_FOLDER: Book nodes in contents pane have a book icon, book's sections have a folder icon. This is the default. • wxHF_ICONS_BOOK_CHAPTER: Both book nodes and nodes of top-level sections of a book (i.e. chapters) have a book icon, all other sections (sections, subsections, ...) have a folder icon. • wxHF_EMBEDDED: Specifies that the help controller controls an embedded window of class wxHtmlHelpWindow that should not be destroyed when the controller is destroyed. • wxHF_DIALOG: Specifies that the help controller should create a dialog containing the help window. • wxHF_FRAME: Specifies that the help controller should create a frame containing the help window. This is the default if neither wxHF_DIALOG nor wxHF_EMBEDDED is specified. • wxHF_MODAL: Specifies that the help controller should create a modal dialog containing the help window (used with the wxHF_DIALOG style). • wxHF_DEFAULT_STYLE: wxHF_TOOLBAR | wxHF_CONTENTS | wxHF_INDEX | wxHF_SEARCH | wxHF_BOOKMARKS | wxHF_PRINT
parentWindow	This is an optional window to be used as the parent for the help window.

wxHtmlHelpController::wxHtmlHelpController (wxWindow * *parentWindow,* int *style* = wxHF_DEFAULT_STYLE)

21.362.3 Member Function Documentation

bool wxHtmlHelpController::AddBook (const wxFileName & *bookFile,* bool *showWaitMsg* = `false`)

Adds a book (i.e.

a .hhp file; an HTML Help Workshop project file) into the list of loaded books.

This must be called at least once before displaying any help. *bookFile* or *bookUrl* may be either "`.hhp`" file or a ZIP archive that contains an arbitrary number of "`.hhp`" files in its top-level directory. This ZIP archive must have

".zip" or ".htb" extension (the latter stands for "HTML book"). In other words,

```
AddBook(wxFileName("help.zip"))
```

is possible and is the recommended way.

Parameters

bookFile	Help book filename. It is recommended to use this prototype instead of the one taking URL, because it is less error-prone.
showWaitMsg	If true then a decoration-less window with progress message is displayed.

bool wxHtmlHelpController::AddBook (const wxString & *bookUrl*, bool *showWaitMsg* = false)

Adds a book (i.e.

a .hhp file; an HTML Help Workshop project file) into the list of loaded books.

See the other overload for additional info.

Parameters

bookUrl	Help book URL (note that syntax of filename and URL is different on most platforms).
showWaitMsg	If true then a decoration-less window with progress message is displayed.

virtual wxHtmlHelpDialog* wxHtmlHelpController::CreateHelpDialog (wxHtmlHelpData * *data*) [protected], [virtual]

This protected virtual method may be overridden so that when specifying the wxHF_DIALOG style, the controller uses a different dialog.

virtual wxHtmlHelpFrame* wxHtmlHelpController::CreateHelpFrame (wxHtmlHelpData * *data*) [protected], [virtual]

This protected virtual method may be overridden so that the controller uses a different frame.

bool wxHtmlHelpController::Display (const wxString & *x*)

Displays page *x*.

This is THE important function - it is used to display the help in application. You can specify the page in many ways:

- as direct filename of HTML document
- as chapter name (from contents) or as a book name
- as some word from index
- even as any word (will be searched)

Looking for the page runs in these steps:

1. try to locate file named x (if x is for example "doc/howto.htm")
2. try to open starting page of book named x
3. try to find x in contents (if x is for example "How To ...")
4. try to find x in index (if x is for example "How To ...")
5. switch to Search panel and start searching

bool wxHtmlHelpController::Display (int *id*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This alternative form is used to search help contents by numeric IDs.

virtual bool wxHtmlHelpController::DisplayContents ( ) [virtual]

Displays help window and focuses contents panel.

Implements wxHelpControllerBase.

bool wxHtmlHelpController::DisplayIndex ( )

Displays help window and focuses index panel.

wxHtmlHelpDialog* wxHtmlHelpController::GetDialog ( )

Returns the current help dialog.

(May be NULL.)

wxHtmlHelpFrame* wxHtmlHelpController::GetFrame ( )

Returns the current help frame.

(May be NULL.)

wxHtmlHelpWindow* wxHtmlHelpController::GetHelpWindow ( )

Get the current help window.

virtual bool wxHtmlHelpController::KeywordSearch (const wxString & *keyword,* wxHelpSearchMode *mode* = wxHELP_SEARCH_ALL) [virtual]

Displays the help window, focuses search panel and starts searching.

Returns true if the keyword was found. Optionally it searches through the index (mode = `wxHELP_SEARCH_INDEX`), default the content (mode = `wxHELP_SEARCH_ALL`).

Note

KeywordSearch() searches only pages listed in `".hhc"` file(s). You should list all pages in the contents file.

Implements wxHelpControllerBase.

virtual void wxHtmlHelpController::ReadCustomization (wxConfigBase * *cfg,* const wxString & *path* = wxEmptyString) [virtual]

Reads the controller's setting (position of window, etc.)

void wxHtmlHelpController::SetHelpWindow (wxHtmlHelpWindow * *helpWindow*)

Set the help window to be managed by this controller.

This makes it possible to have a help window that might not be in a wxHtmlHelpFrame or dialog but is embedded in some other window in the application. Be sure to use the wxHF_EMBEDDED style in this case.

void wxHtmlHelpController::SetShouldPreventAppExit (bool *enable*)

Sets whether the help frame should prevent application from exiting if it's the only remaining top level window.

Parameters

enable	If true, the application will not quit unless the help frame is closed. Default is false, i.e. the application does exit if only the help window remains opened.

See also

wxApp::SetExitOnFrameDelete()

Since

2.9.2

void wxHtmlHelpController::SetTempDir (const wxString & *path*)

Sets the path for storing temporary files - cached binary versions of index and contents files.

These binary forms are much faster to read. Default value is empty string (empty string means that no cached data are stored). Note that these files are *not* deleted when program exits.

Once created these cached files will be used in all subsequent executions of your application. If cached files become older than corresponding `".hhp"` file (e.g. if you regenerate documentation) it will be refreshed.

void wxHtmlHelpController::SetTitleFormat (const wxString & *format*)

Sets format of title of the frame.

Must contain exactly one "%s" (for title of displayed HTML page).

void wxHtmlHelpController::UseConfig (wxConfigBase * *config*, const wxString & *rootpath* = wxEmptyString)

Associates the *config* object with the controller.

If there is associated config object, wxHtmlHelpController automatically reads and writes settings (including wxHtmlWindow's settings) when needed. The only thing you must do is create wxConfig object and call UseConfig().

If you do not use UseConfig(), wxHtmlHelpController will use the default wxConfig object if available (for details see wxConfigBase::Get and wxConfigBase::Set).

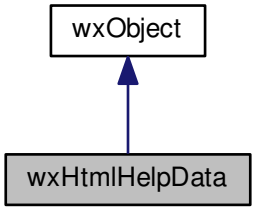
virtual void wxHtmlHelpController::WriteCustomization (wxConfigBase * *cfg*, const wxString & *path* = wxEmptyString) [virtual]

Stores controllers setting (position of window etc.)

21.363 wxHtmlHelpData Class Reference

`#include <wx/html/helpdata.h>`

Inheritance diagram for wxHtmlHelpData:



21.363.1 Detailed Description

This class is used by wxHtmlHelpController and wxHtmlHelpFrame to access HTML help items.

It is internal class and should not be used directly - except for the case you're writing your own HTML help controller.

Library: wxHTML

Category: Help, HTML

Public Member Functions

- wxHtmlHelpData ()

  Constructor.
- bool AddBook (const wxString &book_url)

  Adds new book.
- wxString FindPageById (int id)

  Returns page's URL based on integer ID stored in project.
- wxString FindPageByName (const wxString &page)

  Returns page's URL based on its (file)name.
- const wxHtmlBookRecArray & GetBookRecArray () const

  Returns array with help books info.
- const wxHtmlHelpDataItems & GetContentsArray () const

  Returns reference to array with contents entries.
- const wxHtmlHelpDataItems & GetIndexArray () const

  Returns reference to array with index entries.
- void SetTempDir (const wxString &path)

  Sets the temporary directory where binary cached versions of MS HTML Workshop files will be stored.

Additional Inherited Members

21.363.2 Constructor & Destructor Documentation

wxHtmlHelpData::wxHtmlHelpData ( )

Constructor.

21.363.3 Member Function Documentation

bool wxHtmlHelpData::AddBook (const wxString & *book_url*)

Adds new book.

book_url is URL (not filename!) of HTML help project (hhp) or ZIP file that contains arbitrary number of .hhp projects (this zip file can have either .zip or .htb extension, htb stands for "html book").

Returns success.

wxString wxHtmlHelpData::FindPageById (int *id*)

Returns page's URL based on integer ID stored in project.

wxString wxHtmlHelpData::FindPageByName (const wxString & *page*)

Returns page's URL based on its (file)name.

const wxHtmlBookRecArray& wxHtmlHelpData::GetBookRecArray ( ) const

Returns array with help books info.

const wxHtmlHelpDataItems& wxHtmlHelpData::GetContentsArray ( ) const

Returns reference to array with contents entries.

const wxHtmlHelpDataItems& wxHtmlHelpData::GetIndexArray ( ) const

Returns reference to array with index entries.

void wxHtmlHelpData::SetTempDir (const wxString & *path*)

Sets the temporary directory where binary cached versions of MS HTML Workshop files will be stored.

(This is turned off by default and you can enable this feature by setting non-empty temp dir.)

21.364 wxHtmlHelpDataItem Class Reference

`#include <wx/html/helpdata.h>`

21.364.1 Detailed Description

Helper class for wxHtmlHelpData.

Public Member Functions

- wxHtmlHelpDataItem ()
- wxString GetFullPath () const
- wxString GetIndentedName () const

Public Attributes

- int level
- wxHtmlHelpDataItem * parent
- int id
- wxString name
- wxString page
- wxHtmlBookRecord * book

21.364.2 Constructor & Destructor Documentation

wxHtmlHelpDataItem::wxHtmlHelpDataItem ( )

21.364.3 Member Function Documentation

wxString wxHtmlHelpDataItem::GetFullPath ( ) const

wxString wxHtmlHelpDataItem::GetIndentedName ( ) const

21.364.4 Member Data Documentation

wxHtmlBookRecord* wxHtmlHelpDataItem::book

int wxHtmlHelpDataItem::id

int wxHtmlHelpDataItem::level

wxString wxHtmlHelpDataItem::name

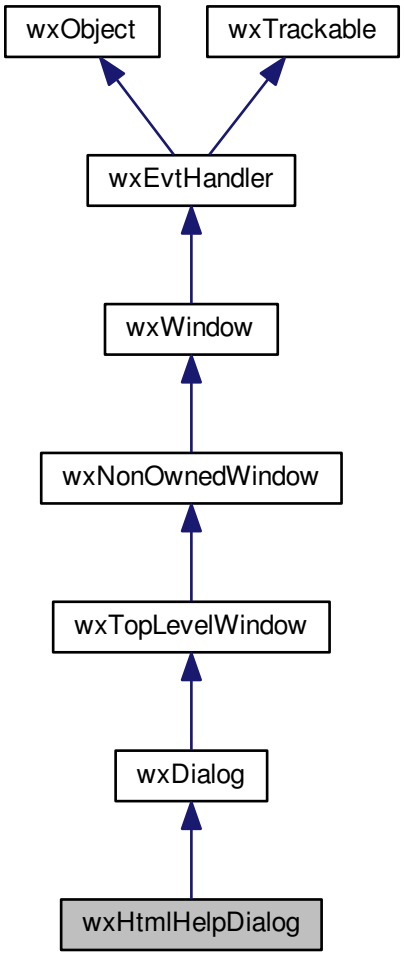
wxString wxHtmlHelpDataItem::page

wxHtmlHelpDataItem* wxHtmlHelpDataItem::parent

21.365 wxHtmlHelpDialog Class Reference

```
#include <wx/html/helpdlg.h>
```

Inheritance diagram for wxHtmlHelpDialog:



21.365.1 Detailed Description

This class is used by wxHtmlHelpController to display help.

It is an internal class and should not be used directly - except for the case when you're writing your own HTML help controller.

Library: wxHTML

Category: Help, HTML

Public Member Functions

- wxHtmlHelpDialog (wxHtmlHelpData *data=NULL)

- wxHtmlHelpDialog (wxWindow *parent, wxWindowID id, const wxString &title=wxEmptyString, int style=wxHF_DEFAULT_STYLE, wxHtmlHelpData *data=NULL)

  Constructor.
- virtual void AddToolbarButtons (wxToolBar *toolBar, int style)

  You may override this virtual method to add more buttons to the help window's toolbar.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title=wxEmptyString, int style=wxHF_DEFAULT_STYLE)

  Creates the dialog.
- wxHtmlHelpController * GetController () const

  Returns the help controller associated with the dialog.
- void SetController (wxHtmlHelpController *controller)

  Sets the help controller associated with the dialog.
- void SetTitleFormat (const wxString &format)

  Sets the dialog's title format.

Additional Inherited Members

21.365.2 Constructor & Destructor Documentation

wxHtmlHelpDialog::wxHtmlHelpDialog (wxHtmlHelpData * *data* = NULL)

wxHtmlHelpDialog::wxHtmlHelpDialog (wxWindow * *parent*, wxWindowID *id*, const wxString & *title* = wxEmptyString, int *style* = wxHF_DEFAULT_STYLE, wxHtmlHelpData * *data* = NULL)

Constructor.

For the possible values of *style*, please see wxHtmlHelpController.

21.365.3 Member Function Documentation

virtual void wxHtmlHelpDialog::AddToolbarButtons (wxToolBar * *toolBar*, int *style*) [virtual]

You may override this virtual method to add more buttons to the help window's toolbar.

toolBar is a pointer to the toolbar and *style* is the style flag as passed to the Create() method.

wxToolBar::Realize is called immediately after returning from this function.

bool wxHtmlHelpDialog::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title* = wxEmptyString, int *style* = wxHF_DEFAULT_STYLE)

Creates the dialog.

See the constructor for a description of the parameters.

wxHtmlHelpController* wxHtmlHelpDialog::GetController ( ) const

Returns the help controller associated with the dialog.

void wxHtmlHelpDialog::SetController (wxHtmlHelpController * *controller*)

Sets the help controller associated with the dialog.

void wxHtmlHelpDialog::SetTitleFormat (const wxString & *format*)

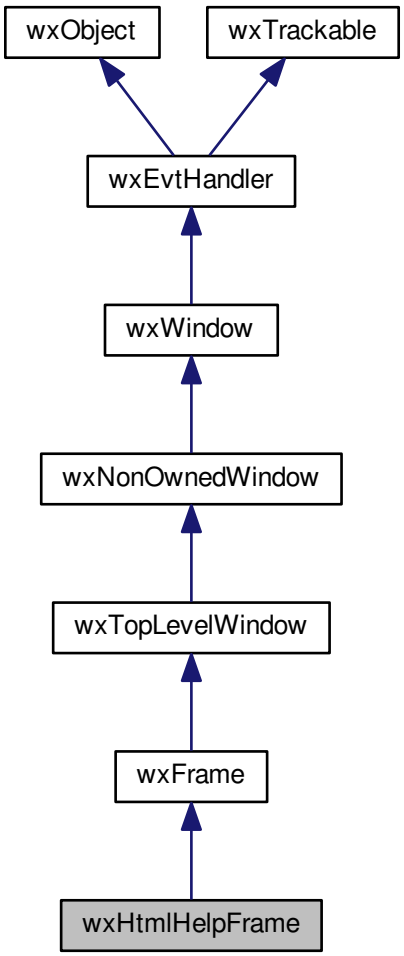
Sets the dialog's title format.

format must contain exactly one "%s" (it will be replaced by the page title).

21.366 wxHtmlHelpFrame Class Reference

`#include <wx/html/helpfrm.h>`

Inheritance diagram for wxHtmlHelpFrame:



21.366.1 Detailed Description

This class is used by wxHtmlHelpController to display help.

It is an internal class and should not be used directly - except for the case when you're writing your own HTML help controller.

Library: wxHTML

Category: Help, HTML

Public Member Functions

- wxHtmlHelpFrame (wxHtmlHelpData *data=NULL)
- wxHtmlHelpFrame (wxWindow *parent, wxWindowID id, const wxString &title=wxEmptyString, int style=wxHF_DEFAULT_STYLE, wxHtmlHelpData *data=NULL, wxConfigBase *config=NULL, const wxString &rootpath=wxEmptyString)

  Constructor.
- virtual void AddToolbarButtons (wxToolBar *toolBar, int style)

  You may override this virtual method to add more buttons to the help window's toolbar.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title=wxEmptyString, int style=wxHF_DEFAULT_STYLE, wxConfigBase *config=NULL, const wxString &rootpath=wxEmptyString)

  Creates the frame.
- wxHtmlHelpController * GetController () const

  Returns the help controller associated with the frame.
- void SetController (wxHtmlHelpController *controller)

  Sets the help controller associated with the frame.
- void SetTitleFormat (const wxString &format)

  Sets the frame's title format.

Additional Inherited Members

21.366.2 Constructor & Destructor Documentation

wxHtmlHelpFrame::wxHtmlHelpFrame (wxHtmlHelpData * *data* = NULL)

wxHtmlHelpFrame::wxHtmlHelpFrame (wxWindow * *parent*, wxWindowID *id*, const wxString & *title* = wxEmptyString, int *style* = wxHF_DEFAULT_STYLE, wxHtmlHelpData * *data* = NULL, wxConfigBase * *config* = NULL, const wxString & *rootpath* = wxEmptyString)

Constructor.

For the possible values of *style*, please see wxHtmlHelpController.

21.366.3 Member Function Documentation

virtual void wxHtmlHelpFrame::AddToolbarButtons (wxToolBar * *toolBar*, int *style*) [virtual]

You may override this virtual method to add more buttons to the help window's toolbar.

toolBar is a pointer to the toolbar and *style* is the style flag as passed to the Create() method.

wxToolBar::Realize is called immediately after returning from this function.

bool wxHtmlHelpFrame::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title* = wxEmptyString, int *style* = wxHF_DEFAULT_STYLE, wxConfigBase * *config* = NULL, const wxString & *rootpath* = wxEmptyString)

Creates the frame.

See the constructor for a description of the parameters.

wxHtmlHelpController* **wxHtmlHelpFrame::GetController ( ) const**

Returns the help controller associated with the frame.

void wxHtmlHelpFrame::SetController (wxHtmlHelpController * *controller*)

Sets the help controller associated with the frame.

void wxHtmlHelpFrame::SetTitleFormat (const wxString & *format*)

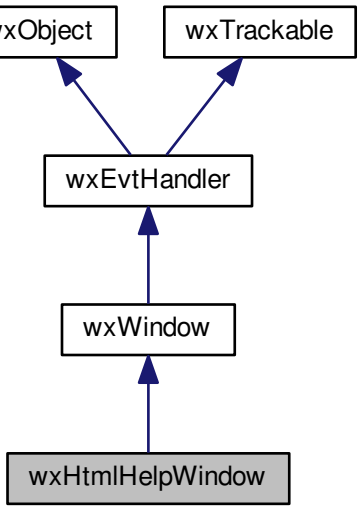
Sets the frame's title format.

format must contain exactly one "%s" (it will be replaced by the page title).

21.367 wxHtmlHelpWindow Class Reference

`#include <wx/html/helpwnd.h>`

Inheritance diagram for wxHtmlHelpWindow:



21.367.1 Detailed Description

This class is used by wxHtmlHelpController to display help within a frame or dialog, but you can use it yourself to create an embedded HTML help window.

For example:

```
// m_embeddedHelpWindow is a wxHtmlHelpWindow
// m_embeddedHtmlHelp is a wxHtmlHelpController

// Create embedded HTML Help window
m_embeddedHelpWindow = new wxHtmlHelpWindow;
m_embeddedHtmlHelp.UseConfig(config, rootPath); // Set your own config object here
```

```
m_embeddedHtmlHelp.SetHelpWindow(m_embeddedHelpWindow);
m_embeddedHelpWindow->Create(this, wxID_ANY, wxDefaultPosition,
      GetClientSize(),
                           wxTAB_TRAVERSAL|wxBORDER_NONE,
      wxHF_DEFAULT_STYLE);
m_embeddedHtmlHelp.AddBook(wxFileName(wxT("doc.zip")));
```

You should pass the style wxHF_EMBEDDED to the style parameter of wxHtmlHelpController to allow the embedded window to be destroyed independently of the help controller.

Library: wxHTML

Category: Help, HTML

Public Member Functions

- wxHtmlHelpWindow (wxHtmlHelpData *data=NULL)
- wxHtmlHelpWindow (wxWindow *parent, int wxWindowID, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int style=wxTAB_TRAVERSAL|wxBORDER_NONE, int helpStyle=wxHF_DEFAULT_STYLE, wxHtmlHelpData *data=NULL)

  Constructor.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int style=wxTAB_TRAVERSAL|wxBORDER_NONE, int helpStyle=wxHF_DEFAULT_STYLE)

  Creates the help window.
- bool Display (const wxString &x)

  Displays page x.
- bool Display (const int id)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
  This form takes numeric ID as the parameter (uses an extension to MS format, param name="ID" value=id).
- bool DisplayContents ()

  Displays contents panel.
- bool DisplayIndex ()

  Displays index panel.
- wxHtmlHelpData * GetData ()

  Returns the wxHtmlHelpData object, which is usually a pointer to the controller's data.
- bool KeywordSearch (const wxString &keyword, wxHelpSearchMode mode=wxHELP_SEARCH_ALL)

  Search for given keyword.
- void ReadCustomization (wxConfigBase *cfg, const wxString &path=wxEmptyString)

  Reads the user's settings for this window.
- void UseConfig (wxConfigBase *config, const wxString &rootpath=wxEmptyString)

  Associates a wxConfig object with the help window.
- void WriteCustomization (wxConfigBase *cfg, const wxString &path=wxEmptyString)

  Saves the user's settings for this window.
- void RefreshLists ()

  Refresh all panels.
- wxHtmlHelpController * GetController () const
- void SetController (wxHtmlHelpController *controller)

Protected Member Functions

- void CreateSearch ()

  Creates search panel.
- virtual void AddToolbarButtons (wxToolBar *toolBar, int style)

  You may override this virtual method to add more buttons to the help window's toolbar.
- void CreateContents ()

  Creates contents panel.
- void CreateIndex ()

  Creates index panel.

Additional Inherited Members

21.367.2 Constructor & Destructor Documentation

wxHtmlHelpWindow::wxHtmlHelpWindow (wxHtmlHelpData * *data* = NULL)

wxHtmlHelpWindow::wxHtmlHelpWindow (wxWindow * *parent*, int *wxWindowID*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *style* = wxTAB_TRAVERSAL|wxBORDER_NONE, int *helpStyle* = wxHF_DEFAULT_STYLE, wxHtmlHelpData * *data* = NULL)

Constructor.

For the values of *helpStyle*, please see the documentation for wxHtmlHelpController.

21.367.3 Member Function Documentation

virtual void wxHtmlHelpWindow::AddToolbarButtons (wxToolBar * *toolBar*, int *style*) [protected], [virtual]

You may override this virtual method to add more buttons to the help window's toolbar.

toolBar is a pointer to the toolbar and *style* is the style flag as passed to the Create() method.

wxToolBar::Realize is called immediately after returning from this function. See samples/html/helpview for an example.

bool wxHtmlHelpWindow::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *style* = wxTAB_TRAVERSAL|wxBORDER_NONE, int *helpStyle* = wxHF_DEFAULT_STYLE)

Creates the help window.

See the constructor for a description of the parameters.

void wxHtmlHelpWindow::CreateContents ( ) [protected]

Creates contents panel.

(May take some time.)

void wxHtmlHelpWindow::CreateIndex ( ) [protected]

Creates index panel.

(May take some time.)

void wxHtmlHelpWindow::CreateSearch ( ) [protected]

Creates search panel.

bool wxHtmlHelpWindow::Display (const wxString & *x*)

Displays page x.

If not found it will give the user the choice of searching books. Looking for the page runs in these steps:

1. try to locate file named x (if x is for example "doc/howto.htm")
2. try to open starting page of book x
3. try to find x in contents (if x is for example "How To ...")
4. try to find x in index (if x is for example "How To ...")

bool wxHtmlHelpWindow::Display (const int *id*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This form takes numeric ID as the parameter (uses an extension to MS format, param name="ID" value=id).

bool wxHtmlHelpWindow::DisplayContents ( )

Displays contents panel.

bool wxHtmlHelpWindow::DisplayIndex ( )

Displays index panel.

wxHtmlHelpController* wxHtmlHelpWindow::GetController ( ) const

wxHtmlHelpData* wxHtmlHelpWindow::GetData ( )

Returns the wxHtmlHelpData object, which is usually a pointer to the controller's data.

bool wxHtmlHelpWindow::KeywordSearch (const wxString & *keyword,* wxHelpSearchMode *mode* = wxHELP_SEARCH_ALL)

Search for given keyword.

Optionally it searches through the index (mode = `wxHELP_SEARCH_INDEX`), default the content (mode = `wxHELP_SEARCH_ALL`).

void wxHtmlHelpWindow::ReadCustomization (wxConfigBase * *cfg,* const wxString & *path* = wxEmptyString)

Reads the user's settings for this window.

See also

wxHtmlHelpController::ReadCustomization

void wxHtmlHelpWindow::RefreshLists ( )

Refresh all panels.

This is necessary if a new book was added.

void wxHtmlHelpWindow::SetController (wxHtmlHelpController * *controller*)

void wxHtmlHelpWindow::UseConfig (wxConfigBase * *config*, const wxString & *rootpath* = wxEmptyString)

Associates a wxConfig object with the help window.

It is recommended that you use wxHtmlHelpController::UseConfig instead.

void wxHtmlHelpWindow::WriteCustomization (wxConfigBase * *cfg*, const wxString & *path* = wxEmptyString)

Saves the user's settings for this window.

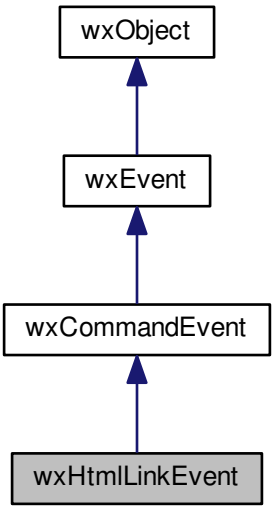
See also

wxHtmlHelpController::WriteCustomization

21.368 wxHtmlLinkEvent Class Reference

`#include <wx/html/htmlwin.h>`

Inheritance diagram for wxHtmlLinkEvent:



21.368.1 Detailed Description

This event class is used for the events generated by wxHtmlWindow.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHtmlLinkEvent& event)

Event macros:

- EVT_HTML_LINK_CLICKED(id, func): User clicked on an hyperlink.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlLinkEvent (int id, const wxHtmlLinkInfo &linkinfo)

  The constructor is not normally used by the user code.

- const wxHtmlLinkInfo & GetLinkInfo () const

  Returns the wxHtmlLinkInfo which contains info about the cell clicked and the hyperlink it contains.

Additional Inherited Members

21.368.2 Constructor & Destructor Documentation

wxHtmlLinkEvent::wxHtmlLinkEvent (int *id,* const wxHtmlLinkInfo & *linkinfo*)

The constructor is not normally used by the user code.

21.368.3 Member Function Documentation

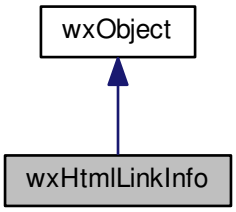
const wxHtmlLinkInfo& wxHtmlLinkEvent::GetLinkInfo ( ) const

Returns the wxHtmlLinkInfo which contains info about the cell clicked and the hyperlink it contains.

21.369 wxHtmlLinkInfo Class Reference

```
#include <wx/html/htmlcell.h>
```

Inheritance diagram for wxHtmlLinkInfo:



21.369.1 Detailed Description

This class stores all necessary information about hypertext links (as represented by <A> tag in HTML documents).

In current implementation it stores URL and target frame name.

Note

Frames are not currently supported by wxHTML!

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlLinkInfo ()
  Default ctor.
- wxHtmlLinkInfo (const wxString &href, const wxString &target=wxEmptyString)
  Construct hypertext link from HREF (aka URL) and TARGET (name of target frame).
- const wxMouseEvent * GetEvent () const
  Return pointer to event that generated OnLinkClicked() event.
- wxString GetHref () const
  Return HREF value of the <A> tag.
- const wxHtmlCell * GetHtmlCell () const
  Return pointer to the cell that was clicked.
- wxString GetTarget () const
  Return TARGET value of the <A> tag (this value is used to specify in which frame should be the page pointed by GetHref() Href opened).

Additional Inherited Members

21.369.2 Constructor & Destructor Documentation

wxHtmlLinkInfo::wxHtmlLinkInfo ( )

Default ctor.

wxHtmlLinkInfo::wxHtmlLinkInfo (const wxString & *href,* const wxString & *target* = wxEmptyString)

Construct hypertext link from HREF (aka URL) and TARGET (name of target frame).

21.369.3 Member Function Documentation

const wxMouseEvent* wxHtmlLinkInfo::GetEvent ( ) const

Return pointer to event that generated OnLinkClicked() event.

Valid only within wxHtmlWindow::OnLinkClicked, NULL otherwise.

wxString wxHtmlLinkInfo::GetHref ( ) const

Return *HREF* value of the <A> tag.

const wxHtmlCell* wxHtmlLinkInfo::GetHtmlCell ( ) const

Return pointer to the cell that was clicked.

Valid only within wxHtmlWindow::OnLinkClicked, NULL otherwise.

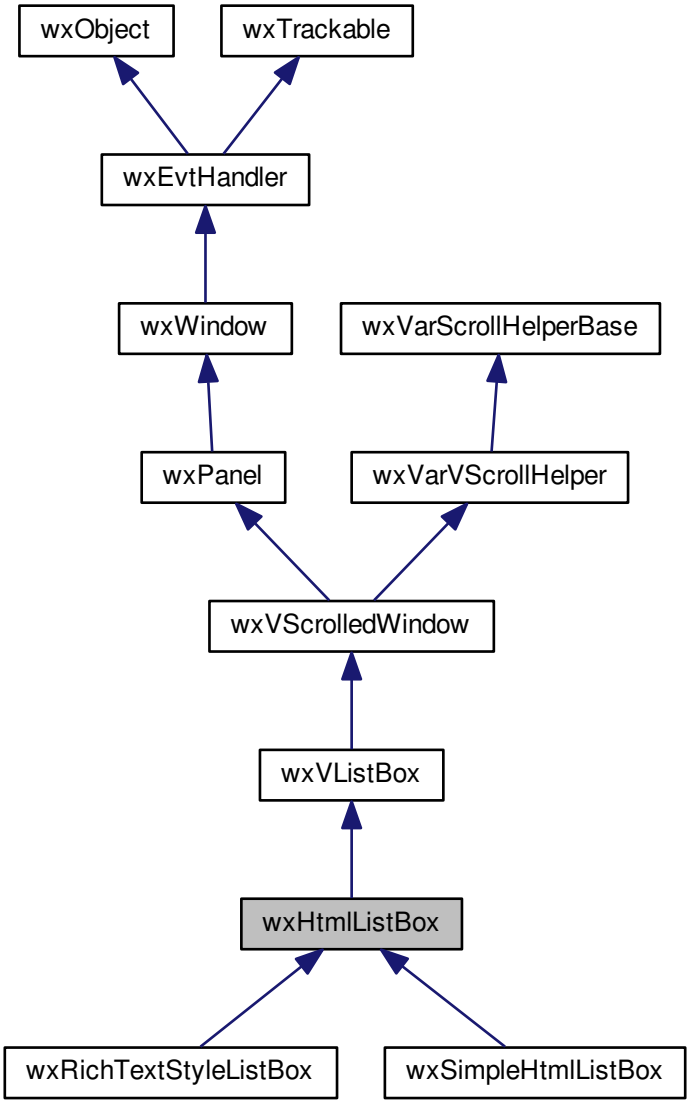
wxString wxHtmlLinkInfo::GetTarget ( ) const

Return *TARGET* value of the <A> tag (this value is used to specify in which frame should be the page pointed by GetHref() Href opened).

21.370 wxHtmlListBox Class Reference

`#include <wx/htmllbox.h>`

Inheritance diagram for wxHtmlListBox:



21.370.1 Detailed Description

wxHtmlListBox is an implementation of wxVListBox which shows HTML content in the listbox rows.

This is still an abstract base class and you will need to derive your own class from it (see htlbox sample for the example) but you will only need to override a single wxHtmlListBox::OnGetItem function.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHtmlCellEvent& event) or void handlerFuncName(wxHtmlLinkEvent& event)

Event macros for events emitted by this class:

- EVT_HTML_CELL_CLICKED(id, func): A wxHtmlCell was clicked.
- EVT_HTML_CELL_HOVER(id, func): The mouse passed over a wxHtmlCell.
- EVT_HTML_LINK_CLICKED(id, func): A wxHtmlCell which contains an hyperlink was clicked.

Library: wxHTML

Category: Controls

See also

wxSimpleHtmlListBox

Public Member Functions

- wxHtmlListBox (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxHtmlListBoxNameStr)

  Normal constructor which calls Create() internally.
- wxHtmlListBox ()

  Default constructor, you must call Create() later.
- virtual ~wxHtmlListBox ()

  Destructor cleans up whatever resources we use.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxHtmlListBoxNameStr)

  Creates the control and optionally sets the initial number of items in it (it may also be set or changed later with wxVListBox::SetItemCount).
- wxFileSystem & GetFileSystem () const

  Returns the wxFileSystem used by the HTML parser of this object.
- const wxFileSystem & GetFileSystem () const

  Returns the wxFileSystem used by the HTML parser of this object.

Protected Member Functions

- virtual void OnLinkClicked (size_t n, const wxHtmlLinkInfo &link)

  Called when the user clicks on hypertext link.
- virtual wxColour GetSelectedTextBgColour (const wxColour &colBg) const

  This virtual function may be overridden to change the appearance of the background of the selected cells in the same way as GetSelectedTextColour().
- virtual wxColour GetSelectedTextColour (const wxColour &colFg) const

  This virtual function may be overridden to customize the appearance of the selected cells.
- virtual wxString OnGetItemMarkup (size_t n) const

  This function may be overridden to decorate HTML returned by OnGetItem().
- virtual wxString OnGetItem (size_t n) const =0

  This method must be implemented in the derived class and should return the body (i.e. without `html` nor `body` tags) of the HTML fragment for the given item.

Additional Inherited Members

21.370.2 Constructor & Destructor Documentation

wxHtmlListBox::wxHtmlListBox (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxHtmlListBoxNameStr)

Normal constructor which calls Create() internally.

wxHtmlListBox::wxHtmlListBox ( )

Default constructor, you must call Create() later.

virtual wxHtmlListBox::~wxHtmlListBox ( ) [virtual]

Destructor cleans up whatever resources we use.

21.370.3 Member Function Documentation

bool wxHtmlListBox::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxHtmlListBoxNameStr)

Creates the control and optionally sets the initial number of items in it (it may also be set or changed later with wxVListBox::SetItemCount).

There are no special styles defined for wxHtmlListBox, in particular the wxListBox styles (with the exception of wxLB_MULTIPLE) cannot be used here.

Returns true on success or false if the control couldn't be created

wxFileSystem& wxHtmlListBox::GetFileSystem ( ) const

Returns the wxFileSystem used by the HTML parser of this object.

The file system object is used to resolve the paths in HTML fragments displayed in the control and you should use wxFileSystem::ChangePathTo if you use relative paths for the images or other resources embedded in your HTML.

const wxFileSystem& wxHtmlListBox::GetFileSystem ( ) const

Returns the wxFileSystem used by the HTML parser of this object.

The file system object is used to resolve the paths in HTML fragments displayed in the control and you should use wxFileSystem::ChangePathTo if you use relative paths for the images or other resources embedded in your HTML.

virtual wxColour wxHtmlListBox::GetSelectedTextBgColour (const wxColour & *colBg*) const [protected], [virtual]

This virtual function may be overridden to change the appearance of the background of the selected cells in the same way as GetSelectedTextColour().

It should be rarely, if ever, used because wxVListBox::SetSelectionBackground allows to change the selection background for all cells at once and doing anything more fancy is probably going to look strangely.

See also

GetSelectedTextColour()

virtual wxColour wxHtmlListBox::GetSelectedTextColour (const wxColour & *colFg*) const `[protected], [virtual]`

This virtual function may be overridden to customize the appearance of the selected cells.

It is used to determine how the colour *colFg* is going to look inside selection. By default all original colours are completely ignored and the standard, system-dependent, selection colour is used but the program may wish to override this to achieve some custom appearance.

See also

GetSelectedTextBgColour(), wxVListBox::SetSelectionBackground, wxSystemSettings::GetColour

virtual wxString wxHtmlListBox::OnGetItem (size_t *n*) const `[protected], [pure virtual]`

This method must be implemented in the derived class and should return the body (i.e. without `html` nor `body` tags) of the HTML fragment for the given item.

Note that this function should always return a text fragment for the *n* item which renders with the same height both when it is selected and when it's not: i.e. if you call, inside your OnGetItem() implementation, `IsSelected(n)` to make the items appear differently when they are selected, then you should make sure that the returned HTML fragment will render with the same height or else you'll see some artifacts when the user selects an item.

Implemented in wxRichTextStyleListBox.

virtual wxString wxHtmlListBox::OnGetItemMarkup (size_t *n*) const `[protected], [virtual]`

This function may be overridden to decorate HTML returned by OnGetItem().

virtual void wxHtmlListBox::OnLinkClicked (size_t *n*, const wxHtmlLinkInfo & *link*) `[protected], [virtual]`

Called when the user clicks on hypertext link.

Does nothing by default. Overloading this method is deprecated; intercept the event instead.

Parameters

n	Index of the item containing the link.
link	Description of the link.

See also

wxHtmlLinkInfo.

21.371 wxHtmlModalHelp Class Reference

`#include <wx/html/helpctrl.h>`

21.371.1 Detailed Description

This class uses wxHtmlHelpController to display help in a modal dialog.

This is useful on platforms such as wxMac where if you display help from a modal dialog, the help window must itself be a modal dialog.

Create objects of this class on the stack, for example:

```
// The help can be browsed during the lifetime of this object; when the
// user quits the help, program execution will continue.
wxHtmlModalHelp help(parent, "help", "My topic");
```

Library: wxHTML

Category: Help, HTML

Public Member Functions

- wxHtmlModalHelp (wxWindow *parent, const wxString &helpFile, const wxString &topic=wxEmptyString, int style=wxHF_DEFAULT_STYLE|wxHF_DIALOG|wxHF_MODAL)

  The ctor.

21.371.2 Constructor & Destructor Documentation

wxHtmlModalHelp::wxHtmlModalHelp (wxWindow * *parent,* const wxString & *helpFile,* const wxString & *topic =* wxEmptyString, int *style =* wxHF_DEFAULT_STYLE|wxHF_DIALOG|wxHF_MODAL)

The ctor.

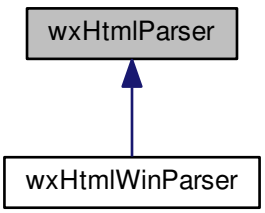
Parameters

parent	is the parent of the dialog.
helpFile	is the HTML help file to show.
topic	is an optional topic. If this is empty, the help contents will be shown.
style	is a combination of the flags described in the wxHtmlHelpController documentation.

21.372 wxHtmlParser Class Reference

`#include <wx/html/htmlpars.h>`

Inheritance diagram for wxHtmlParser:



21.372.1 Detailed Description

Classes derived from this handle the **generic** parsing of HTML documents: it scans the document and divide it into blocks of tags (where one block consists of beginning and ending tag and of text between these two tags).

It is independent from wxHtmlWindow and can be used as stand-alone parser.

It uses system of tag handlers to parse the HTML document. Tag handlers are not statically shared by all instances but are created for each wxHtmlParser instance. The reason is that the handler may contain document-specific temporary data used during parsing (e.g. complicated structures like tables).

Typically the user calls only the wxHtmlParser::Parse method.

Library: wxHTML

Category: HTML

See also

Cells and Containers, Tag Handlers, wxHtmlTag

Public Member Functions

- wxHtmlParser ()
  Constructor.
- virtual void AddTagHandler (wxHtmlTagHandler *handler)
  Adds handler to the internal list (hash table) of handlers.
- virtual void AddWord (const wxString &txt)
  Must be overwritten in derived class.
- void DoParsing (const const_iterator &begin_pos, const const_iterator &end_pos)
  Parses the m_Source from begin_pos to end_pos - 1.
- void DoParsing ()
  Parses the whole m_Source.
- virtual void DoneParser ()
  This must be called after DoParsing().
- wxFileSystem * GetFS () const
  Returns pointer to the file system.
- virtual wxObject * GetProduct ()=0
  Returns product of parsing.
- const wxString * GetSource ()
  Returns pointer to the source being parsed.
- virtual void InitParser (const wxString &source)
  Setups the parser for parsing the source string.
- virtual wxFSFile * OpenURL (wxHtmlURLType type, const wxString &url) const
  Opens given URL and returns `wxFSFile` object that can be used to read data from it.
- wxObject * Parse (const wxString &source)
  Proceeds parsing of the document.
- void PopTagHandler ()
  Restores parser's state before last call to PushTagHandler().
- void PushTagHandler (wxHtmlTagHandler *handler, const wxString &tags)
  Forces the handler to handle additional tags (not returned by wxHtmlTagHandler::GetSupportedTags).
- void SetFS (wxFileSystem *fs)
  Sets the virtual file system that will be used to request additional files.
- virtual void StopParsing ()
  Call this function to interrupt parsing from a tag handler.

Protected Member Functions

- virtual void AddTag (const wxHtmlTag &tag)

  This may (and may not) be overwritten in derived class.

21.372.2 Constructor & Destructor Documentation

wxHtmlParser::wxHtmlParser ( )

Constructor.

21.372.3 Member Function Documentation

virtual void wxHtmlParser::AddTag (const wxHtmlTag & *tag*) `[protected], [virtual]`

This may (and may not) be overwritten in derived class.

This method is called each time new tag is about to be added. *tag* contains information about the tag. (See wxHtmlTag for details.)

Default (wxHtmlParser) behaviour is this: first it finds a handler capable of handling this tag and then it calls handler's HandleTag() method.

virtual void wxHtmlParser::AddTagHandler (wxHtmlTagHandler * *handler*) `[virtual]`

Adds handler to the internal list (hash table) of handlers.

This method should not be called directly by user but rather by derived class' constructor.

This adds the handler to this **instance** of wxHtmlParser, not to all objects of this class! (Static front-end to AddTagHandler is provided by wxHtmlWinParser).

All handlers are deleted on object deletion.

virtual void wxHtmlParser::AddWord (const wxString & *txt*) `[virtual]`

Must be overwritten in derived class.

This method is called by DoParsing() each time a part of text is parsed. *txt* is NOT only one word, it is substring of input. It is not formatted or preprocessed (so white spaces are unmodified).

virtual void wxHtmlParser::DoneParser ( ) `[virtual]`

This must be called after DoParsing().

void wxHtmlParser::DoParsing (const const_iterator & *begin_pos*, const const_iterator & *end_pos*)

Parses the m_Source from *begin_pos* to *end_pos* - 1.

void wxHtmlParser::DoParsing ( )

Parses the whole m_Source.

wxFileSystem* wxHtmlParser::GetFS ( ) const

Returns pointer to the file system.

Because each tag handler has reference to it is parent parser it can easily request the file by calling:

```
wxFSFile *f = m_Parser -> GetFS() -> OpenFile("image.jpg");
```

virtual wxObject* wxHtmlParser::GetProduct ( ) [pure virtual]

Returns product of parsing.

Returned value is result of parsing of the document.

The type of this result depends on internal representation in derived parser (but it must be derived from wxObject!). See wxHtmlWinParser for details.

const wxString* wxHtmlParser::GetSource ( )

Returns pointer to the source being parsed.

virtual void wxHtmlParser::InitParser (const wxString & *source*) [virtual]

Setups the parser for parsing the *source* string.

(Should be overridden in derived class)

virtual wxFSFile* wxHtmlParser::OpenURL (wxHtmlURLType *type*, const wxString & *url*) const [virtual]

Opens given URL and returns wxFSFile object that can be used to read data from it.

This method may return NULL in one of two cases: either the URL doesn't point to any valid resource or the URL is blocked by overridden implementation of *OpenURL* in derived class.

Parameters

type	Indicates type of the resource. Is one of: • wxHTML_URL_PAGE: Opening a HTML page. • wxHTML_URL_IMAGE: Opening an image. • wxHTML_URL_OTHER: Opening a resource that doesn't fall into any other category.
url	URL being opened.

Note

Always use this method in tag handlers instead of GetFS()->OpenFile() because it can block the URL and is thus more secure. Default behaviour is to call wxHtmlWindow::OnOpeningURL of the associated wxHtmlWindow object (which may decide to block the URL or redirect it to another one),if there's any, and always open the URL if the parser is not used with wxHtmlWindow. Returned wxFSFile object is not guaranteed to point to url, it might have been redirected!

wxObject* wxHtmlParser::Parse (const wxString & *source*)

Proceeds parsing of the document.

This is end-user method. You can simply call it when you need to obtain parsed output (which is parser-specific).

The method does these things:

1. calls InitParser(source)
2. calls DoParsing()
3. calls GetProduct()
4. calls DoneParser()
5. returns value returned by GetProduct()

You shouldn't use InitParser(), DoParsing(), GetProduct() or DoneParser() directly.

void wxHtmlParser::PopTagHandler ( )

Restores parser's state before last call to PushTagHandler().

void wxHtmlParser::PushTagHandler (wxHtmlTagHandler * *handler*, const wxString & *tags*)

Forces the handler to handle additional tags (not returned by wxHtmlTagHandler::GetSupportedTags).

The handler should already be added to this parser.

Parameters

handler	the handler
tags	List of tags (in same format as GetSupportedTags()'s return value). The parser will redirect these tags to handler (until call to PopTagHandler()).

Example:

Imagine you want to parse following pseudo-html structure:

```
<myitems>
    <param name="one" value="1">
    <param name="two" value="2">
</myitems>

<execute>
    <param program="text.exe">
</execute>
```

It is obvious that you cannot use only one tag handler for <param> tag. Instead you must use context-sensitive handlers for <param> inside <myitems> and <param> inside <execute>. This is the preferred solution:

```
TAG_HANDLER_BEGIN(MYITEM, "MYITEMS")
    TAG_HANDLER_PROC(tag)
    {
        // ...something...

        m_Parser -> PushTagHandler(this, "PARAM");
        ParseInner(tag);
        m_Parser -> PopTagHandler();

        // ...something...
    }
TAG_HANDLER_END(MYITEM)
```

void wxHtmlParser::SetFS (wxFileSystem * *fs*)

Sets the virtual file system that will be used to request additional files.

(For example `IMG` tag handler requests wxFSFile with the image data.)

virtual void wxHtmlParser::StopParsing ( ) [virtual]

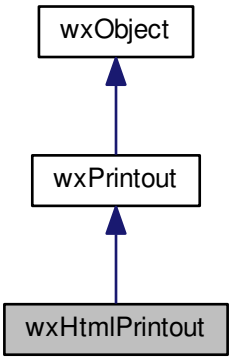
Call this function to interrupt parsing from a tag handler.

No more tags will be parsed afterward. This function may only be called from Parse() or any function called by it (i.e. from tag handlers).

21.373 wxHtmlPrintout Class Reference

`#include <wx/html/htmprint.h>`

Inheritance diagram for wxHtmlPrintout:



21.373.1 Detailed Description

This class serves as printout class for HTML documents.

Library: wxHTML

Category: HTML, Printing Framework

Public Member Functions

- wxHtmlPrintout (const wxString &title="Printout")

  Constructor.
- void SetFonts (const wxString &normal_face, const wxString &fixed_face, const int *sizes=NULL)

  This function sets font sizes and faces.
- void SetFooter (const wxString &footer, int pg=wxPAGE_ALL)

  Set page footer.
- void SetHeader (const wxString &header, int pg=wxPAGE_ALL)

  Set page header.
- void SetHtmlFile (const wxString &htmlfile)

*Prepare the class for printing this HTML **file**.*

- void SetHtmlText (const wxString &html, const wxString &basepath=wxEmptyString, bool isdir=true)

  Prepare the class for printing this HTML text.
- void SetMargins (float top=25.2, float bottom=25.2, float left=25.2, float right=25.2, float spaces=5)

  Sets margins in millimeters.

Static Public Member Functions

- static void AddFilter (wxHtmlFilter *filter)

  Adds a filter to the static list of filters for wxHtmlPrintout.

Additional Inherited Members

21.373.2 Constructor & Destructor Documentation

wxHtmlPrintout::wxHtmlPrintout (const wxString & *title* = "Printout")

Constructor.

21.373.3 Member Function Documentation

static void wxHtmlPrintout::AddFilter (wxHtmlFilter * *filter*) [static]

Adds a filter to the static list of filters for wxHtmlPrintout.

See wxHtmlFilter for further information.

void wxHtmlPrintout::SetFonts (const wxString & *normal_face*, const wxString & *fixed_face*, const int * *sizes* = NULL)

This function sets font sizes and faces.

See wxHtmlWindow::SetFonts for detailed description.

void wxHtmlPrintout::SetFooter (const wxString & *footer*, int *pg* = wxPAGE_ALL)

Set page footer.

The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format
- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document

Parameters

footer	HTML text to be used as footer.
pg	one of wxPAGE_ODD, wxPAGE_EVEN and wxPAGE_ALL constants.

void wxHtmlPrintout::SetHeader (const wxString & *header,* int *pg* = wxPAGE_ALL)

Set page header.

The following macros can be used inside it:

- @DATE@ is replaced by the current date in default format
- @PAGENUM@ is replaced by page number
- @PAGESCNT@ is replaced by total number of pages
- @TIME@ is replaced by the current time in default format
- @TITLE@ is replaced with the title of the document

Parameters

header	HTML text to be used as header.
pg	one of wxPAGE_ODD, wxPAGE_EVEN and wxPAGE_ALL constants.

void wxHtmlPrintout::SetHtmlFile (const wxString & *htmlfile*)

Prepare the class for printing this HTML **file**.

The file may be located on any virtual file system or it may be normal file.

void wxHtmlPrintout::SetHtmlText (const wxString & *html,* const wxString & *basepath* = wxEmptyString, bool *isdir* = `true`)

Prepare the class for printing this HTML text.

Parameters

html	HTML text. (NOT file!)
basepath	base directory (html string would be stored there if it was in file). It is used to determine path for loading images, for example.
isdir	false if basepath is filename, true if it is directory name (see wxFileSystem for detailed explanation).

void wxHtmlPrintout::SetMargins (float *top* = `25.2`, float *bottom* = `25.2`, float *left* = `25.2`, float *right* = `25.2`, float *spaces* = `5`)

Sets margins in millimeters.

Defaults to 1 inch for margins and 0.5cm for space between text and header and/or footer.

21.374 wxHtmlRenderingInfo Class Reference

`#include <wx/html/htmlcell.h>`

21.374.1 Detailed Description

This class contains information given to cells when drawing them.

Contains rendering state, selection information and rendering style object that can be used to customize the output.

Library: wxHTML

Category: HTML

See also

Cells and Containers, wxHtmlCell

Public Member Functions

- wxHtmlRenderingInfo ()

  Default ctor.

- void SetSelection (wxHtmlSelection *s)

  Accessors.
- wxHtmlSelection * GetSelection () const

  Accessors.
- void SetStyle (wxHtmlRenderingStyle *style)

  Accessors.
- wxHtmlRenderingStyle & GetStyle ()

  Accessors.
- wxHtmlRenderingState & GetState ()

  Accessors.

21.374.2 Constructor & Destructor Documentation

wxHtmlRenderingInfo::wxHtmlRenderingInfo ( )

Default ctor.

21.374.3 Member Function Documentation

wxHtmlSelection* wxHtmlRenderingInfo::GetSelection ( ) const

Accessors.

wxHtmlRenderingState& wxHtmlRenderingInfo::GetState ( )

Accessors.

wxHtmlRenderingStyle& wxHtmlRenderingInfo::GetStyle ( )

Accessors.

void wxHtmlRenderingInfo::SetSelection (wxHtmlSelection * *s*)

Accessors.

void wxHtmlRenderingInfo::SetStyle (wxHtmlRenderingStyle * *style*)

Accessors.

21.375 wxHtmlRenderingState Class Reference

`#include <wx/html/htmlcell.h>`

21.375.1 Detailed Description

Selection state is passed to wxHtmlCell::Draw so that it can render itself differently e.g. when inside text selection or outside it.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlRenderingState ()
- void SetSelectionState (wxHtmlSelectionState s)
- wxHtmlSelectionState GetSelectionState () const
- void SetFgColour (const wxColour &c)
- const wxColour & GetFgColour () const
- void SetBgColour (const wxColour &c)
- const wxColour & GetBgColour () const
- void SetBgMode (int m)
- int GetBgMode () const

21.375.2 Constructor & Destructor Documentation

wxHtmlRenderingState::wxHtmlRenderingState ( )

21.375.3 Member Function Documentation

const wxColour& wxHtmlRenderingState::GetBgColour ( ) const

int wxHtmlRenderingState::GetBgMode ( ) const

const wxColour& wxHtmlRenderingState::GetFgColour ( ) const

wxHtmlSelectionState wxHtmlRenderingState::GetSelectionState ( ) const

void wxHtmlRenderingState::SetBgColour (const wxColour & *c*)

void wxHtmlRenderingState::SetBgMode (int *m*)

void wxHtmlRenderingState::SetFgColour (const wxColour & *c*)

void wxHtmlRenderingState::SetSelectionState (wxHtmlSelectionState *s*)

21.376 wxHtmlRenderingStyle Class Reference

`#include <wx/html/htmlcell.h>`

21.376.1 Detailed Description

wxHtmlSelection is data holder with information about text selection.

Allows HTML rendering customizations.

Selection is defined by two positions (beginning and end of the selection) and two leaf(!) cells at these positions.

Library: wxHTML

Category: HTML

This class is used when rendering wxHtmlCells as a callback.

Library: wxHTML

Category: HTML

See also

wxHtmlRenderingInfo

Public Member Functions

- virtual wxColour GetSelectedTextColour (const wxColour &clr)=0

  Returns the colour to use for the selected text.
- virtual wxColour GetSelectedTextBgColour (const wxColour &clr)=0

  Returns the colour to use for the selected text's background.

21.376.2 Member Function Documentation

virtual wxColour wxHtmlRenderingStyle::GetSelectedTextBgColour (const wxColour & *clr*) `[pure virtual]`

Returns the colour to use for the selected text's background.

virtual wxColour wxHtmlRenderingStyle::GetSelectedTextColour (const wxColour & *clr*) `[pure virtual]`

Returns the colour to use for the selected text.

21.377 wxHtmlSelection Class Reference

`#include <wx/html/htmlcell.h>`

Public Member Functions

- wxHtmlSelection ()
- void Set (const wxPoint &fromPos, const wxHtmlCell *fromCell, const wxPoint &toPos, const wxHtmlCell *toCell)
- void Set (const wxHtmlCell *fromCell, const wxHtmlCell *toCell)
- const wxHtmlCell * GetFromCell () const
- const wxHtmlCell * GetToCell () const
- const wxPoint & GetFromPos () const
- const wxPoint & GetToPos () const
- void ClearFromToCharacterPos ()
- bool AreFromToCharacterPosSet () const
- void SetFromCharacterPos (wxCoord pos)
- void SetToCharacterPos (wxCoord pos)
- wxCoord GetFromCharacterPos () const
- wxCoord GetToCharacterPos () const
- bool IsEmpty () const

21.377.1 Constructor & Destructor Documentation

wxHtmlSelection::wxHtmlSelection ( )

21.377.2 Member Function Documentation

bool wxHtmlSelection::AreFromToCharacterPosSet ( ) const

void wxHtmlSelection::ClearFromToCharacterPos ( )

const wxHtmlCell* wxHtmlSelection::GetFromCell ( ) const

wxCoord wxHtmlSelection::GetFromCharacterPos ( ) const

const wxPoint& wxHtmlSelection::GetFromPos ( ) const

const wxHtmlCell* wxHtmlSelection::GetToCell ( ) const

wxCoord wxHtmlSelection::GetToCharacterPos ( ) const

const wxPoint& wxHtmlSelection::GetToPos ( ) const

bool wxHtmlSelection::IsEmpty ( ) const

void wxHtmlSelection::Set (const wxPoint & *fromPos,* const wxHtmlCell * *fromCell,* const wxPoint & *toPos,* const wxHtmlCell * *toCell*)

void wxHtmlSelection::Set (const wxHtmlCell * *fromCell,* const wxHtmlCell * *toCell*)

void wxHtmlSelection::SetFromCharacterPos (wxCoord *pos*)

void wxHtmlSelection::SetToCharacterPos (wxCoord *pos*)

21.378 wxHtmlTag Class Reference

`#include <wx/html/htmltag.h>`

21.378.1 Detailed Description

This class represents a single HTML tag.

It is used by tag handlers.

Library: wxHTML

Category: HTML

Public Member Functions

- wxString GetAllParams () const

  Returns a string containing all parameters.
- int GetBeginPos () const

  Returns beginning position of the text between this tag and paired ending tag.
- int GetEndPos1 () const

  Returns ending position of the text between this tag and paired ending tag.
- int GetEndPos2 () const

  Returns ending position 2 of the text between this tag and paired ending tag.
- wxString GetName () const

  Returns tag's name.
- wxString GetParam (const wxString &par, bool with_quotes=false) const

  Returns the value of the parameter.
- bool GetParamAsColour (const wxString &par, wxColour ∗clr) const

  Interprets tag parameter par as colour specification and saves its value into wxColour variable pointed by clr.
- bool GetParamAsInt (const wxString &par, int ∗value) const

  Interprets tag parameter par as an integer and saves its value into int variable pointed by value.
- bool GetParamAsString (const wxString &par, wxString ∗value) const

  Get the value of the parameter.
- bool HasEnding () const

  Returns true if this tag is paired with ending tag, false otherwise.
- bool HasParam (const wxString &par) const

  Returns true if the tag has a parameter of the given name.
- int ScanParam (const wxString &par, const wchar_t ∗format, void ∗value) const

  This method scans the given parameter.
- int ScanParam (const wxString &par, const char ∗format, void ∗value) const

  This method scans the given parameter.

Static Public Member Functions

- static bool ParseAsColour (const wxString &str, wxColour ∗clr)

  Parses the given string as an HTML colour.

Protected Member Functions

- wxHtmlTag (wxHtmlTag *parent, const wxString *source, const const_iterator &pos, const const_iterator &end_pos, wxHtmlTagsCache *cache, wxHtmlEntitiesParser *entParser)

  Constructor.

21.378.2 Constructor & Destructor Documentation

wxHtmlTag::wxHtmlTag (wxHtmlTag * *parent,* const wxString * *source,* const const_iterator & *pos,* const const_iterator & *end_pos,* wxHtmlTagsCache * *cache,* wxHtmlEntitiesParser * *entParser*) `[protected]`

Constructor.

You will probably never have to construct a wxHtmlTag object yourself. Feel free to ignore the constructor parameters. Have a look at `src/html/htmlpars.cpp` if you're interested in creating it.

21.378.3 Member Function Documentation

wxString wxHtmlTag::GetAllParams ( ) const

Returns a string containing all parameters.

Example: tag contains <FONT SIZE=+2 COLOR="#000000">. Call to tag.GetAllParams() would return `'SIZE=+2 COLOR="#000000"'`.

int wxHtmlTag::GetBeginPos ( ) const

Returns beginning position of the text *between* this tag and paired ending tag.

See explanation (returned position is marked with '|'):

```
bla bla bla <MYTAG> bla bla internal text</MYTAG> bla bla
                  |
```

Deprecated

Todo provide deprecation description

int wxHtmlTag::GetEndPos1 ( ) const

Returns ending position of the text *between* this tag and paired ending tag.

See explanation (returned position is marked with '|'):

```
bla bla bla <MYTAG> bla bla internal text</MYTAG> bla bla
                                          |
```

Deprecated

Todo provide deprecation description

int wxHtmlTag::GetEndPos2 ( ) const

Returns ending position 2 of the text *between* this tag and paired ending tag.

See explanation (returned position is marked with '|'):

```
bla bla bla <MYTAG> bla bla internal text</MYTAG> bla bla
                                                |
```

Deprecated

Todo provide deprecation description

wxString wxHtmlTag::GetName ( ) const

Returns tag's name.

The name is always in uppercase and it doesn't contain " or '/' characters. (So the name of <FONT SIZE=+2> tag is "FONT" and name of </table> is "TABLE").

wxString wxHtmlTag::GetParam (const wxString & *par*, bool *with_quotes* = `false`) const

Returns the value of the parameter.

You should check whether the parameter exists or not (use wxHtmlTag::HasParam) first or use GetParamAsString() if you need to distinguish between non-specified and empty parameter values.

Parameters

par	The parameter's name.
with_quotes	true if you want to get quotes as well. See example.

Example:

```
...
// you have wxHtmlTag variable tag which is equal to the
// HTML tag <FONT SIZE=+2 COLOR="#0000FF">
dummy = tag.GetParam("SIZE");
// dummy == "+2"
dummy = tag.GetParam("COLOR");
// dummy == "#0000FF"
dummy = tag.GetParam("COLOR", true);
// dummy == "\"#0000FF\"" -- see the difference!!
```

bool wxHtmlTag::GetParamAsColour (const wxString & *par*, wxColour * *clr*) const

Interprets tag parameter *par* as colour specification and saves its value into wxColour variable pointed by *clr*.

Returns true on success and false if *par* is not colour specification or if the tag has no such parameter.

See also

ParseAsColour()

bool wxHtmlTag::GetParamAsInt (const wxString & *par*, int * *value*) const

Interprets tag parameter *par* as an integer and saves its value into int variable pointed by *value*.

Returns true on success and false if *par* is not an integer or if the tag has no such parameter.

bool wxHtmlTag::GetParamAsString (const wxString & *par,* wxString * *value*) const

Get the value of the parameter.

If the tag doesn't have such parameter at all, simply returns false. Otherwise, fills *value* with the parameter value and returns true.

Parameters

par	The parameter's name.
value	Pointer to the string to be filled with the parameter value, must be non-NULL.

Since

3.0

bool wxHtmlTag::HasEnding ( ) const

Returns true if this tag is paired with ending tag, false otherwise.

See the example of HTML document:

```
<html><body>
Hello<p>
How are you?
<p align=center>This is centered...</p>
Oops<br>Oooops!
</body></html>
```

In this example tags HTML and BODY have ending tags, first P and BR doesn't have ending tag while the second P has. The third P tag (which is ending itself) of course doesn't have ending tag.

bool wxHtmlTag::HasParam (const wxString & *par*) const

Returns true if the tag has a parameter of the given name.

Example: <FONT SIZE=+2 COLOR="\#FF00FF"> has two parameters named "SIZE" and "COLOR".

Parameters

par	the parameter you're looking for.

static bool wxHtmlTag::ParseAsColour (const wxString & *str,* wxColour * *clr*) [static]

Parses the given string as an HTML colour.

This function recognizes the standard named HTML 4 colours as well as the usual RGB syntax.

Since

2.9.1

See also

wxColour::Set()

Returns

true if the string was successfully parsed and *clr* was filled with the result or false otherwise.

int wxHtmlTag::ScanParam (const wxString & *par,* const wchar_t * *format,* void * *value*) const

This method scans the given parameter.

Usage is exactly the same as sscanf's usage except that you don't pass a string but a parameter name as the first argument and you can only retrieve one value (i.e. you can use only one "%" element in *format*).

Parameters

par	The name of the tag you want to query
format	scanf()-like format string.
value	pointer to a variable to store the value in

int wxHtmlTag::ScanParam (const wxString & *par,* const char * *format,* void * *value*) const

This method scans the given parameter.

Usage is exactly the same as sscanf's usage except that you don't pass a string but a parameter name as the first argument and you can only retrieve one value (i.e. you can use only one "%" element in *format*).

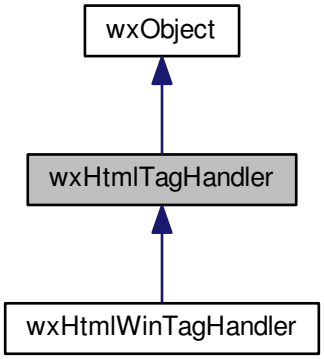
Parameters

par	The name of the tag you want to query
format	scanf()-like format string.
value	pointer to a variable to store the value in

21.379 wxHtmlTagHandler Class Reference

`#include <wx/html/htmlpars.h>`

Inheritance diagram for wxHtmlTagHandler:



21.379.1 Detailed Description

Todo describe me

Library: wxHTML

Category: HTML

See also

Tag Handlers, wxHtmlTag

Public Member Functions

- wxHtmlTagHandler ()

  Constructor.
- virtual wxString GetSupportedTags ()=0

  Returns list of supported tags.
- virtual bool HandleTag (const wxHtmlTag &tag)=0

  This is the core method of each handler.
- virtual void SetParser (wxHtmlParser *parser)

  Assigns parser to this handler.
- wxHtmlParser * GetParser () const

  Returns the parser associated with this tag handler.

Protected Member Functions

- void ParseInner (const wxHtmlTag &tag)

  This method calls parser's wxHtmlParser::DoParsing method for the string between this tag and the paired ending tag:
- void ParseInnerSource (const wxString &source)

  Parses given source as if it was tag's inner code (see wxHtmlParser::GetInnerSource).

Protected Attributes

- wxHtmlParser * m_Parser

  This attribute is used to access parent parser.

21.379.2 Constructor & Destructor Documentation

wxHtmlTagHandler::wxHtmlTagHandler ( )

Constructor.

21.379.3 Member Function Documentation

wxHtmlParser* wxHtmlTagHandler::GetParser ( ) const

Returns the parser associated with this tag handler.

Since

2.9.5

virtual wxString wxHtmlTagHandler::GetSupportedTags ( ) [pure virtual]

Returns list of supported tags.

The list is in uppercase and tags are delimited by ','. Example: `"I,B,FONT,P"`

virtual bool wxHtmlTagHandler::HandleTag (const wxHtmlTag & *tag*) [pure virtual]

This is the core method of each handler.

It is called each time one of supported tags is detected. *tag* contains all necessary info (see wxHtmlTag for details).

Example:

```
bool MyHandler::HandleTag(const wxHtmlTag& tag)
{
    ...
    // change state of parser (e.g. set bold face)
    ParseInner(tag);
    ...
    // restore original state of parser
}
```

You shouldn't call ParseInner() if the tag is not paired with an ending one.

Returns

true if ParseInner() was called, false otherwise.

void wxHtmlTagHandler::ParseInner (const wxHtmlTag & *tag*) [protected]

This method calls parser's wxHtmlParser::DoParsing method for the string between this tag and the paired ending tag:

```
...<A HREF="x.htm">Hello, world!</A>...
```

In this example, a call to ParseInner() (with *tag* pointing to A tag) will parse 'Hello, world!'.

void wxHtmlTagHandler::ParseInnerSource (const wxString & *source*) [protected]

Parses given source as if it was tag's inner code (see wxHtmlParser::GetInnerSource).

Unlike ParseInner(), this method lets you specify the source code to parse. This is useful when you need to modify the inner text before parsing.

virtual void wxHtmlTagHandler::SetParser (wxHtmlParser * *parser*) [virtual]

Assigns *parser* to this handler.

Each **instance** of handler is guaranteed to be called only from the one parser.

21.379.4 Member Data Documentation

wxHtmlParser* wxHtmlTagHandler::m_Parser [protected]

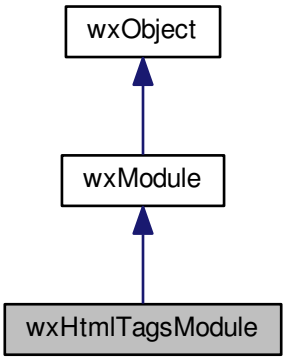
This attribute is used to access parent parser.

It is protected so that it can't be accessed by user but can be accessed from derived classes.

21.380 wxHtmlTagsModule Class Reference

`#include <wx/html/winpars.h>`

Inheritance diagram for wxHtmlTagsModule:



21.380.1 Detailed Description

This class provides easy way of filling wxHtmlWinParser's table of tag handlers.

It is used almost exclusively together with the set of TAGS_MODULE_* macros

Library: wxHTML

Category: HTML

See also

Tag Handlers, wxHtmlTagHandler, wxHtmlWinTagHandler

Public Member Functions

- virtual void FillHandlersTable (wxHtmlWinParser *parser)

  You must override this method.

Additional Inherited Members

21.380.2 Member Function Documentation

virtual void wxHtmlTagsModule::FillHandlersTable (wxHtmlWinParser * *parser*) `[virtual]`

You must override this method.

In most common case its body consists only of lines of the following type:

```
parser -> AddTagHandler(new MyHandler);
```

It's recommended to use the **TAGS_MODULE_*** macros.

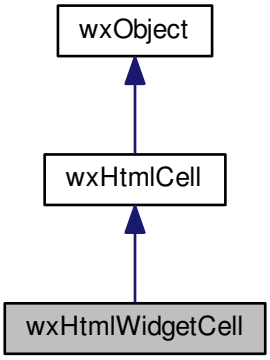
Parameters

parser	Pointer to the parser that requested tables filling.

21.381 wxHtmlWidgetCell Class Reference

`#include <wx/html/htmlcell.h>`

Inheritance diagram for wxHtmlWidgetCell:



21.381.1 Detailed Description

wxHtmlWidgetCell is a class that provides a connection between HTML cells and widgets (an object derived from wxWindow).

You can use it to display things like forms, input boxes etc. in an HTML window.

wxHtmlWidgetCell takes care of resizing and moving window.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlWidgetCell (wxWindow *wnd, int w=0)

  Constructor.

Additional Inherited Members

21.381.2 Constructor & Destructor Documentation

wxHtmlWidgetCell::wxHtmlWidgetCell (wxWindow * *wnd*, int *w* = 0)

Constructor.

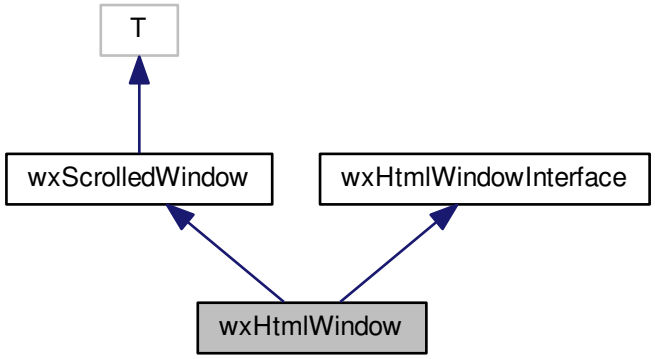
Parameters

wnd	Connected window. It is parent window **must** be the wxHtmlWindow object within which it is displayed!
w	Floating width. If non-zero width of wnd window is adjusted so that it is always w percents of parent container's width. (For example w = 100 means that the window will always have same width as parent container).

21.382 wxHtmlWindow Class Reference

`#include <wx/html/htmlwin.h>`

Inheritance diagram for wxHtmlWindow:



21.382.1 Detailed Description

wxHtmlWindow is probably the only class you will directly use unless you want to do something special (like adding new tag handlers or MIME filters).

The purpose of this class is to display rich content pages (either local file or downloaded via HTTP protocol) in a window based on a subset of the HTML standard. The width of the window is constant - given in the constructor - and virtual height is changed dynamically depending on page size. Once the window is created you can set its content by calling SetPage() with raw HTML, LoadPage() with a wxFileSystem location or LoadFile() with a filename.

Note

If you want complete HTML/CSS support as well as a Javascript engine, see instead wxWebView.
wxHtmlWindow uses the wxImage class for displaying images, as such you need to initialize the handlers for any image formats you use before loading a page. See wxInitAllImageHandlers and wxImage::AddHandler.

Styles

This class supports the following styles:

- wxHW_SCROLLBAR_NEVER: Never display scrollbars, not even when the page is larger than the window.
- wxHW_SCROLLBAR_AUTO: Display scrollbars only if page's size exceeds window's size.

- wxHW_NO_SELECTION: Don't allow the user to select text.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHtmlCellEvent& event) or void handlerFuncName(wxHtmlLinkEvent& event)

Event macros for events emitted by this class:

- EVT_HTML_CELL_CLICKED(id, func): A wxHtmlCell was clicked.
- EVT_HTML_CELL_HOVER(id, func): The mouse passed over a wxHtmlCell.
- EVT_HTML_LINK_CLICKED(id, func): A wxHtmlCell which contains an hyperlink was clicked.

Library: wxHTML

Category: HTML

See also

wxHtmlLinkEvent, wxHtmlCellEvent

Public Member Functions

- wxHtmlWindow ()

  Default ctor.
- wxHtmlWindow (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHW_DEFAULT_STYLE, const wxString &name="html↩ Window")

  Constructor.
- bool AppendToPage (const wxString &source)

  Appends HTML fragment to currently displayed text and refreshes the window.
- wxHtmlContainerCell * GetInternalRepresentation () const

  Returns pointer to the top-level container.
- wxString GetOpenedAnchor () const

  Returns anchor within currently opened page (see wxHtmlWindow::GetOpenedPage).
- wxString GetOpenedPage () const

  Returns full location of the opened page.
- wxString GetOpenedPageTitle () const

  Returns title of the opened page or wxEmptyString if the current page does not contain <TITLE> tag.
- wxFrame * GetRelatedFrame () const

  Returns the related frame.
- bool HistoryBack ()

  Moves back to the previous page.
- bool HistoryCanBack ()

  Returns true if it is possible to go back in the history i.e.
- bool HistoryCanForward ()

  Returns true if it is possible to go forward in the history i.e.
- void HistoryClear ()

  Clears history.
- bool HistoryForward ()

Moves to next page in history.

- bool LoadFile (const wxFileName &filename)

  Loads an HTML page from a file and displays it.
- virtual bool LoadPage (const wxString &location)

  Unlike SetPage() this function first loads the HTML page from location and then displays it.
- virtual void OnLinkClicked (const wxHtmlLinkInfo &link)

  Called when user clicks on hypertext link.
- virtual wxHtmlOpeningStatus OnOpeningURL (wxHtmlURLType type, const wxString &url, wxString *redirect) const

  Called when an URL is being opened (either when the user clicks on a link or an image is loaded).
- virtual void OnSetTitle (const wxString &title)

  Called on parsing <TITLE> tag.
- virtual void ReadCustomization (wxConfigBase *cfg, wxString path=wxEmptyString)

  This reads custom settings from wxConfig.
- void SelectAll ()

  Selects all text in the window.
- void SelectLine (const wxPoint &pos)

  Selects the line of text that pos points at.
- void SelectWord (const wxPoint &pos)

  Selects the word at position pos.
- wxString SelectionToText ()

  Returns the current selection as plain text.
- void SetBorders (int b)

  This function sets the space between border of window and HTML contents.
- void SetFonts (const wxString &normal_face, const wxString &fixed_face, const int *sizes=NULL)

  This function sets font sizes and faces.
- void SetStandardFonts (int size=-1, const wxString &normal_face=wxEmptyString, const wxString &fixed_face=wxEmptyString)

  Sets default font sizes and/or default font size.
- virtual bool SetPage (const wxString &source)

  Sets the source of a page and displays it, for example:
- void SetRelatedFrame (wxFrame *frame, const wxString &format)

  Sets the frame in which page title will be displayed.
- void SetRelatedStatusBar (int index)

  ***After** calling SetRelatedFrame(), this sets statusbar slot where messages will be displayed.*
- void SetRelatedStatusBar (wxStatusBar *statusbar, int index=0)

  ***Sets** the associated statusbar where messages will be displayed.*
- wxString ToText ()

  Returns content of currently displayed page as plain text.
- virtual void WriteCustomization (wxConfigBase *cfg, wxString path=wxEmptyString)

  Saves custom settings into wxConfig.

Static Public Member Functions

- static void AddFilter (wxHtmlFilter *filter)

  Adds input filter to the static list of available filters.
- static wxCursor GetDefaultHTMLCursor (HTMLCursor type)

  Retrieves the default cursor for a given HTMLCursor type.
- static void SetDefaultHTMLCursor (HTMLCursor type, const wxCursor &cursor)

  Sets the default cursor for a given HTMLCursor type.

Protected Member Functions

- virtual bool OnCellClicked (wxHtmlCell *cell, wxCoord x, wxCoord y, const wxMouseEvent &event)

  This method is called when a mouse button is clicked inside wxHtmlWindow.
- virtual void OnCellMouseHover (wxHtmlCell *cell, wxCoord x, wxCoord y)

  This method is called when a mouse moves over an HTML cell.

Additional Inherited Members

21.382.2 Constructor & Destructor Documentation

wxHtmlWindow::wxHtmlWindow ( )

Default ctor.

wxHtmlWindow::wxHtmlWindow (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHW_DEFAULT_STYLE, const wxString & *name* = "htmlWindow")

Constructor.

The parameters are the same as wxScrolled::wxScrolled() constructor.

21.382.3 Member Function Documentation

static void wxHtmlWindow::AddFilter (wxHtmlFilter * *filter*) [static]

Adds input filter to the static list of available filters.

These filters are present by default:

- text/html MIME type
- image/* MIME types
- Plain Text filter (this filter is used if no other filter matches)

bool wxHtmlWindow::AppendToPage (const wxString & *source*)

Appends HTML fragment to currently displayed text and refreshes the window.

Parameters

source	HTML code fragment

Returns

false if an error occurred, true otherwise.

static wxCursor wxHtmlWindow::GetDefaultHTMLCursor (HTMLCursor *type*) [static]

Retrieves the default cursor for a given HTMLCursor type.

Parameters

type	HTMLCursor type to retrieve.

Since

3.1.0

wxHtmlContainerCell* **wxHtmlWindow::GetInternalRepresentation ( ) const**

Returns pointer to the top-level container.

See also

Cells and Containers, Printing Framework Overview

wxString wxHtmlWindow::GetOpenedAnchor ( ) const

Returns anchor within currently opened page (see wxHtmlWindow::GetOpenedPage).

If no page is opened or if the displayed page wasn't produced by call to LoadPage(), empty string is returned.

wxString wxHtmlWindow::GetOpenedPage ( ) const

Returns full location of the opened page.

If no page is opened or if the displayed page wasn't produced by call to LoadPage(), empty string is returned.

wxString wxHtmlWindow::GetOpenedPageTitle ( ) const

Returns title of the opened page or wxEmptyString if the current page does not contain <TITLE> tag.

wxFrame* **wxHtmlWindow::GetRelatedFrame ( ) const**

Returns the related frame.

bool wxHtmlWindow::HistoryBack ( )

Moves back to the previous page.

Only pages displayed using LoadPage() are stored in history list.

bool wxHtmlWindow::HistoryCanBack ( )

Returns true if it is possible to go back in the history i.e.

HistoryBack() won't fail.

bool wxHtmlWindow::HistoryCanForward ( )

Returns true if it is possible to go forward in the history i.e.

HistoryForward() won't fail.

void wxHtmlWindow::HistoryClear ( )

Clears history.

bool wxHtmlWindow::HistoryForward ( )

Moves to next page in history.

Only pages displayed using LoadPage() are stored in history list.

bool wxHtmlWindow::LoadFile (const wxFileName & *filename*)

Loads an HTML page from a file and displays it.

Returns

false if an error occurred, true otherwise

See also

LoadPage()

virtual bool wxHtmlWindow::LoadPage (const wxString & *location*) [virtual]

Unlike SetPage() this function first loads the HTML page from *location* and then displays it.

Parameters

location	The address of the document. See the wxFileSystem Overview for details on the address format and wxFileSystem for a description of how the file is opened.

Returns

false if an error occurred, true otherwise

See also

LoadFile()

virtual bool wxHtmlWindow::OnCellClicked (wxHtmlCell * *cell*, wxCoord *x*, wxCoord *y*, const wxMouseEvent & *event*) [protected], [virtual]

This method is called when a mouse button is clicked inside wxHtmlWindow.

The default behaviour is to emit a wxHtmlCellEvent and, if the event was not processed or skipped, call OnLink↩Clicked() if the cell contains an hypertext link.

Overloading this method is deprecated; intercept the event instead.

Parameters

cell	The cell inside which the mouse was clicked, always a simple (i.e. non-container) cell

x	The logical x coordinate of the click point
y	The logical y coordinate of the click point
event	The mouse event containing other information about the click

Returns

true if a link was clicked, false otherwise.

virtual void wxHtmlWindow::OnCellMouseHover (wxHtmlCell * *cell*, wxCoord *x*, wxCoord *y*) [protected], [virtual]

This method is called when a mouse moves over an HTML cell.

Default behaviour is to emit a wxHtmlCellEvent.

Overloading this method is deprecated; intercept the event instead.

Parameters

cell	The cell inside which the mouse is currently, always a simple (i.e. non-container) cell
x	The logical x coordinate of the click point
y	The logical y coordinate of the click point

virtual void wxHtmlWindow::OnLinkClicked (const wxHtmlLinkInfo & *link*) [virtual]

Called when user clicks on hypertext link.

Default behaviour is to emit a wxHtmlLinkEvent and, if the event was not processed or skipped, call LoadPage() and do nothing else.

Overloading this method is deprecated; intercept the event instead.

Also see wxHtmlLinkInfo.

virtual wxHtmlOpeningStatus wxHtmlWindow::OnOpeningURL (wxHtmlURLType *type*, const wxString & *url*, wxString * *redirect*) const [virtual]

Called when an URL is being opened (either when the user clicks on a link or an image is loaded).

The URL will be opened only if OnOpeningURL() returns `wxHTML_OPEN`. This method is called by wxHtmlParser::OpenURL.

You can override OnOpeningURL() to selectively block some URLs (e.g. for security reasons) or to redirect them elsewhere. Default behaviour is to always return `wxHTML_OPEN`.

Parameters

type	Indicates type of the resource. Is one of • wxHTML_URL_PAGE: Opening a HTML page. • wxHTML_URL_IMAGE: Opening an image. • wxHTML_URL_OTHER: Opening a resource that doesn't fall into any other category.

url	URL being opened.
redirect	Pointer to wxString variable that must be filled with an URL if OnOpeningURL() returns `wx↩HTML_REDIRECT`.

The return value is:

- wxHTML_OPEN: Open the URL.
- wxHTML_BLOCK: Deny access to the URL, wxHtmlParser::OpenURL will return NULL.
- wxHTML_REDIRECT: Don't open url, redirect to another URL. OnOpeningURL() must fill *redirect with the new URL. OnOpeningURL() will be called again on returned URL.

virtual void wxHtmlWindow::OnSetTitle (const wxString & *title*) [virtual]

Called on parsing <TITLE> tag.

virtual void wxHtmlWindow::ReadCustomization (wxConfigBase * *cfg*, wxString *path* = wxEmptyString) [virtual]

This reads custom settings from wxConfig.

It uses the path 'path' if given, otherwise it saves info into currently selected path. The values are stored in sub-path `wxHtmlWindow`. Read values: all things set by SetFonts(), SetBorders().

Parameters

cfg	wxConfig from which you want to read the configuration.
path	Optional path in config tree. If not given current path is used.

void wxHtmlWindow::SelectAll ( )

Selects all text in the window.

See also

SelectLine(), SelectWord()

wxString wxHtmlWindow::SelectionToText ( )

Returns the current selection as plain text.

Returns an empty string if no text is currently selected.

void wxHtmlWindow::SelectLine (const wxPoint & *pos*)

Selects the line of text that *pos* points at.

Note that *pos* is relative to the top of displayed page, not to window's origin, use wxScrolled::CalcUnscrolled↩Position() to convert physical coordinate.

See also

SelectAll(), SelectWord()

void wxHtmlWindow::SelectWord (const wxPoint & *pos*)

Selects the word at position *pos*.

Note that *pos* is relative to the top of displayed page, not to window's origin, use wxScrolled::CalcUnscrolled↩ Position() to convert physical coordinate.

See also

SelectAll(), SelectLine()

void wxHtmlWindow::SetBorders (int *b*)

This function sets the space between border of window and HTML contents.

See image:

Parameters

b	indentation from borders in pixels

static void wxHtmlWindow::SetDefaultHTMLCursor (HTMLCursor *type*, const wxCursor & *cursor*) [static]

Sets the default cursor for a given HTMLCursor type.

These cursors are used for all wxHtmlWindow objects by default, but can be overridden on a per-window basis.

Parameters

type	HTMLCursor type to retrieve.
cursor	The default cursor for the specified cursor type.

Since

3.1.0

void wxHtmlWindow::SetFonts (const wxString & *normal_face*, const wxString & *fixed_face*, const int * *sizes* = NULL)

This function sets font sizes and faces.

See wxHtmlDCRenderer::SetFonts for detailed description.

See also

SetSize()

virtual bool wxHtmlWindow::SetPage (const wxString & *source*) [virtual]

Sets the source of a page and displays it, for example:

```
htmlwin -> SetPage("<html><body>Hello, world!</body></html>");
```

If you want to load a document from some location use LoadPage() instead.

Parameters

source	The HTML to be displayed.

Returns

false if an error occurred, true otherwise.

void wxHtmlWindow::SetRelatedFrame (wxFrame * *frame*, const wxString & *format*)

Sets the frame in which page title will be displayed.

format is the format of the frame title, e.g. "HtmlHelp : %s". It must contain exactly one s. This s is substituted with HTML page title.

void wxHtmlWindow::SetRelatedStatusBar (int *index*)

After calling SetRelatedFrame(), this sets statusbar slot where messages will be displayed.

(Default is -1 = no messages.)

Parameters

index	Statusbar slot number (0..n)

void wxHtmlWindow::SetRelatedStatusBar (wxStatusBar * *statusbar*, int *index* = 0)

Sets the associated statusbar where messages will be displayed.

Call this instead of SetRelatedFrame() if you want statusbar updates only, no changing of the frame title.

Parameters

statusbar	Statusbar pointer
index	Statusbar slot number (0..n)

Since

2.9.0

void wxHtmlWindow::SetStandardFonts (int *size* = -1, const wxString & *normal_face* = wxEmptyString, const wxString & *fixed_face* = wxEmptyString)

Sets default font sizes and/or default font size.

See wxHtmlDCRenderer::SetStandardFonts for detailed description.

See also

SetFonts()

wxString wxHtmlWindow::ToText ( )

Returns content of currently displayed page as plain text.

virtual void wxHtmlWindow::WriteCustomization (wxConfigBase * *cfg*, wxString *path* = wxEmptyString) [virtual]

Saves custom settings into wxConfig.

It uses the path 'path' if given, otherwise it saves info into currently selected path. Regardless of whether the path is given or not, the function creates sub-path wxHtmlWindow.

Saved values: all things set by SetFonts(), SetBorders().

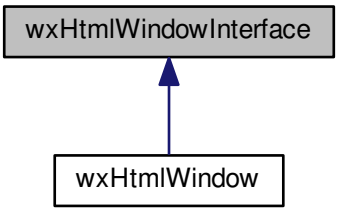
Parameters

cfg	wxConfig to which you want to save the configuration.
path	Optional path in config tree. If not given, the current path is used.

21.383 wxHtmlWindowInterface Class Reference

#include <wx/html/htmlwin.h>

Inheritance diagram for wxHtmlWindowInterface:



21.383.1 Detailed Description

Abstract interface to a HTML rendering window (such as wxHtmlWindow or wxHtmlListBox) that is passed to wxHtmlWinParser.

It encapsulates all communication from the parser to the window.

Public Types

- enum HTMLCursor {
  HTMLCursor_Default,
  HTMLCursor_Link,
  HTMLCursor_Text }

  Type of mouse cursor.

Public Member Functions

- wxHtmlWindowInterface ()

  Ctor.
- virtual ~wxHtmlWindowInterface ()

- virtual void SetHTMLWindowTitle (const wxString &title)=0

  Called by the parser to set window's title to given text.
- virtual void OnHTMLLinkClicked (const wxHtmlLinkInfo &link)=0

  Called when a link is clicked.
- virtual wxHtmlOpeningStatus OnHTMLOpeningURL (wxHtmlURLType type, const wxString &url, wxString *redirect) const =0

  Called when the parser needs to open another URL (e.g.
- virtual wxPoint HTMLCoordsToWindow (wxHtmlCell *cell, const wxPoint &pos) const =0

  Converts coordinates pos relative to given cell to physical coordinates in the window.
- virtual wxWindow * GetHTMLWindow ()=0

  Returns the window used for rendering (may be NULL).
- virtual wxColour GetHTMLBackgroundColour () const =0

  Returns background colour to use by default.
- virtual void SetHTMLBackgroundColour (const wxColour &clr)=0

  Sets window's background to colour clr.
- virtual void SetHTMLBackgroundImage (const wxBitmap &bmpBg)=0

  Sets window's background to given bitmap.
- virtual void SetHTMLStatusText (const wxString &text)=0

  Sets status bar text.
- virtual wxCursor GetHTMLCursor (wxHtmlWindowInterface::HTMLCursor type) const =0

  Returns mouse cursor of given type.

21.383.2 Member Enumeration Documentation

enum wxHtmlWindowInterface::HTMLCursor

Type of mouse cursor.

Enumerator

HTMLCursor_Default Standard mouse cursor (typically an arrow)

HTMLCursor_Link Cursor shown over links.

HTMLCursor_Text Cursor shown over selectable text.

21.383.3 Constructor & Destructor Documentation

wxHtmlWindowInterface::wxHtmlWindowInterface ( )

Ctor.

virtual wxHtmlWindowInterface::~wxHtmlWindowInterface ( ) [virtual]

21.383.4 Member Function Documentation

virtual wxColour wxHtmlWindowInterface::GetHTMLBackgroundColour ( ) const [pure virtual]

Returns background colour to use by default.

virtual wxCursor wxHtmlWindowInterface::GetHTMLCursor (wxHtmlWindowInterface::HTMLCursor *type*) const [pure virtual]

Returns mouse cursor of given *type*.

virtual wxWindow* wxHtmlWindowInterface::GetHTMLWindow ( ) [pure virtual]

Returns the window used for rendering (may be NULL).

virtual wxPoint wxHtmlWindowInterface::HTMLCoordsToWindow (wxHtmlCell * *cell,* const wxPoint & *pos*) const [pure virtual]

Converts coordinates *pos* relative to given *cell* to physical coordinates in the window.

virtual void wxHtmlWindowInterface::OnHTMLLinkClicked (const wxHtmlLinkInfo & *link*) [pure virtual]

Called when a link is clicked.

Parameters

link	information about the clicked link

virtual wxHtmlOpeningStatus wxHtmlWindowInterface::OnHTMLOpeningURL (wxHtmlURLType *type,* const wxString & *url,* wxString * *redirect*) const [pure virtual]

Called when the parser needs to open another URL (e.g.

an image).

Parameters

type	Type of the URL request (e.g. image)
url	URL the parser wants to open
redirect	If the return value is wxHTML_REDIRECT, then the URL to redirect to will be stored in this variable (the pointer must never be NULL)

Returns

indicator of how to treat the request

virtual void wxHtmlWindowInterface::SetHTMLBackgroundColour (const wxColour & *clr*) [pure virtual]

Sets window's background to colour *clr*.

virtual void wxHtmlWindowInterface::SetHTMLBackgroundImage (const wxBitmap & *bmpBg*) [pure virtual]

Sets window's background to given bitmap.

virtual void wxHtmlWindowInterface::SetHTMLStatusText (const wxString & *text*) [pure virtual]

Sets status bar text.

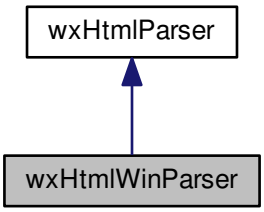
virtual void wxHtmlWindowInterface::SetHTMLWindowTitle (const wxString & *title*) [pure virtual]

Called by the parser to set window's title to given text.

21.384 wxHtmlWinParser Class Reference

`#include <wx/html/winpars.h>`

Inheritance diagram for wxHtmlWinParser:



21.384.1 Detailed Description

This class is derived from wxHtmlParser and its main goal is to parse HTML input so that it can be displayed in wxHtmlWindow.

It uses a special wxHtmlWinTagHandler.

Note

The product of parsing is a wxHtmlCell (resp. wxHtmlContainer) object.

Library: wxHTML

Category: HTML

See also

Tag Handlers

Public Member Functions

- wxHtmlWinParser (wxHtmlWindowInterface *wndIface=NULL)

  Constructor.
- wxHtmlContainerCell * CloseContainer ()

  Closes the container, sets actual container to the parent one and returns pointer to it (see Cells and Containers).
- virtual wxFont * CreateCurrentFont ()

  Creates font based on current setting (see SetFontSize(), SetFontBold(), SetFontItalic(), SetFontFixed(), wxHtmlWinParser::SetFontUnderlined) and returns pointer to it.
- const wxColour & GetActualColor () const

  Returns actual text colour.
- int GetAlign () const

  Returns default horizontal alignment.
- int GetCharHeight () const

Returns (average) char height in standard font.

- int GetCharWidth () const

  Returns average char width in standard font.

- wxHtmlContainerCell * GetContainer () const

  Returns pointer to the currently opened container (see Cells and Containers).

- wxDC * GetDC ()

  Returns pointer to the DC used during parsing.

- wxEncodingConverter * GetEncodingConverter () const

  Returns wxEncodingConverter class used to do conversion between the input encoding and the output encoding.

- int GetFontBold () const

  Returns true if actual font is bold, false otherwise.

- wxString GetFontFace () const

  Returns actual font face name.

- int GetFontFixed () const

  Returns true if actual font is fixed face, false otherwise.

- int GetFontItalic () const

  Returns true if actual font is italic, false otherwise.

- int GetFontSize () const

  Returns actual font size (HTML size varies from -2 to +4)

- int GetFontUnderlined () const

  Returns true if actual font is underlined, false otherwise.

- wxFontEncoding GetInputEncoding () const

  Returns input encoding.

- const wxHtmlLinkInfo & GetLink () const

  Returns actual hypertext link.

- const wxColour & GetLinkColor () const

  Returns the colour of hypertext link text.

- wxFontEncoding GetOutputEncoding () const

  Returns output encoding, i.e.

- wxHtmlWindowInterface * GetWindowInterface ()

  Returns associated window (wxHtmlWindow).

- wxHtmlContainerCell * OpenContainer ()

  Opens new container and returns pointer to it (see Cells and Containers).

- void SetActualColor (const wxColour &clr)

  Sets actual text colour.

- void SetAlign (int a)

  Sets default horizontal alignment (see wxHtmlContainerCell::SetAlignHor).

- wxHtmlContainerCell * SetContainer (wxHtmlContainerCell *c)

  Allows you to directly set opened container.

- virtual void SetDC (wxDC *dc, double pixel_scale=1.0e+0)

  Sets the DC.

- void SetFontBold (int x)

  Sets bold flag of actualfont.

- void SetFontFace (const wxString &face)

  Sets current font face to face.

- void SetFontFixed (int x)

  Sets fixed face flag of actualfont.

- void SetFontItalic (int x)

  Sets italic flag of actualfont.

- void SetFontSize (int s)

  Sets actual font size (HTML size varies from 1 to 7).

- void SetFontUnderlined (int x)

  Sets underlined flag of actualfont.
- void SetFonts (const wxString &normal_face, const wxString &fixed_face, const int *sizes=0)

  Sets fonts.
- void SetInputEncoding (wxFontEncoding enc)

  Sets input encoding.
- void SetLink (const wxHtmlLinkInfo &link)

  Sets actual hypertext link.
- void SetLinkColor (const wxColour &clr)

  Sets colour of hypertext link.

Static Public Member Functions

- static void AddModule (wxHtmlTagsModule *module)

  Adds module() to the list of wxHtmlWinParser tag handler.

Additional Inherited Members

21.384.2 Constructor & Destructor Documentation

wxHtmlWinParser::wxHtmlWinParser (wxHtmlWindowInterface * *wndIface* = NULL)

Constructor.

Don't use the default one, use the constructor with *wndIface* parameter (*wndIface* is a pointer to interface object for the associated wxHtmlWindow or other HTML rendering window such as wxHtmlListBox).

21.384.3 Member Function Documentation

static void wxHtmlWinParser::AddModule (wxHtmlTagsModule * *module*) [static]

Adds module() to the list of wxHtmlWinParser tag handler.

wxHtmlContainerCell* wxHtmlWinParser::CloseContainer ( )

Closes the container, sets actual container to the parent one and returns pointer to it (see Cells and Containers).

virtual wxFont* wxHtmlWinParser::CreateCurrentFont ( ) [virtual]

Creates font based on current setting (see SetFontSize(), SetFontBold(), SetFontItalic(), SetFontFixed(), wxHtmlWinParser::SetFontUnderlined) and returns pointer to it.

If the font was already created only a pointer is returned.

const wxColour& wxHtmlWinParser::GetActualColor ( ) const

Returns actual text colour.

int wxHtmlWinParser::GetAlign ( ) const

Returns default horizontal alignment.

int wxHtmlWinParser::GetCharHeight ( ) const

Returns (average) char height in standard font.

It is used as DC-independent metrics.

Note

This function doesn't return the *actual* height. If you want to know the height of the current font, call `GetDC->GetCharHeight()`.

int wxHtmlWinParser::GetCharWidth ( ) const

Returns average char width in standard font.

It is used as DC-independent metrics.

Note

This function doesn't return the *actual* width. If you want to know the height of the current font, call `GetDC->GetCharWidth()`.

wxHtmlContainerCell* wxHtmlWinParser::GetContainer ( ) const

Returns pointer to the currently opened container (see Cells and Containers).

Common use:

```
m_WParser -> GetContainer() -> InsertCell(new ...);
```

wxDC* wxHtmlWinParser::GetDC ( )

Returns pointer to the DC used during parsing.

wxEncodingConverter* wxHtmlWinParser::GetEncodingConverter ( ) const

Returns wxEncodingConverter class used to do conversion between the input encoding and the output encoding.

int wxHtmlWinParser::GetFontBold ( ) const

Returns true if actual font is bold, false otherwise.

wxString wxHtmlWinParser::GetFontFace ( ) const

Returns actual font face name.

int wxHtmlWinParser::GetFontFixed ( ) const

Returns true if actual font is fixed face, false otherwise.

int wxHtmlWinParser::GetFontItalic ( ) const

Returns true if actual font is italic, false otherwise.

int wxHtmlWinParser::GetFontSize ( ) const

Returns actual font size (HTML size varies from -2 to +4)

int wxHtmlWinParser::GetFontUnderlined ( ) const

Returns true if actual font is underlined, false otherwise.

wxFontEncoding wxHtmlWinParser::GetInputEncoding ( ) const

Returns input encoding.

const wxHtmlLinkInfo& wxHtmlWinParser::GetLink ( ) const

Returns actual hypertext link.

(This value has a non-empty wxHtmlLinkInfo::GetHref Href string if the parser is between <A> and </A> tags, wxEmptyString otherwise.)

const wxColour& wxHtmlWinParser::GetLinkColor ( ) const

Returns the colour of hypertext link text.

wxFontEncoding wxHtmlWinParser::GetOutputEncoding ( ) const

Returns output encoding, i.e.

closest match to document's input encoding that is supported by operating system.

wxHtmlWindowInterface* wxHtmlWinParser::GetWindowInterface ( )

Returns associated window (wxHtmlWindow).

This may be NULL! (You should always test if it is non-NULL. For example `TITLE` handler sets window title only if some window is associated, otherwise it does nothing.

wxHtmlContainerCell* wxHtmlWinParser::OpenContainer ( )

Opens new container and returns pointer to it (see Cells and Containers).

void wxHtmlWinParser::SetActualColor (const wxColour & *clr*)

Sets actual text colour.

Note: this DOESN'T change the colour! You must create wxHtmlColourCell yourself.

void wxHtmlWinParser::SetAlign (int *a*)

Sets default horizontal alignment (see wxHtmlContainerCell::SetAlignHor).

Alignment of newly opened container is set to this value.

wxHtmlContainerCell* wxHtmlWinParser::SetContainer (wxHtmlContainerCell * *c*)

Allows you to directly set opened container.

This is not recommended - you should use OpenContainer() wherever possible.

virtual void wxHtmlWinParser::SetDC (wxDC * *dc*, double *pixel_scale* = `1.0e+0`) [virtual]

Sets the DC.

This must be called before wxHtmlParser::Parse!

pixel_scale can be used when rendering to high-resolution DCs (e.g. printer) to adjust size of pixel metrics. (Many dimensions in HTML are given in pixels – e.g. image sizes. 300x300 image would be only one inch wide on typical printer. With pixel_scale = 3.0 it would be 3 inches.)

void wxHtmlWinParser::SetFontBold (int *x*)

Sets bold flag of actualfont.

x is either true of false.

void wxHtmlWinParser::SetFontFace (const wxString & *face*)

Sets current font face to *face*.

This affects either fixed size font or proportional, depending on context (whether the parser is inside `<TT>` tag or not).

void wxHtmlWinParser::SetFontFixed (int *x*)

Sets fixed face flag of actualfont.

x is either true of false.

void wxHtmlWinParser::SetFontItalic (int *x*)

Sets italic flag of actualfont.

x is either true of false.

void wxHtmlWinParser::SetFonts (const wxString & *normal_face*, const wxString & *fixed_face*, const int * *sizes* = `0`)

Sets fonts.

See wxHtmlWindow::SetFonts for detailed description.

void wxHtmlWinParser::SetFontSize (int *s*)

Sets actual font size (HTML size varies from 1 to 7).

void wxHtmlWinParser::SetFontUnderlined (int *x*)

Sets underlined flag of actualfont.

x is either true of false.

void wxHtmlWinParser::SetInputEncoding (wxFontEncoding *enc*)

Sets input encoding.

The parser uses this information to build conversion tables from document's encoding to some encoding supported by operating system.

void wxHtmlWinParser::SetLink (const wxHtmlLinkInfo & *link*)

Sets actual hypertext link.

Empty link is represented by wxHtmlLinkInfo with *Href* equal to wxEmptyString.

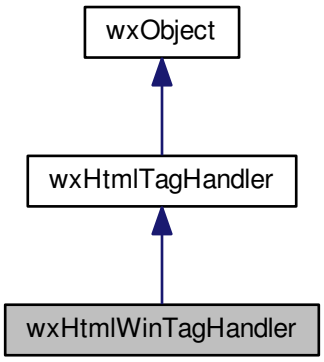
void wxHtmlWinParser::SetLinkColor (const wxColour & *clr*)

Sets colour of hypertext link.

21.385 wxHtmlWinTagHandler Class Reference

`#include <wx/html/winpars.h>`

Inheritance diagram for wxHtmlWinTagHandler:



21.385.1 Detailed Description

This is basically wxHtmlTagHandler except that it is extended with protected member m_WParser pointing to the wxHtmlWinParser object (value of this member is identical to wxHtmlParser's m_Parser).

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlWinTagHandler ()

  Constructor.

- virtual void SetParser (wxHtmlWinParser *parser)

  Assigns parser to this handler.

Protected Attributes

- wxHtmlWinParser * m_WParser

  Value of this attribute is identical to value of m_Parser.

Additional Inherited Members

21.385.2 Constructor & Destructor Documentation

wxHtmlWinTagHandler::wxHtmlWinTagHandler ( )

Constructor.

21.385.3 Member Function Documentation

virtual void wxHtmlWinTagHandler::SetParser (wxHtmlWinParser * *parser*) `[virtual]`

Assigns *parser* to this handler.

Each **instance** of handler is guaranteed to be called only from the one parser.

21.385.4 Member Data Documentation

wxHtmlWinParser* wxHtmlWinTagHandler::m_WParser `[protected]`

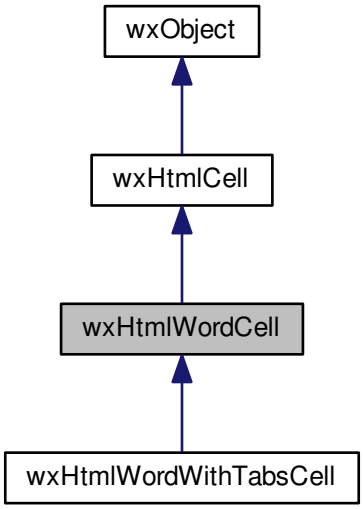
Value of this attribute is identical to value of m_Parser.

The only difference is that m_WParser points to wxHtmlWinParser object while m_Parser points to wxHtmlParser object. (The same object, but overcast.)

21.386 wxHtmlWordCell Class Reference

`#include <wx/html/htmlcell.h>`

Inheritance diagram for wxHtmlWordCell:



21.386.1 Detailed Description

This html cell represents a single word or text fragment in the document stream.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlWordCell (const wxString &word, const wxDC &dc)

Additional Inherited Members

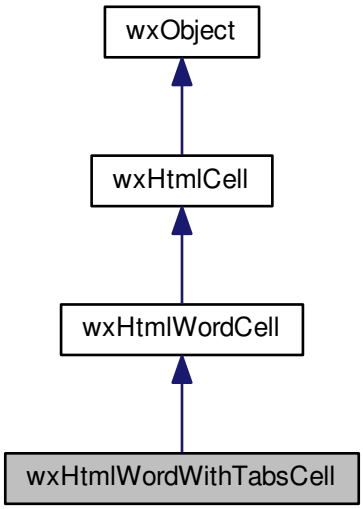
21.386.2 Constructor & Destructor Documentation

wxHtmlWordCell::wxHtmlWordCell (const wxString & *word,* const wxDC & *dc*)

21.387 wxHtmlWordWithTabsCell Class Reference

```
#include <wx/html/htmlcell.h>
```

Inheritance diagram for wxHtmlWordWithTabsCell:



21.387.1 Detailed Description

wxHtmlWordCell is a specialization for storing text fragments with embedded tab characters.

Library: wxHTML

Category: HTML

Public Member Functions

- wxHtmlWordWithTabsCell (const wxString &word, const wxString &wordOrig, size_t linepos, const wxDC &dc)

Additional Inherited Members

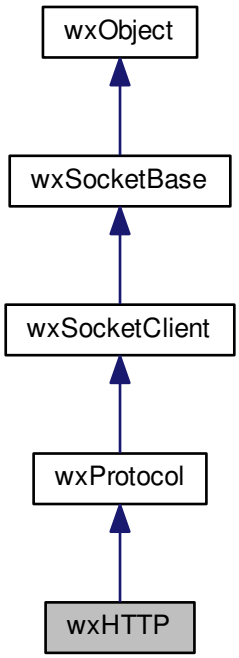
21.387.2 Constructor & Destructor Documentation

wxHtmlWordWithTabsCell::wxHtmlWordWithTabsCell (const wxString & *word,* const wxString & *wordOrig,* size_t *linepos,* const wxDC & *dc*)

21.388 wxHTTP Class Reference

```
#include <wx/protocol/http.h>
```

Inheritance diagram for wxHTTP:



21.388.1 Detailed Description

wxHTTP can be used to establish a connection to an HTTP server.

wxHTTP can thus be used to create a (basic) HTTP **client**.

Library: wxNet

Category: Networking

See also

wxSocketBase, wxURL

Public Member Functions

- wxHTTP ()

  Default constructor.
- virtual ~wxHTTP ()

  Destructor will close the connection if connected.
- wxString GetHeader (const wxString &header) const

  Returns the data attached with a field whose name is specified by header.
- virtual wxInputStream * GetInputStream (const wxString &path)

Creates a new input stream on the specified path.

- int GetResponse () const

  Returns the HTTP response code returned by the server.

- void SetMethod (const wxString &method)

  Set HTTP method.

- void SetHeader (const wxString &header, const wxString &h_data)

  It sets data of a field to be sent during the next request to the HTTP server.

- wxString GetCookie (const wxString &cookie) const

  Returns the value of a cookie.

- bool HasCookies () const

  Returns true if there were cookies.

- bool SetPostBuffer (const wxString &contentType, const wxMemoryBuffer &data)

  Set the binary data to be posted to the server.

- bool SetPostText (const wxString &contentType, const wxString &data, const wxMBConv &conv=wxConvU↩ TF8)

  Set the text to be posted to the server.

- virtual bool Connect (const wxString &host)

  Connect to the HTTP server.

- virtual bool Connect (const wxString &host, unsigned short port)

  Connect to the HTTP server.

- virtual bool Connect (const wxSockAddress &addr, bool wait)

  Connect to the HTTP server.

Additional Inherited Members

21.388.2 Constructor & Destructor Documentation

wxHTTP::wxHTTP ( )

Default constructor.

virtual wxHTTP::~wxHTTP ( ) [virtual]

Destructor will close the connection if connected.

21.388.3 Member Function Documentation

virtual bool wxHTTP::Connect (const wxString & *host*) [virtual]

Connect to the HTTP server.

By default, connection is made to the port 80 of the specified *host*. You may connect to a non-default port by specifying it explicitly using the second overload.

Currently wxHTTP only supports IPv4.

For the overload taking wxSockAddress, the *wait* argument is ignored.

virtual bool wxHTTP::Connect (const wxString & *host,* unsigned short *port*) `[virtual]`

Connect to the HTTP server.

By default, connection is made to the port 80 of the specified *host*. You may connect to a non-default port by specifying it explicitly using the second overload.

Currently wxHTTP only supports IPv4.

For the overload taking wxSockAddress, the *wait* argument is ignored.

virtual bool wxHTTP::Connect (const wxSockAddress & *addr,* bool *wait*) `[virtual]`

Connect to the HTTP server.

By default, connection is made to the port 80 of the specified *host*. You may connect to a non-default port by specifying it explicitly using the second overload.

Currently wxHTTP only supports IPv4.

For the overload taking wxSockAddress, the *wait* argument is ignored.

Reimplemented from wxSocketClient.

wxString wxHTTP::GetCookie (const wxString & *cookie*) const

Returns the value of a cookie.

wxString wxHTTP::GetHeader (const wxString & *header*) const

Returns the data attached with a field whose name is specified by *header*.

If the field doesn't exist, it will return an empty string and not a NULL string.

Note

The header is not case-sensitive, i.e. "CONTENT-TYPE" and "content-type" represent the same header.

virtual wxInputStream* wxHTTP::GetInputStream (const wxString & *path*) `[virtual]`

Creates a new input stream on the specified path.

Notice that this stream is unseekable, i.e. SeekI() and TellI() methods shouldn't be used.

Note that you can still know the size of the file you are getting using wxStreamBase::GetSize(). However there is a limitation: in HTTP protocol, the size is not always specified so sometimes (size_t)-1 can returned to indicate that the size is unknown. In such case, you may want to use wxInputStream::LastRead() method in a loop to get the total size.

Returns

Returns the initialized stream. You must delete it yourself once you don't use it anymore and this must be done before the wxHTTP object itself is destroyed. The destructor closes the network connection. The next time you will try to get a file the network connection will have to be reestablished, but you don't have to take care of this since wxHTTP reestablishes it automatically.

See also

wxInputStream

Implements wxProtocol.

int wxHTTP::GetResponse ( ) const

Returns the HTTP response code returned by the server.

Please refer to RFC 2616 for the list of responses.

bool wxHTTP::HasCookies ( ) const

Returns true if there were cookies.

void wxHTTP::SetHeader (const wxString & *header*, const wxString & *h_data*)

It sets data of a field to be sent during the next request to the HTTP server.

The field name is specified by *header* and the content by *h_data*. This is a low level function and it assumes that you know what you are doing.

void wxHTTP::SetMethod (const wxString & *method*)

Set HTTP method.

Set `common` or expanded HTTP method.

Overrides GET or POST methods that is used by default.

Parameters

method	HTTP method name, e.g. "GET".

Since

3.0

See also

SetPostBuffer(), SetPostText()

bool wxHTTP::SetPostBuffer (const wxString & *contentType*, const wxMemoryBuffer & *data*)

Set the binary data to be posted to the server.

If a non-empty buffer is passed to this method, the next request will be an HTTP `POST` instead of the default HTTP `GET` and the given *data* will be posted as the body of this request.

For textual data a more convenient SetPostText() can be used instead.

Parameters

contentType	The value of HTTP "Content-Type" header, e.g. "image/png".
data	The data to post.

Returns

true if any data was passed in or false if the buffer was empty.

Since

2.9.4

bool wxHTTP::SetPostText (const wxString & *contentType,* const wxString & *data,* const wxMBConv & *conv* = wxConvUTF8)

Set the text to be posted to the server.

After a successful call to this method, the request will use HTTP `POST` instead of the default `GET` when it's executed.

Use SetPostBuffer() if you need to post non-textual data.

Parameters

contentType	The value of HTTP "Content-Type" header, e.g. "text/html; charset=UTF-8".
data	The data to post.
conv	The conversion to use to convert *data* contents to a byte stream. Its value should be consistent with the charset parameter specified in *contentType*.

Returns

true if string was non-empty and was successfully converted using the given *conv* or false otherwise (in this case this request won't be `POST'ed` correctly).

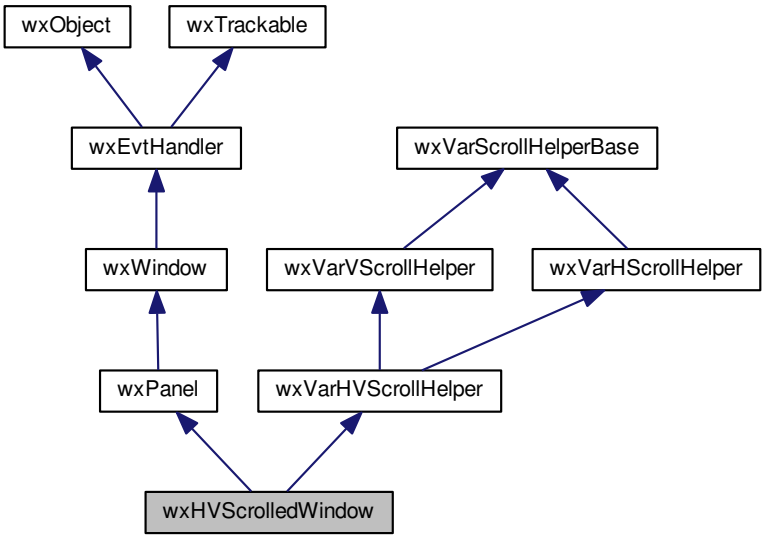
Since

2.9.4

21.389 wxHVScrolledWindow Class Reference

`#include <wx/vscroll.h>`

Inheritance diagram for wxHVScrolledWindow:



21.389.1 Detailed Description

This window inherits all functionality of both vertical and horizontal, variable scrolled windows.

It automatically handles everything needed to scroll both axis simultaneously with both variable row heights and variable column widths.

In any case, this is a generalization of wxScrolled which can be only used when all rows and columns are the same size. It lacks some other wxScrolled features however, notably it can't scroll specific pixel sizes of the window or its exact client area size.

To use this class, you must derive from it and implement both the OnGetRowHeight() and OnGetColumnWidth() pure virtual methods to let the base class know how many rows and columns it should display. You also need to set the total rows and columns the window contains, but from that moment on the scrolling is handled entirely by wxHVScrolledWindow. You only need to draw the visible part of contents in your `OnPaint()` method as usual. You should use GetVisibleBegin() and GetVisibleEnd() to select the lines to display. Note that the device context origin is not shifted so the first visible row and column always appear at the point (0, 0) in physical as well as logical coordinates.

Library: wxCore

Category: Miscellaneous Windows

See also

wxHScrolledWindow, wxVScrolledWindow

Public Member Functions

- wxHVScrolledWindow ()

  Default constructor, you must call Create() later.
- wxHVScrolledWindow (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  This is the normal constructor, no need to call Create() after using this constructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  Same as the non-default constructor, but returns a status code: true if ok, false if the window couldn't be created.

Additional Inherited Members

21.389.2 Constructor & Destructor Documentation

wxHVScrolledWindow::wxHVScrolledWindow ( )

Default constructor, you must call Create() later.

wxHVScrolledWindow::wxHVScrolledWindow (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxPanelNameStr`)

This is the normal constructor, no need to call Create() after using this constructor.

Note

`wxHSCROLL` and `wxVSCROLL` are always automatically added to the style, there is no need to specify it explicitly.

Parameters

parent	The parent window, must not be NULL.
id	The identifier of this window, wxID_ANY by default.
pos	The initial window position.
size	The initial window size.
style	The window style. There are no special style bits defined for this class.
name	The name for this window; usually not used.

21.389.3 Member Function Documentation

bool wxHVScrolledWindow::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxPanelNameStr`)

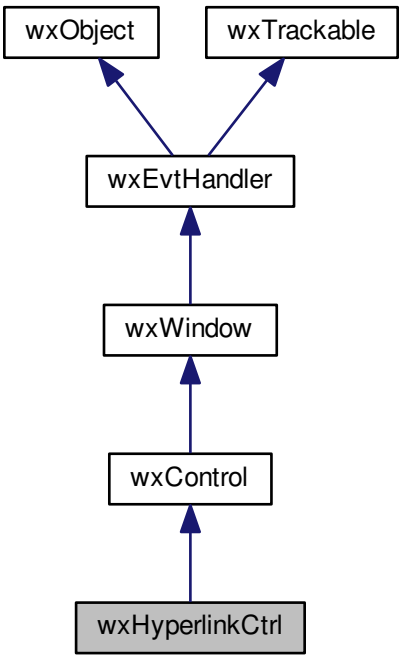
Same as the non-default constructor, but returns a status code: true if ok, false if the window couldn't be created.

Just as with the constructor, the `wxHSCROLL` and `wxVSCROLL` styles are always used, there is no need to specify them explicitly.

21.390 wxHyperlinkCtrl Class Reference

`#include <wx/hyperlink.h>`

Inheritance diagram for wxHyperlinkCtrl:



21.390.1 Detailed Description

This class shows a static text element which links to an URL.

Appearance and behaviour is completely customizable.

In fact, when the user clicks on the hyperlink, a wxHyperlinkEvent is sent but if that event is not handled (or it's skipped; see wxEvent::Skip), then a call to wxLaunchDefaultBrowser() is done with the hyperlink's URL.

Note that standard wxWindow functions like wxWindow::SetBackgroundColour, wxWindow::SetFont, wxWindow::SetCursor, wxWindow::SetLabel can be used to customize appearance of the hyperlink.

Styles

This class supports the following styles:

- wxHL_ALIGN_LEFT: Align the text to the left.
- wxHL_ALIGN_RIGHT: Align the text to the right. This style is not supported under Windows XP but is supported under all the other Windows versions.
- wxHL_ALIGN_CENTRE: Center the text (horizontally). This style is not supported by the native MSW implementation used under Windows XP and later.
- wxHL_CONTEXTMENU: Pop up a context menu when the hyperlink is right-clicked. The context menu contains a "Copy URL" menu item which is automatically handled by the hyperlink and which just copies in the clipboard the URL (not the label) of the control.
- wxHL_DEFAULT_STYLE: The default style for wxHyperlinkCtrl: wxBORDER_NONE|wxHL_CONTEXTMENU|wxHL_ALIGN_CENTRE.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHyperlinkEvent& event)

Event macros for events emitted by this class:

- EVT_HYPERLINK(id, func): The hyperlink was (left) clicked. If this event is not handled in user's code (or it's skipped; see wxEvent::Skip), then a call to wxLaunchDefaultBrowser is done with the hyperlink's URL.

Currently this class is implemented using native support in wxGTK and wxMSW (under Windows XP and later only) and a generic version is used by the other ports.

Library: wxAdvanced

Category: Controls

See also

wxURL, wxHyperlinkEvent

Public Member Functions

- wxHyperlinkCtrl ()
- wxHyperlinkCtrl (wxWindow *parent, wxWindowID id, const wxString &label, const wxString &url, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHL_DEFAULT_STYLE, const wxString &name=wxHyperlinkCtrlNameStr)

Constructor.

- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxString &url, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHL_DEFAULT_STYLE, const wxString &name=wxHyperlinkCtrlNameStr)

  Creates the hyperlink control.
- virtual wxColour GetHoverColour () const

  Returns the colour used to print the label of the hyperlink when the mouse is over the control.
- virtual wxColour GetNormalColour () const

  Returns the colour used to print the label when the link has never been clicked before (i.e. the link has not been visited) and the mouse is not over the control.
- virtual wxString GetURL () const

  Returns the URL associated with the hyperlink.
- virtual bool GetVisited () const =0

  Returns true if the hyperlink has already been clicked by the user at least one time.
- virtual wxColour GetVisitedColour () const

  Returns the colour used to print the label when the mouse is not over the control and the link has already been clicked before (i.e. the link has been visited).
- virtual void SetHoverColour (const wxColour &colour)

  Sets the colour used to print the label of the hyperlink when the mouse is over the control.
- virtual void SetNormalColour (const wxColour &colour)

  Sets the colour used to print the label when the link has never been clicked before (i.e. the link has not been visited) and the mouse is not over the control.
- virtual void SetURL (const wxString &url)

  Sets the URL associated with the hyperlink.
- virtual void SetVisited (bool visited=true)=0

  Marks the hyperlink as visited (see wxHyperlinkCtrl::SetVisitedColour).
- virtual void SetVisitedColour (const wxColour &colour)

  Sets the colour used to print the label when the mouse is not over the control and the link has already been clicked before (i.e. the link has been visited).

Additional Inherited Members

21.390.2 Constructor & Destructor Documentation

wxHyperlinkCtrl::wxHyperlinkCtrl ( )

wxHyperlinkCtrl::wxHyperlinkCtrl (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxString & *url*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHL_DEFAULT_STYLE, const wxString & *name* = `wxHyperlinkCtrlNameStr`)

Constructor.

See Create() for more info.

21.390.3 Member Function Documentation

bool wxHyperlinkCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxString & *url*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHL_DEFAULT_STYLE, const wxString & *name* = `wxHyperlinkCtrlNameStr`)

Creates the hyperlink control.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. A value of wxID_ANY indicates a default value.
label	The label of the hyperlink.
url	The URL associated with the given label.
pos	Window position.
size	Window size. If the wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxHyperlinkCtrl.
name	Window name.

virtual wxColour wxHyperlinkCtrl::GetHoverColour ( ) const `[virtual]`

Returns the colour used to print the label of the hyperlink when the mouse is over the control.

virtual wxColour wxHyperlinkCtrl::GetNormalColour ( ) const `[virtual]`

Returns the colour used to print the label when the link has never been clicked before (i.e. the link has not been *visited*) and the mouse is not over the control.

virtual wxString wxHyperlinkCtrl::GetURL ( ) const `[virtual]`

Returns the URL associated with the hyperlink.

virtual bool wxHyperlinkCtrl::GetVisited ( ) const `[pure virtual]`

Returns true if the hyperlink has already been clicked by the user at least one time.

virtual wxColour wxHyperlinkCtrl::GetVisitedColour ( ) const `[virtual]`

Returns the colour used to print the label when the mouse is not over the control and the link has already been clicked before (i.e. the link has been *visited*).

virtual void wxHyperlinkCtrl::SetHoverColour (const wxColour & *colour*) `[virtual]`

Sets the colour used to print the label of the hyperlink when the mouse is over the control.

virtual void wxHyperlinkCtrl::SetNormalColour (const wxColour & *colour*) `[virtual]`

Sets the colour used to print the label when the link has never been clicked before (i.e. the link has not been *visited*) and the mouse is not over the control.

virtual void wxHyperlinkCtrl::SetURL (const wxString & *url*) `[virtual]`

Sets the URL associated with the hyperlink.

virtual void wxHyperlinkCtrl::SetVisited (bool *visited* = `true`) `[pure virtual]`

Marks the hyperlink as visited (see wxHyperlinkCtrl::SetVisitedColour).

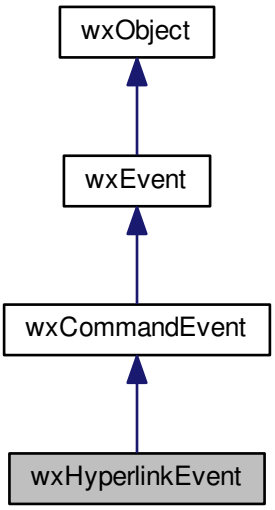
virtual void wxHyperlinkCtrl::SetVisitedColour (const wxColour & *colour*) `[virtual]`

Sets the colour used to print the label when the mouse is not over the control and the link has already been clicked before (i.e. the link has been *visited*).

21.391 wxHyperlinkEvent Class Reference

`#include <wx/hyperlink.h>`

Inheritance diagram for wxHyperlinkEvent:



21.391.1 Detailed Description

This event class is used for the events generated by wxHyperlinkCtrl.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxHyperlinkEvent& event)

Event macros:

- EVT_HYPERLINK(id, func): User clicked on an hyperlink.

Library: wxAdvanced

Category: Events

Public Member Functions

- wxHyperlinkEvent (wxObject *generator, int id, const wxString &url)

  The constructor is not normally used by the user code.
- wxString GetURL () const

  Returns the URL of the hyperlink where the user has just clicked.
- void SetURL (const wxString &url)

  Sets the URL associated with the event.

Additional Inherited Members

21.391.2 Constructor & Destructor Documentation

wxHyperlinkEvent::wxHyperlinkEvent (wxObject * *generator,* int *id,* const wxString & *url*)

The constructor is not normally used by the user code.

21.391.3 Member Function Documentation

wxString wxHyperlinkEvent::GetURL ( ) const

Returns the URL of the hyperlink where the user has just clicked.

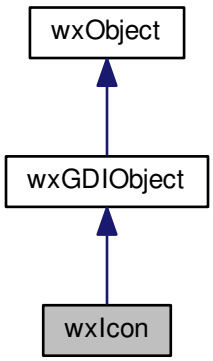
void wxHyperlinkEvent::SetURL (const wxString & *url*)

Sets the URL associated with the event.

21.392 wxIcon Class Reference

`#include <wx/icon.h>`

Inheritance diagram for wxIcon:



21.392.1 Detailed Description

An icon is a small rectangular bitmap usually used for denoting a minimized application.

It differs from a wxBitmap in always having a mask associated with it for transparent drawing. On some platforms, icons and bitmaps are implemented identically, since there is no real distinction between a wxBitmap with a mask and an icon; and there is no specific icon format on some platforms (X-based applications usually standardize on XPMs for small bitmaps and icons). However, some platforms (such as Windows) make the distinction, so a separate class is provided.

Remarks

It is usually desirable to associate a pertinent icon with a frame. Icons can also be used for other purposes, for example with wxTreeCtrl and wxListCtrl. Icons have different formats on different platforms therefore separate icons will usually be created for the different environments. Platform-specific methods for creating a wxIcon structure are catered for, and this is an occasion where conditional compilation will probably be required. Note that a new icon must be created for every time the icon is to be used for a new window. In Windows, the icon will not be reloaded if it has already been used. An icon allocated to a frame will be deleted when the frame is deleted. For more information please see Bitmaps and Icons.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullIcon

See also

Bitmaps and Icons, Supported Bitmap File Formats, wxDC::DrawIcon, wxCursor

Public Member Functions

- wxIcon ()

  Default ctor.
- wxIcon (const wxIcon &icon)

  Copy ctor.
- wxIcon (const char bits[], int width, int height)

  Creates an icon from an array of bits.
- wxIcon (const char ∗const ∗bits)

  Creates a bitmap from XPM data.
- wxIcon (const wxString &name, wxBitmapType type=wxICON_DEFAULT_TYPE, int desiredWidth=-1, int desiredHeight=-1)

  Loads an icon from a file or resource.
- wxIcon (const wxIconLocation &loc)

  Loads an icon from the specified location.
- virtual ∼wxIcon ()

  Destructor.
- bool CreateFromHICON (WXHICON icon)

  Attach a Windows icon handle.
- wxIcon ConvertToDisabled (unsigned char brightness=255) const

  Returns disabled (dimmed) version of the icon.
- void CopyFromBitmap (const wxBitmap &bmp)

  Copies bmp bitmap to this icon.
- int GetDepth () const

Gets the colour depth of the icon.

- int GetHeight () const

  Gets the height of the icon in pixels.

- int GetWidth () const

  Gets the width of the icon in pixels.

- virtual bool IsOk () const

  Returns true if icon data is present.

- bool LoadFile (const wxString &name, wxBitmapType type=wxICON_DEFAULT_TYPE, int desiredWidth=-1, int desiredHeight=-1)

  Loads an icon from a file or resource.

- void SetDepth (int depth)

  Sets the depth member (does not affect the icon data).

- void SetHeight (int height)

  Sets the height member (does not affect the icon data).

- void SetWidth (int width)

  Sets the width member (does not affect the icon data).

- wxIcon & operator= (const wxIcon &icon)

  Assignment operator, using Reference Counting.

Additional Inherited Members

21.392.2 Constructor & Destructor Documentation

wxIcon::wxIcon ( )

Default ctor.

Constructs an icon object with no data; an assignment or another member function such as LoadFile() must be called subsequently.

wxIcon::wxIcon (const wxIcon & *icon*)

Copy ctor.

wxIcon::wxIcon (const char *bits[],* int *width,* int *height*)

Creates an icon from an array of bits.

You should only use this function for monochrome bitmaps (depth 1) in portable programs: in this case the bits parameter should contain an XBM image.

For other bit depths, the behaviour is platform dependent: under Windows, the data is passed without any changes to the underlying CreateBitmap() API. Under other platforms, only monochrome bitmaps may be created using this constructor and wxImage should be used for creating colour bitmaps from static data.

Parameters

bits	Specifies an array of pixel values.
width	The width of the image.
height	The height of the image.

wxPerl Note: In wxPerl use Wx::Icon->newBits(bits, width, height, depth = -1);

Availability: only available for the wxMSW, wxOSX ports.

wxIcon::wxIcon (const char *const * *bits*)

Creates a bitmap from XPM data.

To use this constructor, you must first include an XPM file. For example, assuming that the file mybitmap.xpm contains an XPM array of character pointers called *mybitmap:*

```
#include "mybitmap.xpm"
...
wxIcon *icon = new wxIcon(mybitmap);
```

A macro, wxICON, is available which creates an icon using an XPM on the appropriate platform, or an icon resource on Windows.

```
wxIcon icon(wxICON(sample));

// Equivalent to:
#if defined(__WXGTK__) || defined(__WXMOTIF__)
wxIcon icon(sample_xpm);
#endif

#if defined(__WXMSW__)
wxIcon icon("sample");
#endif
```

wxPerl Note: In wxPerl use Wx::Icon->newFromXPM(data).

wxIcon::wxIcon (const wxString & *name*, wxBitmapType *type* = wxICON_DEFAULT_TYPE, int *desiredWidth* = -1, int *desiredHeight* = -1)

Loads an icon from a file or resource.

Parameters

name	This can refer to a resource name or a filename under MS Windows and X. Its meaning is determined by the *type* parameter.
type	May be one of the wxBitmapType values and indicates which type of bitmap should be loaded. See the note in the class detailed description. Note that the wxICON_DEFAULT_TYP↩ E constant has different value under different wxWidgets ports. See the icon.h header for the value it takes for a specific port.
desiredWidth	Specifies the desired width of the icon. This parameter only has an effect in Windows where icon resources can contain several icons of different sizes.
desiredHeight	Specifies the desired height of the icon. This parameter only has an effect in Windows where icon resources can contain several icons of different sizes.

See also

LoadFile()

wxIcon::wxIcon (const wxIconLocation & *loc*)

Loads an icon from the specified location.

virtual wxIcon::~wxIcon ( ) [virtual]

Destructor.

See Object Destruction for more info.

If the application omits to delete the icon explicitly, the icon will be destroyed automatically by wxWidgets when the application exits.

Warning

Do not delete an icon that is selected into a memory device context.

21.392.3 Member Function Documentation

wxIcon wxIcon::ConvertToDisabled (unsigned char *brightness* = 255) const

Returns disabled (dimmed) version of the icon.

This method is available in wxIcon only under wxMSW, other ports only have it in wxBitmap. You can always use wxImage::ConvertToDisabled() and create the icon from wxImage manually however.

Availability: only available for the wxMSW port.

Since

2.9.0

void wxIcon::CopyFromBitmap (const wxBitmap & *bmp*)

Copies *bmp* bitmap to this icon.

Under MS Windows the bitmap must have mask colour set.

See also

LoadFile()

bool wxIcon::CreateFromHICON (WXHICON *icon*)

Attach a Windows icon handle.

This wxMSW-specific method allows to assign a native Windows HICON (which must be castes to WXHICON opaque handle type) to wxIcon. Notice that this means that the HICON will be destroyed by wxIcon when it is destroyed.

Returns

true if successful.

Availability: only available for the wxMSW port.

Since

2.9.5

int wxIcon::GetDepth ( ) const

Gets the colour depth of the icon.

A value of 1 indicates a monochrome icon.

int wxIcon::GetHeight ( ) const

Gets the height of the icon in pixels.

See also

GetWidth()

int wxIcon::GetWidth ( ) const

Gets the width of the icon in pixels.

See also

GetHeight()

virtual bool wxIcon::IsOk ( ) const `[virtual]`

Returns true if icon data is present.

bool wxIcon::LoadFile (const wxString & *name*, wxBitmapType *type* = `wxICON_DEFAULT_TYPE`, int *desiredWidth* = `-1`, int *desiredHeight* = `-1`)

Loads an icon from a file or resource.

Parameters

name	Either a filename or a Windows resource name. The meaning of name is determined by the *type* parameter.
type	One of the wxBitmapType values; see the note in the class detailed description. Note that the wxICON_DEFAULT_TYPE constant has different value under different wxWidgets ports. See the icon.h header for the value it takes for a specific port.
desiredWidth	Specifies the desired width of the icon. This parameter only has an effect in Windows where icon resources can contain several icons of different sizes.
desiredHeight	Specifies the desired height of the icon. This parameter only has an effect in Windows where icon resources can contain several icons of different sizes.

Returns

true if the operation succeeded, false otherwise.

wxIcon& wxIcon::operator= (const wxIcon & *icon*)

Assignment operator, using Reference Counting.

Parameters

icon	Icon to assign.

void wxIcon::SetDepth (int *depth*)

Sets the depth member (does not affect the icon data).

Parameters

depth	Icon depth.

void wxIcon::SetHeight (int *height*)

Sets the height member (does not affect the icon data).

Parameters

height	Icon height in pixels.

void wxIcon::SetWidth (int *width*)

Sets the width member (does not affect the icon data).

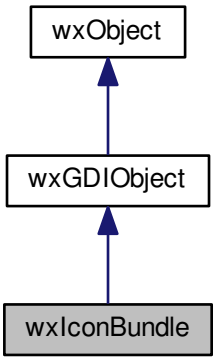
Parameters

width	Icon width in pixels.

21.393 wxIconBundle Class Reference

`#include <wx/iconbndl.h>`

Inheritance diagram for wxIconBundle:



21.393.1 Detailed Description

This class contains multiple copies of an icon in different sizes.

It is typically used in wxDialog::SetIcons and wxTopLevelWindow::SetIcons.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullIconBundle

Public Types

- enum {
  FALLBACK_NONE = 0,
  FALLBACK_SYSTEM = 1,
  FALLBACK_NEAREST_LARGER = 2 }

  The elements of this enum determine what happens if GetIcon() doesn't find the icon of exactly the requested size.

Public Member Functions

- wxIconBundle ()

  Default ctor.
- wxIconBundle (const wxString &file, wxBitmapType type=wxBITMAP_TYPE_ANY)

  Initializes the bundle with the icon(s) found in the file.
- wxIconBundle (wxInputStream &stream, wxBitmapType type=wxBITMAP_TYPE_ANY)

  Initializes the bundle with the icon(s) found in the stream.
- wxIconBundle (const wxIcon &icon)

  Initializes the bundle with a single icon.
- wxIconBundle (const wxIconBundle &ic)

  Copy constructor.
- virtual ~wxIconBundle ()

  Destructor.
- void AddIcon (const wxString &file, wxBitmapType type=wxBITMAP_TYPE_ANY)

  Adds all the icons contained in the file to the bundle; if the collection already contains icons with the same width and height, they are replaced by the new ones.
- void AddIcon (wxInputStream &stream, wxBitmapType type=wxBITMAP_TYPE_ANY)

  Adds all the icons contained in the stream to the bundle; if the collection already contains icons with the same width and height, they are replaced by the new ones.
- void AddIcon (const wxIcon &icon)

  Adds the icon to the collection; if the collection already contains an icon with the same width and height, it is replaced by the new one.
- wxIcon GetIcon (const wxSize &size, int flags=FALLBACK_SYSTEM) const

  Returns the icon with the given size.
- wxIcon GetIcon (wxCoord size=wxDefaultCoord, int flags=FALLBACK_SYSTEM) const

  Same as.
- wxIcon GetIconOfExactSize (const wxSize &size) const

  Returns the icon with exactly the given size or wxNullIcon if this size is not available.
- size_t GetIconCount () const

  return the number of available icons
- wxIcon GetIconByIndex (size_t n) const

  return the icon at index (must be < GetIconCount())
- bool IsEmpty () const

  Returns true if the bundle doesn't contain any icons, false otherwise (in which case a call to GetIcon() with default parameter should return a valid icon).
- wxIconBundle & operator= (const wxIconBundle &ic)

  Assignment operator, using reference counting.

Additional Inherited Members

21.393.2 Member Enumeration Documentation

anonymous enum

The elements of this enum determine what happens if GetIcon() doesn't find the icon of exactly the requested size.

Since

2.9.4

Enumerator

FALLBACK_NONE Return invalid icon if exact size is not found.

FALLBACK_SYSTEM Return the icon of the system icon size if exact size is not found. May be combined with other non-NONE enum elements to determine what happens if the system icon size is not found neither.

FALLBACK_NEAREST_LARGER Return the icon of closest larger size or, if there is no icon of larger size in the bundle, the closest icon of smaller size.

21.393.3 Constructor & Destructor Documentation

wxIconBundle::wxIconBundle ( )

Default ctor.

wxIconBundle::wxIconBundle (const wxString & *file,* wxBitmapType *type* = wxBITMAP_TYPE_ANY)

Initializes the bundle with the icon(s) found in the file.

wxIconBundle::wxIconBundle (wxInputStream & *stream,* wxBitmapType *type* = wxBITMAP_TYPE_ANY)

Initializes the bundle with the icon(s) found in the stream.

Notice that the *stream* must be seekable, at least if it contains more than one icon. The stream pointer is positioned after the last icon read from the stream when this function returns.

Since

2.9.0

wxIconBundle::wxIconBundle (const wxIcon & *icon*)

Initializes the bundle with a single icon.

wxIconBundle::wxIconBundle (const wxIconBundle & *ic*)

Copy constructor.

virtual wxIconBundle::~wxIconBundle ( ) [virtual]

Destructor.

21.393.4 Member Function Documentation

void wxIconBundle::AddIcon (const wxString & *file,* wxBitmapType *type* = wxBITMAP_TYPE_ANY)

Adds all the icons contained in the file to the bundle; if the collection already contains icons with the same width and height, they are replaced by the new ones.

void wxIconBundle::AddIcon (wxInputStream & *stream,* wxBitmapType *type* = wxBITMAP_TYPE_ANY)

Adds all the icons contained in the stream to the bundle; if the collection already contains icons with the same width and height, they are replaced by the new ones.

Notice that, as well as in the constructor loading the icon bundle from stream, the *stream* must be seekable, at least if more than one icon is to be loaded from it.

Since

2.9.0

void wxIconBundle::AddIcon (const wxIcon & *icon*)

Adds the icon to the collection; if the collection already contains an icon with the same width and height, it is replaced by the new one.

wxIcon wxIconBundle::GetIcon (const wxSize & *size,* int *flags* = FALLBACK_SYSTEM) const

Returns the icon with the given size.

If *size* is wxDefaultSize, it is interpreted as the standard system icon size, i.e. the size returned by wxSystemSettings::GetMetric() for `wxSYS_ICON_X` and `wxSYS_ICON_Y`.

If the bundle contains an icon with exactly the requested size, it's always returned. Otherwise, the behaviour depends on the flags. If only wxIconBundle::FALLBACK_NONE is given, the function returns an invalid icon. If wxIconBundle::FALLBACK_SYSTEM is given, it tries to find the icon of standard system size, regardless of the size passed as parameter. Otherwise, or if the icon system size is not found neither, but wxIconBundle::FALLBACK_NEAREST_LARGER flag is specified, the function returns the smallest icon of the size larger than the requested one or, if this fails too, just the icon closest to the specified size.

The *flags* parameter is available only since wxWidgets 2.9.4.

wxIcon wxIconBundle::GetIcon (wxCoord *size* = wxDefaultCoord, int *flags* = FALLBACK_SYSTEM) const

Same as.

```
GetIcon( wxSize( size, size ) )
```

.

wxIcon wxIconBundle::GetIconByIndex (size_t *n*) const

return the icon at index (must be $<$ GetIconCount())

size_t wxIconBundle::GetIconCount ( ) const

return the number of available icons

wxIcon wxIconBundle::GetIconOfExactSize (const wxSize & *size*) const

Returns the icon with exactly the given size or wxNullIcon if this size is not available.

bool wxIconBundle::IsEmpty ( ) const

Returns true if the bundle doesn't contain any icons, false otherwise (in which case a call to GetIcon() with default parameter should return a valid icon).

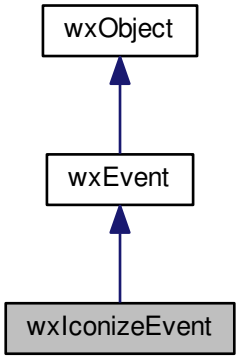
wxIconBundle& wxIconBundle::operator= (const wxIconBundle & *ic*)

Assignment operator, using reference counting.

21.394 wxIconizeEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxIconizeEvent:



21.394.1 Detailed Description

An event being sent when the frame is iconized (minimized) or restored.

Currently only wxMSW and wxGTK generate such events.

Availability: only available for the wxMSW, wxGTK ports.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxIconizeEvent& event)

Event macros:

- EVT_ICONIZE(func): Process a `wxEVT_ICONIZE` event.

Library: wxCore

Category: Events

See also

Events and Event Handling, wxTopLevelWindow::Iconize, wxTopLevelWindow::IsIconized

Public Member Functions

- wxIconizeEvent (int id=0, bool iconized=true)

  Constructor.
- bool IsIconized () const

  Returns true if the frame has been iconized, false if it has been restored.
- bool Iconized () const

Additional Inherited Members

21.394.2 Constructor & Destructor Documentation

wxIconizeEvent::wxIconizeEvent (int *id* = `0`, bool *iconized* = `true`)

Constructor.

21.394.3 Member Function Documentation

bool wxIconizeEvent::Iconized ( ) const

Deprecated This function is deprecated in favour of IsIconized().

bool wxIconizeEvent::IsIconized ( ) const

Returns true if the frame has been iconized, false if it has been restored.

21.395 wxIconLocation Class Reference

```
#include <wx/iconloc.h>
```

21.395.1 Detailed Description

wxIconLocation is a tiny class describing the location of an (external, i.e.

not embedded into the application resources) icon. For most platforms it simply contains the file name but under some others (notably Windows) the same file may contain multiple icons and so this class also stores the index of the icon inside the file.

In any case, its details should be of no interest to the application code and most of them are not even documented here (on purpose) as it is only meant to be used as an opaque class: the application may get the object of this class from somewhere and the only reasonable thing to do with it later is to create a wxIcon from it.

Library: wxBase

Category: Graphics Device Interface (GDI)

See also

wxIcon, wxFileType::GetIcon

Public Member Functions

- bool IsOk () const

  Returns true if the object is valid, i.e. was properly initialized, and false otherwise.
- void SetFileName (const wxString &filename)
- const wxString & GetFileName () const

21.395.2 Member Function Documentation

const wxString& wxIconLocation::GetFileName ( ) const

bool wxIconLocation::IsOk ( ) const

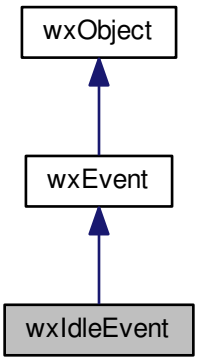
Returns true if the object is valid, i.e. was properly initialized, and false otherwise.

void wxIconLocation::SetFileName (const wxString & *filename*)

21.396 wxIdleEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxIdleEvent:



21.396.1 Detailed Description

This class is used for idle events, which are generated when the system becomes idle.

Note that, unless you do something specifically, the idle events are not sent if the system remains idle once it has become it, e.g. only a single idle event will be generated until something else resulting in more normal events happens and only then is the next idle event sent again.

If you need to ensure a continuous stream of idle events, you can either use wxIdleEvent::RequestMore method in your handler or call wxWakeUpIdle() periodically (for example from a timer event handler), but note that both of these approaches (and especially the first one) increase the system load and so should be avoided if possible.

By default, idle events are sent to all windows, including even the hidden ones because they may be shown if some condition is met from their `wxEVT_IDLE` (or related `wxEVT_UPDATE_UI`) handler. The children of hidden windows do not receive idle events however as they can't change their state in any way noticeable by the user. Finally, the global wxApp object also receives these events, as usual, so it can be used for any global idle time processing.

If sending idle events to all windows is causing a significant overhead in your application, you can call wxIdleEvent::SetMode with the value wxIDLE_PROCESS_SPECIFIED, and set the wxWS_EX_PROCESS_IDLE extra window style for every window which should receive idle events, all the other ones will not receive them in this case.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxIdleEvent& event)

Event macros:

- EVT_IDLE(func): Process a `wxEVT_IDLE` event.

Library: wxBase

Category: Events

21.396.2 Delayed Action Mechanism

wxIdleEvent can be used to perform some action "at slightly later time". This can be necessary in several circumstances when, for whatever reason, something can't be done in the current event handler. For example, if a mouse event handler is called with the mouse button pressed, the mouse can be currently captured and some operations with it – notably capturing it again – might be impossible or lead to undesirable results. If you still want to capture it, you can do it from `wxEVT_IDLE` handler when it is called the next time instead of doing it immediately.

This can be achieved in two different ways: when using static event tables, you will need a flag indicating to the (always connected) idle event handler whether the desired action should be performed. The originally called handler would then set it to indicate that it should indeed be done and the idle handler itself would reset it to prevent it from doing the same action again.

Using dynamically connected event handlers things are even simpler as the original event handler can simply wxEvtHandler::Connect() or wxEvtHandler::Bind() the idle event handler which would only be executed then and could wxEvtHandler::Disconnect() or wxEvtHandler::Unbind() itself.

See also

Events and Event Handling, wxUpdateUIEvent, wxWindow::OnInternalIdle

Public Member Functions

- wxIdleEvent ()

  Constructor.
- bool MoreRequested () const

Returns true if the OnIdle function processing this event requested more processing time.

- void RequestMore (bool needMore=true)

  Tells wxWidgets that more processing is required.

Static Public Member Functions

- static wxIdleMode GetMode ()

  Static function returning a value specifying how wxWidgets will send idle events: to all windows, or only to those which specify that they will process the events.

- static void SetMode (wxIdleMode mode)

  Static function for specifying how wxWidgets will send idle events: to all windows, or only to those which specify that they will process the events.

Additional Inherited Members

21.396.3 Constructor & Destructor Documentation

wxIdleEvent::wxIdleEvent ( )

Constructor.

21.396.4 Member Function Documentation

static wxIdleMode wxIdleEvent::GetMode ( ) [static]

Static function returning a value specifying how wxWidgets will send idle events: to all windows, or only to those which specify that they will process the events.

See also

SetMode().

bool wxIdleEvent::MoreRequested ( ) const

Returns true if the OnIdle function processing this event requested more processing time.

See also

RequestMore()

void wxIdleEvent::RequestMore (bool *needMore* = true)

Tells wxWidgets that more processing is required.

This function can be called by an OnIdle handler for a window or window event handler to indicate that wxApp::↩ OnIdle should forward the OnIdle event once more to the application windows.

If no window calls this function during OnIdle, then the application will remain in a passive event loop (not calling OnIdle) until a new event is posted to the application by the windowing system.

See also

MoreRequested()

static void wxIdleEvent::SetMode (wxIdleMode *mode*) `[static]`

Static function for specifying how wxWidgets will send idle events: to all windows, or only to those which specify that they will process the events.

Parameters

mode	Can be one of the wxIdleMode values. The default is wxIDLE_PROCESS_ALL.

21.397 wxIdManager Class Reference

```
#include <wx/windowid.h>
```

21.397.1 Detailed Description

wxIdManager is responsible for allocating and releasing window IDs.

It is used by wxWindow::NewControlId() and wxWindow::UnreserveControlId(), and can also be used be used directly.

Library: wxCore

Category: Application and System configuration

See also

wxWindow::NewControlId(), wxWindow::UnreserveControlId(), Window IDs

Static Public Member Functions

- static wxWindowID ReserveId (int count=1)

  Called directly by wxWindow::NewControlId(), this function will create a new ID or range of IDs.

- static void UnreserveId (wxWindowID id, int count=1)

  Called directly by wxWindow::UnreserveControlId(), this function will unreserve an ID or range of IDs that is currently reserved.

21.397.2 Member Function Documentation

static wxWindowID wxIdManager::ReserveId (int *count* = 1) [static]

Called directly by wxWindow::NewControlId(), this function will create a new ID or range of IDs.

The IDs will be reserved until assigned to a wxWindowIDRef() or unreserved with UnreserveControlId(). Only ID values that are not assigned to a wxWindowIDRef() need to be unreserved.

Parameters

count	The number of sequential IDs to reserve.

Returns

The value of the first ID in the sequence, or wxID_NONE.

static void wxIdManager::UnreserveId (wxWindowID *id,* int *count* = 1) [static]

Called directly by wxWindow::UnreserveControlId(), this function will unreserve an ID or range of IDs that is currently reserved.

This should only be called for IDs returned by ReserveControlId() that have NOT been assigned to a wxWindowI↩ DRef (see Window IDs).

Parameters

id	The first of the range of IDs to unreserve.
count	The number of sequential IDs to unreserve.

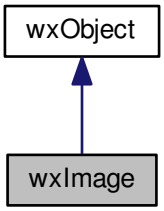
Returns

The value of the first ID in the sequence, or wxID_NONE.

21.398 wxImage Class Reference

`#include <wx/image.h>`

Inheritance diagram for wxImage:



21.398.1 Detailed Description

This class encapsulates a platform-independent image.

An image can be created from data, or using wxBitmap::ConvertToImage. An image can be loaded from a file in a variety of formats, and is extensible to new formats via image format handlers. Functions are available to set and get image bits, so it can be used for basic image manipulation.

A wxImage cannot (currently) be drawn directly to a wxDC. Instead, a platform-specific wxBitmap object must be created from it using the wxBitmap::wxBitmap(wxImage,int depth) constructor. This bitmap can then be drawn in a device context, using wxDC::DrawBitmap.

More on the difference between wxImage and wxBitmap: wxImage is just a buffer of RGB bytes with an optional buffer for the alpha bytes. It is all generic, platform independent and image file format independent code. It includes generic code for scaling, resizing, clipping, and other manipulations of the image data. OTOH, wxBitmap is intended to be a wrapper of whatever is the native image format that is quickest/easiest to draw to a DC or to be the target of the drawing operations performed on a wxMemoryDC. By splitting the responsibilities between wxImage/wxBitmap like this then it's easier to use generic code shared by all platforms and image types for generic operations and platform specific code where performance or compatibility is needed.

One colour value of the image may be used as a mask colour which will lead to the automatic creation of a wxMask object associated to the bitmap object.

21.398.2 Alpha channel support

Starting from wxWidgets 2.5.0 wxImage supports alpha channel data, that is in addition to a byte for the red, green and blue colour components for each pixel it also stores a byte representing the pixel opacity.

An alpha value of 0 corresponds to a transparent pixel (null opacity) while a value of 255 means that the pixel is 100% opaque. The constants wxIMAGE_ALPHA_TRANSPARENT and wxIMAGE_ALPHA_OPAQUE can be used to indicate those values in a more readable form.

While all images have RGB data, not all images have an alpha channel. Before using wxImage::GetAlpha you should check if this image contains an alpha channel with wxImage::HasAlpha. Currently the BMP, PNG, TGA, and TIFF format handlers have full alpha channel support for loading so if you want to use alpha you have to use one of these formats. If you initialize the image alpha channel yourself using wxImage::SetAlpha, you should save it in either PNG, TGA, or TIFF format to avoid losing it as these are the only handlers that currently support saving with alpha.

21.398.3 Available image handlers

The following image handlers are available. wxBMPHandler is always installed by default. To use other image formats, install the appropriate handler with wxImage::AddHandler or call wxInitAllImageHandlers().

- wxBMPHandler: For loading (including alpha support) and saving, always installed.
- wxPNGHandler: For loading and saving. Includes alpha support.
- wxJPEGHandler: For loading and saving.
- wxGIFHandler: For loading and saving (see below).
- wxPCXHandler: For loading and saving (see below).
- wxPNMHandler: For loading and saving (see below).
- wxTIFFHandler: For loading and saving. Includes alpha support.
- wxTGAHandler: For loading and saving. Includes alpha support.
- wxIFFHandler: For loading only.
- wxXPMHandler: For loading and saving.
- wxICOHandler: For loading and saving.
- wxCURHandler: For loading and saving.
- wxANIHandler: For loading only.

When saving in PCX format, wxPCXHandler will count the number of different colours in the image; if there are 256 or less colours, it will save as 8 bit, else it will save as 24 bit.

Loading PNMs only works for ASCII or raw RGB images. When saving in PNM format, wxPNMHandler will always save as raw RGB.

Saving GIFs requires images of maximum 8 bpp (see wxQuantize), and the alpha channel converted to a mask (see wxImage::ConvertAlphaToMask). Saving an animated GIF requires images of the same size (see wxGIFHandler↩ ::SaveAnimation)

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullImage

See also

wxBitmap, wxInitAllImageHandlers(), wxPixelData

Classes

- class HSVValue

  A simple class which stores hue, saturation and value as doubles in the range 0.0-1.0.
- class RGBValue

  A simple class which stores red, green and blue values as 8 bit unsigned integers in the range of 0-255.

Public Member Functions

- wxImage ()

  Creates an empty wxImage object without an alpha channel.
- wxImage (int width, int height, bool clear=true)

  Creates an image with the given size and clears it if requested.
- wxImage (const wxSize &sz, bool clear=true)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxImage (int width, int height, unsigned char ∗data, bool static_data=false)

  Creates an image from data in memory.
- wxImage (const wxSize &sz, unsigned char ∗data, bool static_data=false)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxImage (int width, int height, unsigned char ∗data, unsigned char ∗alpha, bool static_data=false)

  Creates an image from data in memory.
- wxImage (const wxSize &sz, unsigned char ∗data, unsigned char ∗alpha, bool static_data=false)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxImage (const char ∗const ∗xpmData)

  Creates an image from XPM data.
- wxImage (const wxString &name, wxBitmapType type=wxBITMAP_TYPE_ANY, int index=-1)

  Creates an image from a file.
- wxImage (const wxString &name, const wxString &mimetype, int index=-1)

  Creates an image from a file using MIME-types to specify the type.
- wxImage (wxInputStream &stream, wxBitmapType type=wxBITMAP_TYPE_ANY, int index=-1)

  Creates an image from a stream.
- wxImage (wxInputStream &stream, const wxString &mimetype, int index=-1)

  Creates an image from a stream using MIME-types to specify the type.
- virtual ∼wxImage ()

  Destructor.

Image creation, initialization and deletion functions

- wxImage Copy () const

  Returns an identical copy of this image.
- bool Create (int width, int height, bool clear=true)

  Creates a fresh image.
- bool Create (const wxSize &sz, bool clear=true)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool Create (int width, int height, unsigned char ∗data, bool static_data=false)

  Creates a fresh image.
- bool Create (const wxSize &sz, unsigned char ∗data, bool static_data=false)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool Create (int width, int height, unsigned char ∗data, unsigned char ∗alpha, bool static_data=false)

Creates a fresh image.

- bool Create (const wxSize &sz, unsigned char *data, unsigned char *alpha, bool static_data=false)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void Clear (unsigned char value=0)

  Initialize the image data with zeroes (the default) or with the byte value given as value.
- void Destroy ()

  Destroys the image data.
- void InitAlpha ()

  Initializes the image alpha channel data.

Image manipulation functions

- wxImage Blur (int blurRadius) const

  Blurs the image in both horizontal and vertical directions by the specified pixel blurRadius.
- wxImage BlurHorizontal (int blurRadius) const

  Blurs the image in the horizontal direction only.
- wxImage BlurVertical (int blurRadius) const

  Blurs the image in the vertical direction only.
- wxImage Mirror (bool horizontally=true) const

  Returns a mirrored copy of the image.
- void Paste (const wxImage &image, int x, int y)

  Copy the data of the given image to the specified position in this image.
- void Replace (unsigned char r1, unsigned char g1, unsigned char b1, unsigned char r2, unsigned char g2, unsigned char b2)

  Replaces the colour specified by r1,g1,b1 by the colour r2,g2,b2.
- wxImage & Rescale (int width, int height, wxImageResizeQuality quality=wxIMAGE_QUALITY_NORMAL)

  Changes the size of the image in-place by scaling it: after a call to this function,the image will have the given width and height.
- wxImage & Resize (const wxSize &size, const wxPoint &pos, int red=-1, int green=-1, int blue=-1)

  Changes the size of the image in-place without scaling it by adding either a border with the given colour or cropping as necessary.
- wxImage Rotate (double angle, const wxPoint &rotationCentre, bool interpolating=true, wxPoint *offset↩ AfterRotation=NULL) const

  Rotates the image about the given point, by angle radians.
- wxImage Rotate90 (bool clockwise=true) const

  Returns a copy of the image rotated 90 degrees in the direction indicated by clockwise.
- wxImage Rotate180 () const

  Returns a copy of the image rotated by 180 degrees.
- void RotateHue (double angle)

  Rotates the hue of each pixel in the image by angle, which is a double in the range of -1.0 to +1.0, where -1.0 corresponds to -360 degrees and +1.0 corresponds to +360 degrees.
- wxImage Scale (int width, int height, wxImageResizeQuality quality=wxIMAGE_QUALITY_NORMAL) const

  Returns a scaled version of the image.
- wxImage Size (const wxSize &size, const wxPoint &pos, int red=-1, int green=-1, int blue=-1) const

  Returns a resized version of this image without scaling it by adding either a border with the given colour or cropping as necessary.

Conversion functions

- bool ConvertAlphaToMask (unsigned char threshold=wxIMAGE_ALPHA_THRESHOLD)

  If the image has alpha channel, this method converts it to mask.
- bool ConvertAlphaToMask (unsigned char mr, unsigned char mg, unsigned char mb, unsigned char threshold=wxIMAGE_ALPHA_THRESHOLD)

  If the image has alpha channel, this method converts it to mask using the specified colour as the mask colour.
- wxImage ConvertToGreyscale (double weight_r, double weight_g, double weight_b) const

  Returns a greyscale version of the image.
- wxImage ConvertToGreyscale () const

Returns a greyscale version of the image.

- wxImage ConvertToMono (unsigned char r, unsigned char g, unsigned char b) const
  Returns monochromatic version of the image.
- wxImage ConvertToDisabled (unsigned char brightness=255) const
  Returns disabled (dimmed) version of the image.

Miscellaneous functions

- unsigned long ComputeHistogram (wxImageHistogram &histogram) const
  Computes the histogram of the image.
- bool FindFirstUnusedColour (unsigned char *r, unsigned char *g, unsigned char *b, unsigned char startR=1, unsigned char startG=0, unsigned char startB=0) const
  Finds the first colour that is never used in the image.
- wxImage & operator= (const wxImage &image)
  Assignment operator, using reference counting.

Getters

- unsigned char * GetAlpha () const
  Returns pointer to the array storing the alpha values for this image.
- unsigned char * GetData () const
  Returns the image data as an array.
- unsigned char GetAlpha (int x, int y) const
  Return alpha value at given pixel location.
- unsigned char GetRed (int x, int y) const
  Returns the red intensity at the given coordinate.
- unsigned char GetGreen (int x, int y) const
  Returns the green intensity at the given coordinate.
- unsigned char GetBlue (int x, int y) const
  Returns the blue intensity at the given coordinate.
- unsigned char GetMaskRed () const
  Gets the red value of the mask colour.
- unsigned char GetMaskGreen () const
  Gets the green value of the mask colour.
- unsigned char GetMaskBlue () const
  Gets the blue value of the mask colour.
- int GetWidth () const
  Gets the width of the image in pixels.
- int GetHeight () const
  Gets the height of the image in pixels.
- wxSize GetSize () const
  Returns the size of the image in pixels.
- wxString GetOption (const wxString &name) const
  Gets a user-defined string-valued option.
- int GetOptionInt (const wxString &name) const
  Gets a user-defined integer-valued option.
- bool GetOrFindMaskColour (unsigned char *r, unsigned char *g, unsigned char *b) const
  Get the current mask colour or find a suitable unused colour that could be used as a mask colour.
- const wxPalette & GetPalette () const
  Returns the palette associated with the image.
- wxImage GetSubImage (const wxRect &rect) const
  Returns a sub image of the current one as long as the rect belongs entirely to the image.
- wxBitmapType GetType () const
  Gets the type of image found by LoadFile() or specified with SaveFile().
- bool HasAlpha () const
  Returns true if this image has alpha channel, false otherwise.
- bool HasMask () const
  Returns true if there is a mask active, false otherwise.
- bool HasOption (const wxString &name) const

Returns true if the given option is present.

- bool IsOk () const

  Returns true if image data is present.
- bool IsTransparent (int x, int y, unsigned char threshold=wxIMAGE_ALPHA_THRESHOLD) const

  Returns true if the given pixel is transparent, i.e. either has the mask colour if this image has a mask or if this image has alpha channel and alpha value of this pixel is strictly less than threshold.

Loading and saving functions

- virtual bool LoadFile (wxInputStream &stream, wxBitmapType type=wxBITMAP_TYPE_ANY, int index=-1)

  Loads an image from an input stream.
- virtual bool LoadFile (const wxString &name, wxBitmapType type=wxBITMAP_TYPE_ANY, int index=-1)

  Loads an image from a file.
- virtual bool LoadFile (const wxString &name, const wxString &mimetype, int index=-1)

  Loads an image from a file.
- virtual bool LoadFile (wxInputStream &stream, const wxString &mimetype, int index=-1)

  Loads an image from an input stream.
- virtual bool SaveFile (wxOutputStream &stream, const wxString &mimetype) const

  Saves an image in the given stream.
- virtual bool SaveFile (const wxString &name, wxBitmapType type) const

  Saves an image in the named file.
- virtual bool SaveFile (const wxString &name, const wxString &mimetype) const

  Saves an image in the named file.
- virtual bool SaveFile (const wxString &name) const

  Saves an image in the named file.
- virtual bool SaveFile (wxOutputStream &stream, wxBitmapType type) const

  Saves an image in the given stream.

Setters

- void SetAlpha (unsigned char *alpha=NULL, bool static_data=false)

  This function is similar to SetData() and has similar restrictions.
- void SetAlpha (int x, int y, unsigned char alpha)

  Sets the alpha value for the given pixel.
- void ClearAlpha ()

  Removes the alpha channel from the image.
- void SetData (unsigned char *data, bool static_data=false)

  Sets the image data without performing checks.
- void SetData (unsigned char *data, int new_width, int new_height, bool static_data=false)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetMask (bool hasMask=true)

  Specifies whether there is a mask or not.
- void SetMaskColour (unsigned char red, unsigned char green, unsigned char blue)

  Sets the mask colour for this image (and tells the image to use the mask).
- bool SetMaskFromImage (const wxImage &mask, unsigned char mr, unsigned char mg, unsigned char mb)

  Sets image's mask so that the pixels that have RGB value of mr,mg,mb in mask will be masked in the image.
- void SetOption (const wxString &name, const wxString &value)

  Sets a user-defined option.
- void SetOption (const wxString &name, int value)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetPalette (const wxPalette &palette)

  Associates a palette with the image.
- void SetRGB (int x, int y, unsigned char r, unsigned char g, unsigned char b)

  Set the color of the pixel at the given x and y coordinate.
- void SetRGB (const wxRect &rect, unsigned char red, unsigned char green, unsigned char blue)

  Sets the colour of the pixels within the given rectangle.
- void SetType (wxBitmapType type)

  Set the type of image returned by GetType().

Static Public Member Functions

- static bool CanRead (const wxString &filename)

  Returns true if at least one of the available image handlers can read the file with the given name.
- static bool CanRead (wxInputStream &stream)

  Returns true if at least one of the available image handlers can read the data in the given stream.
- static wxString GetImageExtWildcard ()

  Iterates all registered wxImageHandler objects, and returns a string containing file extension masks suitable for passing to file open/save dialog boxes.
- static wxImage::HSVValue RGBtoHSV (const wxImage::RGBValue &rgb)

  Converts a color in RGB color space to HSV color space.
- static wxImage::RGBValue HSVtoRGB (const wxImage::HSVValue &hsv)

  Converts a color in HSV color space to RGB color space.

Handler management functions

- static void AddHandler (wxImageHandler *handler)

  Register an image handler.
- static void CleanUpHandlers ()

  Deletes all image handlers.
- static wxImageHandler * FindHandler (const wxString &name)

  Finds the handler with the given name.
- static wxImageHandler * FindHandler (const wxString &extension, wxBitmapType imageType)

  Finds the handler associated with the given extension and type.
- static wxImageHandler * FindHandler (wxBitmapType imageType)

  Finds the handler associated with the given image type.
- static wxImageHandler * FindHandlerMime (const wxString &mimetype)

  Finds the handler associated with the given MIME type.
- static wxList & GetHandlers ()

  Returns the static list of image format handlers.
- static void InitStandardHandlers ()

  Internal use only.
- static void InsertHandler (wxImageHandler *handler)

  Adds a handler at the start of the static list of format handlers.
- static bool RemoveHandler (const wxString &name)

  Finds the handler with the given name, and removes it.

- static int GetImageCount (const wxString &filename, wxBitmapType type=wxBITMAP_TYPE_ANY)

  If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.
- static int GetImageCount (wxInputStream &stream, wxBitmapType type=wxBITMAP_TYPE_ANY)

  If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.

Additional Inherited Members

21.398.4 Constructor & Destructor Documentation

wxImage::wxImage ( )

Creates an empty wxImage object without an alpha channel.

wxImage::wxImage (int *width*, int *height*, bool *clear* = `true`)

Creates an image with the given size and clears it if requested.

Does not create an alpha channel.

Parameters

width	Specifies the width of the image.
height	Specifies the height of the image.
clear	If true, initialize the image to black.

wxImage::wxImage (const wxSize & *sz*, bool *clear* = `true`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxImage::wxImage (int *width*, int *height*, unsigned char * *data*, bool *static_data* = `false`)

Creates an image from data in memory.

If *static_data* is false then the wxImage will take ownership of the data and free it afterwards. For this, it has to be allocated with *malloc*.

Parameters

width	Specifies the width of the image.
height	Specifies the height of the image.
data	A pointer to RGB data
static_data	Indicates if the data should be free'd after use

wxImage::wxImage (const wxSize & *sz*, unsigned char * *data*, bool *static_data* = `false`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxImage::wxImage (int *width*, int *height*, unsigned char * *data*, unsigned char * *alpha*, bool *static_data* = `false`)

Creates an image from data in memory.

If *static_data* is false then the wxImage will take ownership of the data and free it afterwards. For this, it has to be allocated with *malloc*.

Parameters

width	Specifies the width of the image.
height	Specifies the height of the image.
data	A pointer to RGB data
alpha	A pointer to alpha-channel data
static_data	Indicates if the data should be free'd after use

wxImage::wxImage (const wxSize & *sz*, unsigned char * *data*, unsigned char * *alpha*, bool *static_data* = `false`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxImage::wxImage (const char *const * *xpmData*)

Creates an image from XPM data.

Parameters

xpmData	A pointer to XPM image data.

wxPerl Note: Not supported by wxPerl.

wxImage::wxImage (const wxString & *name*, wxBitmapType *type* = wxBITMAP_TYPE_ANY, int *index* = -1)

Creates an image from a file.

Parameters

name	Name of the file from which to load the image.
type	May be one of the following: • wxBITMAP_TYPE_BMP: Load a Windows bitmap file. • wxBITMAP_TYPE_GIF: Load a GIF bitmap file. • wxBITMAP_TYPE_JPEG: Load a JPEG bitmap file. • wxBITMAP_TYPE_PNG: Load a PNG bitmap file. • wxBITMAP_TYPE_PCX: Load a PCX bitmap file. • wxBITMAP_TYPE_PNM: Load a PNM bitmap file. • wxBITMAP_TYPE_TIFF: Load a TIFF bitmap file. • wxBITMAP_TYPE_TGA: Load a TGA bitmap file. • wxBITMAP_TYPE_XPM: Load a XPM bitmap file. • wxBITMAP_TYPE_ICO: Load a Windows icon file (ICO). • wxBITMAP_TYPE_CUR: Load a Windows cursor file (CUR). • wxBITMAP_TYPE_ANI: Load a Windows animated cursor file (ANI). • wxBITMAP_TYPE_ANY: Will try to autodetect the format.

index	Index of the image to load in the case that the image file contains multiple images. This is only used by GIF, ICO and TIFF handlers. The default value (-1) means "choose the default image" and is interpreted as the first image (index=0) by the GIF and TIFF handler and as the largest and most colourful one by the ICO handler.

Remarks

Depending on how wxWidgets has been configured and by which handlers have been loaded, not all formats may be available. Any handler other than BMP must be previously initialized with wxImage::AddHandler or wxInitAllImageHandlers.

Note

You can use GetOptionInt() to get the hotspot when loading cursor files:

```
int hotspot_x = image.GetOptionInt(wxIMAGE_OPTION_CUR_HOTSPOT_X);
int hotspot_y = image.GetOptionInt(wxIMAGE_OPTION_CUR_HOTSPOT_Y);
```

See also

LoadFile()

wxImage::wxImage (const wxString & *name*, const wxString & *mimetype*, int *index* = -1)

Creates an image from a file using MIME-types to specify the type.

Parameters

name	Name of the file from which to load the image.
mimetype	MIME type string (for example 'image/jpeg')
index	See description in wxImage(const wxString&, wxBitmapType, int) overload.

wxImage::wxImage (wxInputStream & *stream*, wxBitmapType *type* = wxBITMAP_TYPE_ANY, int *index* = -1)

Creates an image from a stream.

Parameters

stream	Opened input stream from which to load the image. Currently, the stream must support seeking.
type	See description in wxImage(const wxString&, wxBitmapType, int) overload.
index	See description in wxImage(const wxString&, wxBitmapType, int) overload.

wxImage::wxImage (wxInputStream & *stream*, const wxString & *mimetype*, int *index* = -1)

Creates an image from a stream using MIME-types to specify the type.

Parameters

stream	Opened input stream from which to load the image. Currently, the stream must support seeking.

mimetype	MIME type string (for example 'image/jpeg')
index	See description in wxImage(const wxString&, wxBitmapType, int) overload.

virtual wxImage::~wxImage ( ) [virtual]

Destructor.

See reference-counted object destruction for more info.

21.398.5 Member Function Documentation

static void wxImage::AddHandler (wxImageHandler * *handler*) [static]

Register an image handler.

Typical example of use:

```
wxImage::AddHandler(new wxPNGHandler);
```

See Available image handlers for a list of the available handlers. You can also use wxInitAllImageHandlers() to add handlers for all the image formats supported by wxWidgets at once.

Parameters

handler	A heap-allocated handler object which will be deleted by wxImage if it is removed later by RemoveHandler() or at program shutdown.

wxImage wxImage::Blur (int *blurRadius*) const

Blurs the image in both horizontal and vertical directions by the specified pixel *blurRadius*.

This should not be used when using a single mask colour for transparency.

See also

BlurHorizontal(), BlurVertical()

wxImage wxImage::BlurHorizontal (int *blurRadius*) const

Blurs the image in the horizontal direction only.

This should not be used when using a single mask colour for transparency.

See also

Blur(), BlurVertical()

wxImage wxImage::BlurVertical (int *blurRadius*) const

Blurs the image in the vertical direction only.

This should not be used when using a single mask colour for transparency.

See also

Blur(), BlurHorizontal()

static bool wxImage::CanRead (const wxString & *filename*) `[static]`

Returns true if at least one of the available image handlers can read the file with the given name.

See wxImageHandler::CanRead for more info.

static bool wxImage::CanRead (wxInputStream & *stream*) `[static]`

Returns true if at least one of the available image handlers can read the data in the given stream.

See wxImageHandler::CanRead for more info.

static void wxImage::CleanUpHandlers ( ) `[static]`

Deletes all image handlers.

This function is called by wxWidgets on exit.

void wxImage::Clear (unsigned char *value* = 0)

Initialize the image data with zeroes (the default) or with the byte value given as *value*.

Since

2.9.0

void wxImage::ClearAlpha ( )

Removes the alpha channel from the image.

This function should only be called if the image has alpha channel data, use HasAlpha() to check for this.

Since

2.9.1

unsigned long wxImage::ComputeHistogram (wxImageHistogram & *histogram*) const

Computes the histogram of the image.

histogram is a reference to wxImageHistogram object. wxImageHistogram is a specialization of wxHashMap "template" and is defined as follows:

```
class WXDLLEXPORT wxImageHistogramEntry
{
public:
    wxImageHistogramEntry() : index(0), value(0) {}
    unsigned long index;
    unsigned long value;
};

WX_DECLARE_EXPORTED_HASH_MAP(unsigned long, wxImageHistogramEntry,
                             wxIntegerHash, wxIntegerEqual,
                             wxImageHistogram);
```

Returns

Returns number of colours in the histogram.

bool wxImage::ConvertAlphaToMask (unsigned char *threshold* = wxIMAGE_ALPHA_THRESHOLD)

If the image has alpha channel, this method converts it to mask.

If the image has an alpha channel, all pixels with alpha value less than *threshold* are replaced with the mask colour and the alpha channel is removed. Otherwise nothing is done.

The mask colour is chosen automatically using FindFirstUnusedColour() by this function, see the overload below if you this is not appropriate.

Returns

Returns true on success, false on error.

bool wxImage::ConvertAlphaToMask (unsigned char *mr*, unsigned char *mg*, unsigned char *mb*, unsigned char *threshold* = wxIMAGE_ALPHA_THRESHOLD)

If the image has alpha channel, this method converts it to mask using the specified colour as the mask colour.

If the image has an alpha channel, all pixels with alpha value less than *threshold* are replaced with the mask colour and the alpha channel is removed. Otherwise nothing is done.

Since

2.9.0

Parameters

mr	The red component of the mask colour.
mg	The green component of the mask colour.
mb	The blue component of the mask colour.
threshold	Pixels with alpha channel values below the given threshold are considered to be transparent, i.e. the corresponding mask pixels are set. Pixels with the alpha values above the threshold are considered to be opaque.

Returns

Returns true on success, false on error.

wxImage wxImage::ConvertToDisabled (unsigned char *brightness* = 255) const

Returns disabled (dimmed) version of the image.

Since

2.9.0

wxImage wxImage::ConvertToGreyscale (double *weight_r*, double *weight_g*, double *weight_b*) const

Returns a greyscale version of the image.

The returned image uses the luminance component of the original to calculate the greyscale. Defaults to using the standard ITU-T BT.601 when converting to YUV, where every pixel equals (R $*$ *weight_r*) + (G $*$ *weight_g*) + (B $*$ *weight_b*).

wxImage wxImage::ConvertToGreyscale ( ) const

Returns a greyscale version of the image.

Since

2.9.0

wxImage wxImage::ConvertToMono (unsigned char *r*, unsigned char *g*, unsigned char *b*) const

Returns monochromatic version of the image.

The returned image has white colour where the original has (r,g,b) colour and black colour everywhere else.

wxImage wxImage::Copy ( ) const

Returns an identical copy of this image.

bool wxImage::Create (int *width*, int *height*, bool *clear* = `true`)

Creates a fresh image.

See wxImage::wxImage(int,int,bool) for more info.

Returns

true if the call succeeded, false otherwise.

bool wxImage::Create (const wxSize & *sz*, bool *clear* = `true`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxImage::Create (int *width*, int *height*, unsigned char * *data*, bool *static_data* = `false`)

Creates a fresh image.

See wxImage::wxImage(int,int,unsigned char*,bool) for more info.

Returns

true if the call succeeded, false otherwise.

bool wxImage::Create (const wxSize & *sz*, unsigned char * *data*, bool *static_data* = `false`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxImage::Create (int *width*, int *height*, unsigned char * *data*, unsigned char * *alpha*, bool *static_data* = `false`)

Creates a fresh image.

See wxImage::wxImage(int,int,unsigned char*,unsigned char*,bool) for more info.

Returns

true if the call succeeded, false otherwise.

bool wxImage::Create (const wxSize & *sz*, unsigned char * *data*, unsigned char * *alpha*, bool *static_data* = `false`)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxImage::Destroy ( )

Destroys the image data.

bool wxImage::FindFirstUnusedColour (unsigned char * *r*, unsigned char * *g*, unsigned char * *b*, unsigned char *startR* = `1`, unsigned char *startG* = `0`, unsigned char *startB* = `0`) const

Finds the first colour that is never used in the image.

The search begins at given initial colour and continues by increasing R, G and B components (in this order) by 1 until an unused colour is found or the colour space exhausted.

The parameters *r*, *g*, *b* are pointers to variables to save the colour.

The parameters *startR*, *startG*, *startB* define the initial values of the colour. The returned colour will have RGB values equal to or greater than these.

Returns

Returns false if there is no unused colour left, true on success.

Note

This method involves computing the histogram, which is a computationally intensive operation.

static wxImageHandler* wxImage::FindHandler (const wxString & *name*) `[static]`

Finds the handler with the given name.

Parameters

name	The handler name.

Returns

A pointer to the handler if found, NULL otherwise.

See also

wxImageHandler

static wxImageHandler* wxImage::FindHandler (const wxString & *extension*, wxBitmapType *imageType*) `[static]`

Finds the handler associated with the given extension and type.

Parameters

extension	The file extension, such as "bmp".
imageType	The image type; one of the wxBitmapType values.

Returns

A pointer to the handler if found, NULL otherwise.

See also

wxImageHandler

static wxImageHandler* wxImage::FindHandler (wxBitmapType *imageType*) [static]

Finds the handler associated with the given image type.

Parameters

imageType	The image type; one of the wxBitmapType values.

Returns

A pointer to the handler if found, NULL otherwise.

See also

wxImageHandler

static wxImageHandler* wxImage::FindHandlerMime (const wxString & *mimetype*) [static]

Finds the handler associated with the given MIME type.

Parameters

mimetype	MIME type.

Returns

A pointer to the handler if found, NULL otherwise.

See also

wxImageHandler

unsigned char* wxImage::GetAlpha ( ) const

Returns pointer to the array storing the alpha values for this image.

This pointer is NULL for the images without the alpha channel. If the image does have it, this pointer may be used to directly manipulate the alpha values which are stored as the RGB ones.

unsigned char wxImage::GetAlpha (int *x*, int *y*) const

Return alpha value at given pixel location.

unsigned char wxImage::GetBlue (int *x*, int *y*) const

Returns the blue intensity at the given coordinate.

unsigned char* wxImage::GetData ( ) const

Returns the image data as an array.

This is most often used when doing direct image manipulation. The return value points to an array of characters in RGBRGBRGB... format in the top-to-bottom, left-to-right order, that is the first RGB triplet corresponds to the first pixel of the first row, the second one — to the second pixel of the first row and so on until the end of the first row, with second row following after it and so on.

You should not delete the returned pointer nor pass it to SetData().

unsigned char wxImage::GetGreen (int *x*, int *y*) const

Returns the green intensity at the given coordinate.

static wxList& wxImage::GetHandlers ( ) [static]

Returns the static list of image format handlers.

See also

wxImageHandler

int wxImage::GetHeight ( ) const

Gets the height of the image in pixels.

See also

GetWidth(), GetSize()

static int wxImage::GetImageCount (const wxString & *filename*, wxBitmapType *type* = wxBITMAP_TYPE_ANY) [static]

If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.

For the overload taking the parameter *filename*, that's the name of the file to query. For the overload taking the parameter *stream*, that's the opened input stream with image data.

See wxImageHandler::GetImageCount() for more info.

The parameter *type* may be one of the following values:

- wxBITMAP_TYPE_BMP: Load a Windows bitmap file.
- wxBITMAP_TYPE_GIF: Load a GIF bitmap file.
- wxBITMAP_TYPE_JPEG: Load a JPEG bitmap file.
- wxBITMAP_TYPE_PNG: Load a PNG bitmap file.
- wxBITMAP_TYPE_PCX: Load a PCX bitmap file.

- wxBITMAP_TYPE_PNM: Load a PNM bitmap file.
- wxBITMAP_TYPE_TIFF: Load a TIFF bitmap file.
- wxBITMAP_TYPE_TGA: Load a TGA bitmap file.
- wxBITMAP_TYPE_XPM: Load a XPM bitmap file.
- wxBITMAP_TYPE_ICO: Load a Windows icon file (ICO).
- wxBITMAP_TYPE_CUR: Load a Windows cursor file (CUR).
- wxBITMAP_TYPE_ANI: Load a Windows animated cursor file (ANI).
- wxBITMAP_TYPE_ANY: Will try to autodetect the format.

Returns

Number of available images. For most image handlers, this is 1 (exceptions are TIFF and ICO formats as well as animated GIFs for which this function returns the number of frames in the animation).

static int wxImage::GetImageCount (wxInputStream & *stream,* wxBitmapType *type* = wxBITMAP_TYPE_ANY) `[static]`

If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.

For the overload taking the parameter *filename*, that's the name of the file to query. For the overload taking the parameter *stream*, that's the opened input stream with image data.

See wxImageHandler::GetImageCount() for more info.

The parameter *type* may be one of the following values:

- wxBITMAP_TYPE_BMP: Load a Windows bitmap file.
- wxBITMAP_TYPE_GIF: Load a GIF bitmap file.
- wxBITMAP_TYPE_JPEG: Load a JPEG bitmap file.
- wxBITMAP_TYPE_PNG: Load a PNG bitmap file.
- wxBITMAP_TYPE_PCX: Load a PCX bitmap file.
- wxBITMAP_TYPE_PNM: Load a PNM bitmap file.
- wxBITMAP_TYPE_TIFF: Load a TIFF bitmap file.
- wxBITMAP_TYPE_TGA: Load a TGA bitmap file.
- wxBITMAP_TYPE_XPM: Load a XPM bitmap file.
- wxBITMAP_TYPE_ICO: Load a Windows icon file (ICO).
- wxBITMAP_TYPE_CUR: Load a Windows cursor file (CUR).
- wxBITMAP_TYPE_ANI: Load a Windows animated cursor file (ANI).
- wxBITMAP_TYPE_ANY: Will try to autodetect the format.

Returns

Number of available images. For most image handlers, this is 1 (exceptions are TIFF and ICO formats as well as animated GIFs for which this function returns the number of frames in the animation).

static wxString wxImage::GetImageExtWildcard ( ) [static]

Iterates all registered wxImageHandler objects, and returns a string containing file extension masks suitable for passing to file open/save dialog boxes.

Returns

The format of the returned string is "(*.ext1;*.ext2)|*.ext1;*.ext2". It is usually a good idea to prepend a description before passing the result to the dialog. Example:

```
wxFileDialog FileDlg( this, "Choose Image", ::wxGetCwd(), "",
                      _("Image Files ") + wxImage::GetImageExtWildcard(),
                      wxFD_OPEN );
```

See also

wxImageHandler

unsigned char wxImage::GetMaskBlue ( ) const

Gets the blue value of the mask colour.

unsigned char wxImage::GetMaskGreen ( ) const

Gets the green value of the mask colour.

unsigned char wxImage::GetMaskRed ( ) const

Gets the red value of the mask colour.

wxString wxImage::GetOption (const wxString & *name*) const

Gets a user-defined string-valued option.

Generic options:

- wxIMAGE_OPTION_FILENAME: The name of the file from which the image was loaded.

Options specific to wxGIFHandler:

- wxIMAGE_OPTION_GIF_COMMENT: The comment text that is read from or written to the GIF file. In an animated GIF each frame can have its own comment. If there is only a comment in the first frame of a GIF it will not be repeated in other frames.

Parameters

name	The name of the option, case-insensitive.

Returns

The value of the option or an empty string if not found. Use HasOption() if an empty string can be a valid option value.

See also

SetOption(), GetOptionInt(), HasOption()

int wxImage::GetOptionInt (const wxString & *name*) const

Gets a user-defined integer-valued option.

The function is case-insensitive to *name*. If the given option is not present, the function returns 0. Use HasOption() if 0 is a possibly valid value for the option.

Generic options:

- `wxIMAGE_OPTION_MAX_WIDTH` and `wxIMAGE_OPTION_MAX_HEIGHT`: If either of these options is specified, the loaded image will be scaled down (preserving its aspect ratio) so that its width is less than the max width given if it is not 0 *and* its height is less than the max height given if it is not 0. This is typically used for loading thumbnails and the advantage of using these options compared to calling Rescale() after loading is that some handlers (only JPEG one right now) support rescaling the image during loading which is vastly more efficient than loading the entire huge image and rescaling it later (if these options are not supported by the handler, this is still what happens however). These options must be set before calling LoadFile() to have any effect.

- `wxIMAGE_OPTION_ORIGINAL_WIDTH` and `wxIMAGE_OPTION_ORIGINAL_HEIGHT`: These options will return the original size of the image if either `wxIMAGE_OPTION_MAX_WIDTH` or `wxIMAGE_OPTION_MAX_HEIGHT` is specified.

  Since

  2.9.3

- `wxIMAGE_OPTION_QUALITY`: JPEG quality used when saving. This is an integer in 0..100 range with 0 meaning very poor and 100 excellent (but very badly compressed). This option is currently ignored for the other formats.

- `wxIMAGE_OPTION_RESOLUTIONUNIT`: The value of this option determines whether the resolution of the image is specified in centimetres or inches, see wxImageResolution enum elements.

- `wxIMAGE_OPTION_RESOLUTION`, `wxIMAGE_OPTION_RESOLUTIONX` and `wxIMAGE_OPTION_RESOLUTIONY`: These options define the resolution of the image in the units corresponding to `wxIMAGE_OPTION_RESOLUTIONUNIT` options value. The first option can be set before saving the image to set both horizontal and vertical resolution to the same value. The X and Y options are set by the image handlers if they support the image resolution (currently BMP, JPEG and TIFF handlers do) and the image provides the resolution information and can be queried after loading the image.

Options specific to wxPNGHandler:

- `wxIMAGE_OPTION_PNG_FORMAT`: Format for saving a PNG file, see wxImagePNGType for the supported values.
- `wxIMAGE_OPTION_PNG_BITDEPTH`: Bit depth for every channel (R/G/B/A).
- `wxIMAGE_OPTION_PNG_FILTER`: Filter for saving a PNG file, see libpng (`http://www.libpng.org/pub/png/libpng-1.2.5-manual.html`) for possible values (e.g. PNG_FILTER_NONE, PNG_FILTER_SUB, PNG_FILTER_UP, etc).
- `wxIMAGE_OPTION_PNG_COMPRESSION_LEVEL`: Compression level (0..9) for saving a PNG file. An high value creates smaller-but-slower PNG file. Note that unlike other formats (e.g. JPEG) the PNG format is always lossless and thus this compression level doesn't tradeoff the image quality.
- `wxIMAGE_OPTION_PNG_COMPRESSION_MEM_LEVEL`: Compression memory usage level (1..9) for saving a PNG file. An high value means the saving process consumes more memory, but may create smaller PNG file.
- `wxIMAGE_OPTION_PNG_COMPRESSION_STRATEGY`: Possible values are 0 for default strategy, 1 for filter, and 2 for Huffman-only. You can use OptiPNG (`http://optipng.sourceforge.net/`) to get a suitable value for your application.

- wxIMAGE_OPTION_PNG_COMPRESSION_BUFFER_SIZE: Internal buffer size (in bytes) for saving a PNG file. Ideally this should be as big as the resulting PNG file. Use this option if your application produces images with small size variation.

Options specific to wxTIFFHandler:

- wxIMAGE_OPTION_TIFF_BITSPERSAMPLE: Number of bits per sample (channel). Currently values of 1 and 8 are supported. A value of 1 results in a black and white image. A value of 8 (the default) can mean greyscale or RGB, depending on the value of wxIMAGE_OPTION_TIFF_SAMPLESPERPIXEL.
- wxIMAGE_OPTION_TIFF_SAMPLESPERPIXEL: Number of samples (channels) per pixel. Currently values of 1 and 3 are supported. A value of 1 results in either a greyscale (by default) or black and white image, depending on the value of wxIMAGE_OPTION_TIFF_BITSPERSAMPLE. A value of 3 (the default) will result in an RGB image.
- wxIMAGE_OPTION_TIFF_COMPRESSION: Compression type. By default it is set to 1 (COMPRESSION_NONE). Typical other values are 5 (COMPRESSION_LZW) and 7 (COMPRESSION_JPEG). See tiff.h for more options.
- wxIMAGE_OPTION_TIFF_PHOTOMETRIC: Specifies the photometric interpretation. By default it is set to 2 (PHOTOMETRIC_RGB) for RGB images and 0 (PHOTOMETRIC_MINISWHITE) for greyscale or black and white images. It can also be set to 1 (PHOTOMETRIC_MINISBLACK) to treat the lowest value as black and highest as white. If you want a greyscale image it is also sufficient to only specify wxIMAGE_OPTION_TIFF_PHOTOMETRIC and set it to either PHOTOMETRIC_MINISWHITE or PHOTOMETRIC_MINISBLACK. The other values are taken care of.

Note

Be careful when combining the options wxIMAGE_OPTION_TIFF_SAMPLESPERPIXEL, wxIMAGE_OPTION_TIFF_BITSPERSAMPLE, and wxIMAGE_OPTION_TIFF_PHOTOMETRIC. While some measures are taken to prevent illegal combinations and/or values, it is still easy to abuse them and come up with invalid results in the form of either corrupted images or crashes.

Parameters

name	The name of the option, case-insensitive.

Returns

The value of the option or 0 if not found. Use HasOption() if 0 can be a valid option value.

See also

SetOption(), GetOption()

bool wxImage::GetOrFindMaskColour (unsigned char * *r*, unsigned char * *g*, unsigned char * *b*) const

Get the current mask colour or find a suitable unused colour that could be used as a mask colour.

Returns true if the image currently has a mask.

const wxPalette& wxImage::GetPalette ( ) const

Returns the palette associated with the image.

Currently the palette is only used when converting to wxBitmap under Windows.

Some of the wxImage handlers have been modified to set the palette if one exists in the image file (usually 256 or less colour images in GIF or PNG format).

unsigned char wxImage::GetRed (int *x*, int *y*) const

Returns the red intensity at the given coordinate.

wxSize wxImage::GetSize ( ) const

Returns the size of the image in pixels.

Since

2.9.0

See also

GetHeight(), GetWidth()

wxImage wxImage::GetSubImage (const wxRect & *rect*) const

Returns a sub image of the current one as long as the rect belongs entirely to the image.

wxBitmapType wxImage::GetType ( ) const

Gets the type of image found by LoadFile() or specified with SaveFile().

Since

2.9.0

int wxImage::GetWidth ( ) const

Gets the width of the image in pixels.

See also

GetHeight(), GetSize()

bool wxImage::HasAlpha ( ) const

Returns true if this image has alpha channel, false otherwise.

See also

GetAlpha(), SetAlpha()

bool wxImage::HasMask ( ) const

Returns true if there is a mask active, false otherwise.

bool wxImage::HasOption (const wxString & *name*) const

Returns true if the given option is present.

The function is case-insensitive to *name*.

The lists of the currently supported options are in GetOption() and GetOptionInt() function docs.

See also

SetOption(), GetOption(), GetOptionInt()

static wxImage::RGBValue wxImage::HSVtoRGB (const wxImage::HSVValue & *hsv*) [static]

Converts a color in HSV color space to RGB color space.

void wxImage::InitAlpha ( )

Initializes the image alpha channel data.

It is an error to call it if the image already has alpha data. If it doesn't, alpha data will be by default initialized to all pixels being fully opaque. But if the image has a mask colour, all mask pixels will be completely transparent.

static void wxImage::InitStandardHandlers ( ) [static]

Internal use only.

Adds standard image format handlers. It only install wxBMPHandler for the time being, which is used by wxBitmap.

This function is called by wxWidgets on startup, and shouldn't be called by the user.

See also

wxImageHandler, wxInitAllImageHandlers(), wxQuantize

static void wxImage::InsertHandler (wxImageHandler * *handler*) [static]

Adds a handler at the start of the static list of format handlers.

Parameters

handler	A new image format handler object. There is usually only one instance of a given handler class in an application session.

See also

wxImageHandler

bool wxImage::IsOk ( ) const

Returns true if image data is present.

bool wxImage::IsTransparent (int *x*, int *y*, unsigned char *threshold* = wxIMAGE_ALPHA_THRESHOLD) const

Returns true if the given pixel is transparent, i.e. either has the mask colour if this image has a mask or if this image has alpha channel and alpha value of this pixel is strictly less than *threshold*.

virtual bool wxImage::LoadFile (wxInputStream & *stream,* wxBitmapType *type* = wxBITMAP_TYPE_ANY, int *index* = -1) [virtual]

Loads an image from an input stream.

Parameters

stream	Opened input stream from which to load the image. Currently, the stream must support seeking.
type	May be one of the following: • wxBITMAP_TYPE_BMP: Load a Windows bitmap file. • wxBITMAP_TYPE_GIF: Load a GIF bitmap file. • wxBITMAP_TYPE_JPEG: Load a JPEG bitmap file. • wxBITMAP_TYPE_PNG: Load a PNG bitmap file. • wxBITMAP_TYPE_PCX: Load a PCX bitmap file. • wxBITMAP_TYPE_PNM: Load a PNM bitmap file. • wxBITMAP_TYPE_TIFF: Load a TIFF bitmap file. • wxBITMAP_TYPE_TGA: Load a TGA bitmap file. • wxBITMAP_TYPE_XPM: Load a XPM bitmap file. • wxBITMAP_TYPE_ICO: Load a Windows icon file (ICO). • wxBITMAP_TYPE_CUR: Load a Windows cursor file (CUR). • wxBITMAP_TYPE_ANI: Load a Windows animated cursor file (ANI). • wxBITMAP_TYPE_ANY: Will try to autodetect the format.

index	Index of the image to load in the case that the image file contains multiple images. This is only used by GIF, ICO and TIFF handlers. The default value (-1) means "choose the default image" and is interpreted as the first image (index=0) by the GIF and TIFF handler and as the largest and most colourful one by the ICO handler.

Returns

true if the operation succeeded, false otherwise. If the optional index parameter is out of range, false is returned and a call to wxLogError() takes place.

Remarks

Depending on how wxWidgets has been configured, not all formats may be available.

Note

You can use GetOptionInt() to get the hotspot when loading cursor files:

```
int hotspot_x = image.GetOptionInt(wxIMAGE_OPTION_CUR_HOTSPOT_X);
int hotspot_y = image.GetOptionInt(wxIMAGE_OPTION_CUR_HOTSPOT_Y);
```

See also

SaveFile()

virtual bool wxImage::LoadFile (const wxString & *name*, wxBitmapType *type* = wxBITMAP_TYPE_ANY, int *index* = -1) [virtual]

Loads an image from a file.

If no handler type is provided, the library will try to autodetect the format.

Parameters

name	Name of the file from which to load the image.
type	See the description in the LoadFile(wxInputStream&, wxBitmapType, int) overload.
index	See the description in the LoadFile(wxInputStream&, wxBitmapType, int) overload.

virtual bool wxImage::LoadFile (const wxString & *name*, const wxString & *mimetype*, int *index* = -1) [virtual]

Loads an image from a file.

If no handler type is provided, the library will try to autodetect the format.

Parameters

name	Name of the file from which to load the image.
mimetype	MIME type string (for example 'image/jpeg')
index	See the description in the LoadFile(wxInputStream&, wxBitmapType, int) overload.

virtual bool wxImage::LoadFile (wxInputStream & *stream*, const wxString & *mimetype*, int *index* = -1) [virtual]

Loads an image from an input stream.

Parameters

stream	Opened input stream from which to load the image. Currently, the stream must support seeking.
mimetype	MIME type string (for example 'image/jpeg')
index	See the description in the LoadFile(wxInputStream&, wxBitmapType, int) overload.

wxImage wxImage::Mirror (bool *horizontally* = `true`) const

Returns a mirrored copy of the image.

The parameter *horizontally* indicates the orientation.

wxImage& wxImage::operator= (const wxImage & *image*)

Assignment operator, using reference counting.

Parameters

image	Image to assign.

Returns

Returns 'this' object.

void wxImage::Paste (const wxImage & *image,* int *x,* int *y*)

Copy the data of the given *image* to the specified position in this image.

static bool wxImage::RemoveHandler (const wxString & *name*) `[static]`

Finds the handler with the given name, and removes it.

The handler is also deleted.

Parameters

name	The handler name.

Returns

true if the handler was found and removed, false otherwise.

See also

wxImageHandler

void wxImage::Replace (unsigned char *r1,* unsigned char *g1,* unsigned char *b1,* unsigned char *r2,* unsigned char *g2,* unsigned char *b2*)

Replaces the colour specified by *r1*,g1,b1 by the colour *r2*,g2,b2.

wxImage& wxImage::Rescale (int *width,* int *height,* wxImageResizeQuality *quality* = wxIMAGE_QUALITY_NORMAL)

Changes the size of the image in-place by scaling it: after a call to this function,the image will have the given width and height.

For a description of the *quality* parameter, see the Scale() function. Returns the (modified) image itself.

See also

Scale()

wxImage& wxImage::Resize (const wxSize & *size,* const wxPoint & *pos,* int *red* = -1, int *green* = -1, int *blue* = -1)

Changes the size of the image in-place without scaling it by adding either a border with the given colour or cropping as necessary.

The image is pasted into a new image with the given *size* and background colour at the position *pos* relative to the upper left of the new image.

If *red* = green = blue = -1 then use either the current mask colour if set or find, use, and set a suitable mask colour for any newly exposed areas.

Returns

The (modified) image itself.

See also

Size()

static wxImage::HSVValue wxImage::RGBtoHSV (const wxImage::RGBValue & *rgb*) [static]

Converts a color in RGB color space to HSV color space.

wxImage wxImage::Rotate (double *angle,* const wxPoint & *rotationCentre,* bool *interpolating* = true, wxPoint * *offsetAfterRotation* = NULL) const

Rotates the image about the given point, by *angle* radians.

Passing true to *interpolating* results in better image quality, but is slower.

If the image has a mask, then the mask colour is used for the uncovered pixels in the rotated image background. Else, black (rgb 0, 0, 0) will be used.

Returns the rotated image, leaving this image intact.

wxImage wxImage::Rotate180 ( ) const

Returns a copy of the image rotated by 180 degrees.

Since

2.9.2

wxImage wxImage::Rotate90 (bool *clockwise* = true) const

Returns a copy of the image rotated 90 degrees in the direction indicated by *clockwise*.

void wxImage::RotateHue (double *angle*)

Rotates the hue of each pixel in the image by *angle*, which is a double in the range of -1.0 to +1.0, where -1.0 corresponds to -360 degrees and +1.0 corresponds to +360 degrees.

virtual bool wxImage::SaveFile (wxOutputStream & *stream*, const wxString & *mimetype*) const `[virtual]`

Saves an image in the given stream.

Parameters

stream	Opened output stream to save the image to.
mimetype	MIME type.

Returns

true if the operation succeeded, false otherwise.

Remarks

Depending on how wxWidgets has been configured, not all formats may be available.

Note

You can use SetOption() to set the hotspot when saving an image into a cursor file (default hotspot is in the centre of the image):

```
image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_X, hotspotX);
image.SetOption(wxIMAGE_OPTION_CUR_HOTSPOT_Y, hotspotY);
```

See also

LoadFile()

virtual bool wxImage::SaveFile (const wxString & *name*, wxBitmapType *type*) const `[virtual]`

Saves an image in the named file.

Parameters

name	Name of the file to save the image to.
type	Currently these types can be used: • wxBITMAP_TYPE_BMP: Save a BMP image file. • wxBITMAP_TYPE_JPEG: Save a JPEG image file. • wxBITMAP_TYPE_PNG: Save a PNG image file. • wxBITMAP_TYPE_PCX: Save a PCX image file (tries to save as 8-bit if possible, falls back to 24-bit otherwise). • wxBITMAP_TYPE_PNM: Save a PNM image file (as raw RGB always). • wxBITMAP_TYPE_TIFF: Save a TIFF image file. • wxBITMAP_TYPE_XPM: Save a XPM image file. • wxBITMAP_TYPE_ICO: Save a Windows icon file (ICO). The size may be up to 255 wide by 127 high. A single image is saved in 8 colors at the size supplied. • wxBITMAP_TYPE_CUR: Save a Windows cursor file (CUR).

virtual bool wxImage::SaveFile (const wxString & *name*, const wxString & *mimetype*) const `[virtual]`

Saves an image in the named file.

Parameters

name	Name of the file to save the image to.
mimetype	MIME type.

virtual bool wxImage::SaveFile (const wxString & *name*) const `[virtual]`

Saves an image in the named file.

File type is determined from the extension of the file name. Note that this function may fail if the extension is not recognized! You can use one of the forms above to save images to files with non-standard extensions.

Parameters

name	Name of the file to save the image to.

virtual bool wxImage::SaveFile (wxOutputStream & *stream*, wxBitmapType *type*) const `[virtual]`

Saves an image in the given stream.

Parameters

stream	Opened output stream to save the image to.
type	MIME type.

wxImage wxImage::Scale (int *width*, int *height*, wxImageResizeQuality *quality* = wxIMAGE_QUALITY_NORMAL) const

Returns a scaled version of the image.

This is also useful for scaling bitmaps in general as the only other way to scale bitmaps is to blit a wxMemoryDC into another wxMemoryDC.

The parameter *quality* determines what method to use for resampling the image, see wxImageResizeQuality documentation.

It should be noted that although using `wxIMAGE_QUALITY_HIGH` produces much nicer looking results it is a slower method. Downsampling will use the box averaging method which seems to operate very fast. If you are upsampling larger images using this method you will most likely notice that it is a bit slower and in extreme cases it will be quite substantially slower as the bicubic algorithm has to process a lot of data.

It should also be noted that the high quality scaling may not work as expected when using a single mask colour for transparency, as the scaling will blur the image and will therefore remove the mask partially. Using the alpha channel will work.

Example:

```
// get the bitmap from somewhere
wxBitmap bmp = ...;

// rescale it to have size of 32*32
if ( bmp.GetWidth() != 32 || bmp.GetHeight() != 32 )
{
    wxImage image = bmp.ConvertToImage();
    bmp = wxBitmap(image.Scale(32, 32));

    // another possibility:
    image.Rescale(32, 32);
    bmp = image;
}
```

See also

Rescale()

void wxImage::SetAlpha (unsigned char * *alpha* = NULL, bool *static_data* = false)

This function is similar to SetData() and has similar restrictions.

The pointer passed to it may however be NULL in which case the function will allocate the alpha array internally – this is useful to add alpha channel data to an image which doesn't have any.

If the pointer is not NULL, it must have one byte for each image pixel and be allocated with malloc(). wxImage takes ownership of the pointer and will free it unless *static_data* parameter is set to true – in this case the caller should do it.

void wxImage::SetAlpha (int *x*, int *y*, unsigned char *alpha*)

Sets the alpha value for the given pixel.

This function should only be called if the image has alpha channel data, use HasAlpha() to check for this.

void wxImage::SetData (unsigned char * *data*, bool *static_data* = false)

Sets the image data without performing checks.

The data given must have the size (width*height*3) or results will be unexpected. Don't use this method if you aren't sure you know what you are doing.

The data must have been allocated with `malloc()`, **NOT** with `operator new`.

If *static_data* is false, after this call the pointer to the data is owned by the wxImage object, that will be responsible for deleting it. Do not pass to this function a pointer obtained through GetData().

void wxImage::SetData (unsigned char * *data*, int *new_width*, int *new_height*, bool *static_data* = false)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxImage::SetMask (bool *hasMask* = true)

Specifies whether there is a mask or not.

The area of the mask is determined by the current mask colour.

void wxImage::SetMaskColour (unsigned char *red*, unsigned char *green*, unsigned char *blue*)

Sets the mask colour for this image (and tells the image to use the mask).

bool wxImage::SetMaskFromImage (const wxImage & *mask*, unsigned char *mr*, unsigned char *mg*, unsigned char *mb*)

Sets image's mask so that the pixels that have RGB value of mr,mg,mb in mask will be masked in the image.

This is done by first finding an unused colour in the image, setting this colour as the mask colour and then using this colour to draw all pixels in the image who corresponding pixel in mask has given RGB value.

The parameter *mask* is the mask image to extract mask shape from. It must have the same dimensions as the image.

The parameters *mr*, *mg*, *mb* are the RGB values of the pixels in mask that will be used to create the mask.

Returns

Returns false if mask does not have same dimensions as the image or if there is no unused colour left. Returns true if the mask was successfully applied.

Note

Note that this method involves computing the histogram, which is a computationally intensive operation.

void wxImage::SetOption (const wxString & *name,* const wxString & *value*)

Sets a user-defined option.

The function is case-insensitive to *name*.

For example, when saving as a JPEG file, the option **quality** is used, which is a number between 0 and 100 (0 is terrible, 100 is very good).

The lists of the currently supported options are in GetOption() and GetOptionInt() function docs.

See also

GetOption(), GetOptionInt(), HasOption()

void wxImage::SetOption (const wxString & *name,* int *value*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxImage::SetPalette (const wxPalette & *palette*)

Associates a palette with the image.

The palette may be used when converting wxImage to wxBitmap (MSW only at present) or in file save operations (none as yet).

void wxImage::SetRGB (int *x,* int *y,* unsigned char *r,* unsigned char *g,* unsigned char *b*)

Set the color of the pixel at the given x and y coordinate.

void wxImage::SetRGB (const wxRect & *rect,* unsigned char *red,* unsigned char *green,* unsigned char *blue*)

Sets the colour of the pixels within the given rectangle.

This routine performs bounds-checks for the coordinate so it can be considered a safe way to manipulate the data.

void wxImage::SetType (wxBitmapType *type*)

Set the type of image returned by GetType().

This method is mostly used internally by the library but can also be called from the user code if the image was created from data in the given bitmap format without using LoadFile() (which would set the type correctly automatically).

Notice that the image must be created before this function is called.

Since

2.9.0

Parameters

type	One of bitmap type constants, `wxBITMAP_TYPE_INVALID` is a valid value for it and can be used to reset the bitmap type to default but `wxBITMAP_TYPE_MAX` is not allowed here.

wxImage wxImage::Size (const wxSize & *size*, const wxPoint & *pos*, int *red* = `-1`, int *green* = `-1`, int *blue* = `-1`) const

Returns a resized version of this image without scaling it by adding either a border with the given colour or cropping as necessary.

The image is pasted into a new image with the given *size* and background colour at the position *pos* relative to the upper left of the new image.

If *red* = green = blue = -1 then the areas of the larger image not covered by this image are made transparent by filling them with the image mask colour (which will be allocated automatically if it isn't currently set).

Otherwise, the areas will be filled with the colour with the specified RGB components.

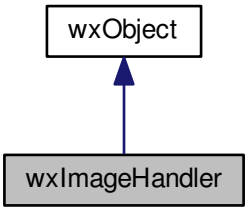
See also

Resize()

21.399 wxImageHandler Class Reference

`#include <wx/image.h>`

Inheritance diagram for wxImageHandler:



21.399.1 Detailed Description

This is the base class for implementing image file loading/saving, and image creation from data.

It is used within wxImage and is not normally seen by the application.

If you wish to extend the capabilities of wxImage, derive a class from wxImageHandler and add the handler using wxImage::AddHandler in your application initialization.

Note that all wxImageHandlers provided by wxWidgets are part of the wxCore library. For details about the default handlers, please see the section Available image handlers in the wxImage class documentation.

21.399.2 Note (Legal Issue)

This software is based in part on the work of the Independent JPEG Group. (Applies when wxWidgets is linked with JPEG support. wxJPEGHandler uses libjpeg created by IJG.)

Predefined objects/pointers: wxNullImage

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxImage, wxInitAllImageHandlers()

Public Member Functions

- wxImageHandler ()

  Default constructor.
- virtual ~wxImageHandler ()

  Destroys the wxImageHandler object.
- bool CanRead (wxInputStream &stream)

  Returns true if this handler supports the image format contained in the given stream.
- bool CanRead (const wxString &filename)

  Returns true if this handler supports the image format contained in the file with the given name.
- const wxString & GetExtension () const

  Gets the preferred file extension associated with this handler.
- const wxArrayString & GetAltExtensions () const

  Returns the other file extensions associated with this handler.
- virtual int GetImageCount (wxInputStream &stream)

  If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.
- const wxString & GetMimeType () const

  Gets the MIME type associated with this handler.
- const wxString & GetName () const

  Gets the name of this handler.
- wxBitmapType GetType () const

  Gets the image type associated with this handler.
- virtual bool LoadFile (wxImage *image, wxInputStream &stream, bool verbose=true, int index=-1)

  Loads a image from a stream, putting the resulting data into image.
- virtual bool SaveFile (wxImage *image, wxOutputStream &stream, bool verbose=true)

  Saves a image in the output stream.
- void SetExtension (const wxString &extension)

  Sets the preferred file extension associated with this handler.
- void SetAltExtensions (const wxArrayString &extensions)

  Sets the alternative file extensions associated with this handler.
- void SetMimeType (const wxString &mimetype)

  Sets the handler MIME type.
- void SetName (const wxString &name)

  Sets the handler name.

Static Public Member Functions

- static wxVersionInfo GetLibraryVersionInfo ()

  Retrieve the version information about the image library used by this handler.

Protected Member Functions

- virtual int DoGetImageCount (wxInputStream &stream)

  Called to get the number of images available in a multi-image file type, if supported.
- virtual bool DoCanRead (wxInputStream &stream)=0

  Called to test if this handler can read an image from the given stream.

Additional Inherited Members

21.399.3 Constructor & Destructor Documentation

wxImageHandler::wxImageHandler ( )

Default constructor.

In your own default constructor, initialise the members m_name, m_extension and m_type.

virtual wxImageHandler::~wxImageHandler ( ) [virtual]

Destroys the wxImageHandler object.

21.399.4 Member Function Documentation

bool wxImageHandler::CanRead (wxInputStream & *stream*)

Returns true if this handler supports the image format contained in the given stream.

This function doesn't modify the current stream position (because it restores the original position before returning; this however requires the stream to be seekable; see wxStreamBase::IsSeekable).

bool wxImageHandler::CanRead (const wxString & *filename*)

Returns true if this handler supports the image format contained in the file with the given name.

This function doesn't modify the current stream position (because it restores the original position before returning; this however requires the stream to be seekable; see wxStreamBase::IsSeekable).

virtual bool wxImageHandler::DoCanRead (wxInputStream & *stream*) [protected], [pure virtual]

Called to test if this handler can read an image from the given stream.

NOTE: this function is allowed to change the current stream position since CallDoCanRead() will take care of restoring it later

virtual int wxImageHandler::DoGetImageCount (wxInputStream & *stream*) [protected], [virtual]

Called to get the number of images available in a multi-image file type, if supported.

NOTE: this function is allowed to change the current stream position since GetImageCount() will take care of restoring it later

const wxArrayString& wxImageHandler::GetAltExtensions ( ) const

Returns the other file extensions associated with this handler.

The preferred extension for this handler is returned by GetExtension().

Since

2.9.0

const wxString& wxImageHandler::GetExtension ( ) const

Gets the preferred file extension associated with this handler.

See also

GetAltExtensions()

virtual int wxImageHandler::GetImageCount (wxInputStream & *stream*) [virtual]

If the image file contains more than one image and the image handler is capable of retrieving these individually, this function will return the number of available images.

Parameters

stream	Opened input stream for reading image data. This function doesn't modify the current stream position (because it restores the original position before returning; this however requires the stream to be seekable; see wxStreamBase::IsSeekable).

Returns

Number of available images. For most image handlers, this is 1 (exceptions are TIFF and ICO formats as well as animated GIFs for which this function returns the number of frames in the animation).

static wxVersionInfo wxImageHandler::GetLibraryVersionInfo ( ) [static]

Retrieve the version information about the image library used by this handler.

This method is not present in wxImageHandler class itself but is present in a few of the classes deriving from it, currently wxJPEGHandler, wxPNGHandler and wxTIFFHandler. It returns the information about the version of the image library being used for the corresponding handler implementation.

Since

2.9.2

const wxString& wxImageHandler::GetMimeType ( ) const

Gets the MIME type associated with this handler.

const wxString& wxImageHandler::GetName ( ) const

Gets the name of this handler.

wxBitmapType wxImageHandler::GetType ( ) const

Gets the image type associated with this handler.

virtual bool wxImageHandler::LoadFile (wxImage * *image*, wxInputStream & *stream*, bool *verbose* = `true`, int *index* = `-1`) [virtual]

Loads a image from a stream, putting the resulting data into *image*.

If the image file contains more than one image and the image handler is capable of retrieving these individually, *index* indicates which image to read from the stream.

Parameters

image	The image object which is to be affected by this operation.
stream	Opened input stream for reading image data.
verbose	If set to true, errors reported by the image handler will produce wxLogMessages.
index	The index of the image in the file (starting from zero).

Returns

true if the operation succeeded, false otherwise.

See also

wxImage::LoadFile, wxImage::SaveFile, SaveFile()

virtual bool wxImageHandler::SaveFile (wxImage * *image*, wxOutputStream & *stream*, bool *verbose* = `true`) [virtual]

Saves a image in the output stream.

Parameters

image	The image object which is to be affected by this operation.
stream	Opened output stream for writing the data.
verbose	If set to true, errors reported by the image handler will produce wxLogMessages.

Returns

true if the operation succeeded, false otherwise.

See also

wxImage::LoadFile, wxImage::SaveFile, LoadFile()

void wxImageHandler::SetAltExtensions (const wxArrayString & *extensions*)

Sets the alternative file extensions associated with this handler.

Parameters

extensions	Array of file extensions.

See also

SetExtension()

Since

2.9.0

void wxImageHandler::SetExtension (const wxString & *extension*)

Sets the preferred file extension associated with this handler.

Parameters

extension	File extension without leading dot.

See also

SetAltExtensions()

void wxImageHandler::SetMimeType (const wxString & *mimetype*)

Sets the handler MIME type.

Parameters

mimetype	Handler MIME type.

void wxImageHandler::SetName (const wxString & *name*)

Sets the handler name.

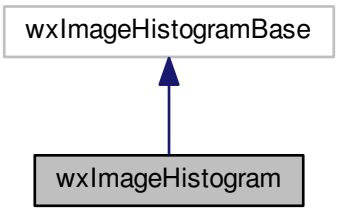
Parameters

name	Handler name.

21.400 wxImageHistogram Class Reference

`#include <wx/image.h>`

Inheritance diagram for wxImageHistogram:



Public Member Functions

- wxImageHistogram ()
- bool FindFirstUnusedColour (unsigned char ∗r, unsigned char ∗g, unsigned char ∗b, unsigned char startR=1, unsigned char startG=0, unsigned char startB=0) const

Static Public Member Functions

- static unsigned long MakeKey (unsigned char r, unsigned char g, unsigned char b)

21.400.1 Constructor & Destructor Documentation

wxImageHistogram::wxImageHistogram ( )

21.400.2 Member Function Documentation

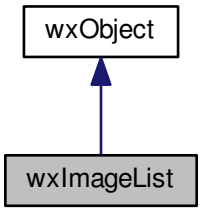
bool wxImageHistogram::FindFirstUnusedColour (unsigned char ∗ *r,* unsigned char ∗ *g,* unsigned char ∗ *b,* unsigned char *startR* = `1`, unsigned char *startG* = `0`, unsigned char *startB* = `0`) const

static unsigned long wxImageHistogram::MakeKey (unsigned char *r,* unsigned char *g,* unsigned char *b*) `[static]`

21.401 wxImageList Class Reference

```
#include <wx/imaglist.h>
```

Inheritance diagram for wxImageList:



21.401.1 Detailed Description

A wxImageList contains a list of images, which are stored in an unspecified form.

Images can have masks for transparent drawing, and can be made from a variety of sources including bitmaps and icons.

wxImageList is used principally in conjunction with wxTreeCtrl and wxListCtrl classes.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxTreeCtrl, wxListCtrl

Public Member Functions

- wxImageList ()

  Default ctor.
- wxImageList (int width, int height, bool mask=true, int initialCount=1)

  Constructor specifying the image size, whether image masks should be created, and the initial size of the list.
- int Add (const wxBitmap &bitmap, const wxBitmap &mask=wxNullBitmap)

  Adds a new image or images using a bitmap and optional mask bitmap.
- int Add (const wxBitmap &bitmap, const wxColour &maskColour)

  Adds a new image or images using a bitmap and mask colour.
- int Add (const wxIcon &icon)

  Adds a new image using an icon.
- bool Create (int width, int height, bool mask=true, int initialCount=1)

  Initializes the list.
- virtual bool Draw (int index, wxDC &dc, int x, int y, int flags=wxIMAGELIST_DRAW_NORMAL, bool solid↩ Background=false)

  Draws a specified image onto a device context.
- wxBitmap GetBitmap (int index) const

  Returns the bitmap corresponding to the given index.

- wxIcon GetIcon (int index) const

  Returns the icon corresponding to the given index.
- virtual int GetImageCount () const

  Returns the number of images in the list.
- virtual bool GetSize (int index, int &width, int &height) const

  Retrieves the size of the images in the list.
- bool Remove (int index)

  Removes the image at the given position.
- bool RemoveAll ()

  Removes all the images in the list.
- bool Replace (int index, const wxBitmap &bitmap, const wxBitmap &mask=wxNullBitmap)

  Replaces the existing image with the new image.
- bool Replace (int index, const wxIcon &icon)

  Replaces the existing image with the new image.

Additional Inherited Members

21.401.2 Constructor & Destructor Documentation

wxImageList::wxImageList ( )

Default ctor.

wxImageList::wxImageList (int *width*, int *height*, bool *mask* = `true`, int *initialCount* = `1`)

Constructor specifying the image size, whether image masks should be created, and the initial size of the list.

Parameters

width	Width of the images in the list.
height	Height of the images in the list.
mask	true if masks should be created for all images.
initialCount	The initial size of the list.

See also

Create()

21.401.3 Member Function Documentation

int wxImageList::Add (const wxBitmap & *bitmap*, const wxBitmap & *mask* = wxNullBitmap)

Adds a new image or images using a bitmap and optional mask bitmap.

Parameters

bitmap	Bitmap representing the opaque areas of the image.
mask	Monochrome mask bitmap, representing the transparent areas of the image.

Returns

The new zero-based image index.

Remarks

The original bitmap or icon is not affected by the Add() operation, and can be deleted afterwards. If the bitmap is wider than the images in the list, then the bitmap will automatically be split into smaller images, each matching the dimensions of the image list. This does not apply when adding icons.

int wxImageList::Add (const wxBitmap & *bitmap*, const wxColour & *maskColour*)

Adds a new image or images using a bitmap and mask colour.

Parameters

bitmap	Bitmap representing the opaque areas of the image.
maskColour	Colour indicating which parts of the image are transparent.

Returns

The new zero-based image index.

Remarks

The original bitmap or icon is not affected by the Add() operation, and can be deleted afterwards. If the bitmap is wider than the images in the list, then the bitmap will automatically be split into smaller images, each matching the dimensions of the image list. This does not apply when adding icons.

int wxImageList::Add (const wxIcon & *icon*)

Adds a new image using an icon.

Parameters

icon	Icon to use as the image.

Returns

The new zero-based image index.

Remarks

The original bitmap or icon is not affected by the Add() operation, and can be deleted afterwards. If the bitmap is wider than the images in the list, then the bitmap will automatically be split into smaller images, each matching the dimensions of the image list. This does not apply when adding icons.

Availability: only available for the wxMSW, wxOSX ports.

bool wxImageList::Create (int *width*, int *height*, bool *mask* = `true`, int *initialCount* = `1`)

Initializes the list.

See wxImageList() for details.

virtual bool wxImageList::Draw (int *index*, wxDC & *dc*, int *x*, int *y*, int *flags* = wxIMAGELIST_DRAW_NORMAL, bool *solidBackground* = `false`) `[virtual]`

Draws a specified image onto a device context.

Parameters

index	Image index, starting from zero.
dc	Device context to draw on.
x	X position on the device context.
y	Y position on the device context.
flags	How to draw the image. A bitlist of a selection of the following: • wxIMAGELIST_DRAW_NORMAL: Draw the image normally. • wxIMAGELIST_DRAW_TRANSPARENT: Draw the image with transparency. • wxIMAGELIST_DRAW_SELECTED: Draw the image in selected state. • wxIMAGELIST_DRAW_FOCUSED: Draw the image in a focused state.
solidBackground	For optimisation - drawing can be faster if the function is told that the background is solid.

wxBitmap wxImageList::GetBitmap (int *index*) const

Returns the bitmap corresponding to the given index.

wxIcon wxImageList::GetIcon (int *index*) const

Returns the icon corresponding to the given index.

virtual int wxImageList::GetImageCount ( ) const `[virtual]`

Returns the number of images in the list.

virtual bool wxImageList::GetSize (int *index,* int & *width,* int & *height*) const `[virtual]`

Retrieves the size of the images in the list.

Currently, the *index* parameter is ignored as all images in the list have the same size.

Parameters

index	currently unused, should be 0
width	receives the width of the images in the list
height	receives the height of the images in the list

Returns

true if the function succeeded, false if it failed (for example, if the image list was not yet initialized).

bool wxImageList::Remove (int *index*)

Removes the image at the given position.

bool wxImageList::RemoveAll ( )

Removes all the images in the list.

bool wxImageList::Replace (int *index,* const wxBitmap & *bitmap,* const wxBitmap & *mask* = wxNullBitmap)

Replaces the existing image with the new image.

Windows only.

Parameters

index	The index of the bitmap to be replaced.
bitmap	Bitmap representing the opaque areas of the image.
mask	Monochrome mask bitmap, representing the transparent areas of the image.

Returns

true if the replacement was successful, false otherwise.

Remarks

The original bitmap or icon is not affected by the Replace() operation, and can be deleted afterwards.

bool wxImageList::Replace (int *index,* const wxIcon & *icon*)

Replaces the existing image with the new image.

Parameters

index	The index of the bitmap to be replaced.
icon	Icon to use as the image.

Returns

true if the replacement was successful, false otherwise.

Remarks

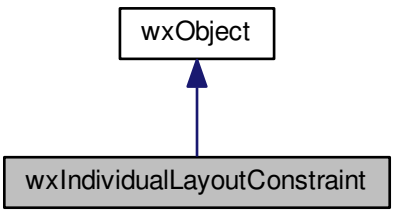
The original bitmap or icon is not affected by the Replace() operation, and can be deleted afterwards.

Availability: only available for the wxMSW, wxOSX ports.

21.402 wxIndividualLayoutConstraint Class Reference

`#include <wx/layout.h>`

Inheritance diagram for wxIndividualLayoutConstraint:



Public Member Functions

- wxIndividualLayoutConstraint ()
- virtual ~wxIndividualLayoutConstraint ()
- void Set (wxRelationship rel, wxWindow *otherW, wxEdge otherE, int val=0, int margin=wxLAYOUT_DEFAULT_MARGIN)
- void LeftOf (wxWindow *sibling, int margin=wxLAYOUT_DEFAULT_MARGIN)
- void RightOf (wxWindow *sibling, int margin=wxLAYOUT_DEFAULT_MARGIN)
- void Above (wxWindow *sibling, int margin=wxLAYOUT_DEFAULT_MARGIN)
- void Below (wxWindow *sibling, int margin=wxLAYOUT_DEFAULT_MARGIN)
- void SameAs (wxWindow *otherW, wxEdge edge, int margin=wxLAYOUT_DEFAULT_MARGIN)
- void PercentOf (wxWindow *otherW, wxEdge wh, int per)
- void Absolute (int val)
- void Unconstrained ()
- void AsIs ()
- wxWindow * GetOtherWindow ()
- wxEdge GetMyEdge () const
- void SetEdge (wxEdge which)
- void SetValue (int v)
- int GetMargin ()
- void SetMargin (int m)
- int GetValue () const
- int GetPercent () const
- int GetOtherEdge () const
- bool GetDone () const
- void SetDone (bool d)
- wxRelationship GetRelationship ()
- void SetRelationship (wxRelationship r)
- bool ResetIfWin (wxWindow *otherW)
- bool SatisfyConstraint (wxLayoutConstraints *constraints, wxWindow *win)
- int GetEdge (wxEdge which, wxWindow *thisWin, wxWindow *other) const

Additional Inherited Members

21.402.1 Constructor & Destructor Documentation

wxIndividualLayoutConstraint::wxIndividualLayoutConstraint ( )

virtual wxIndividualLayoutConstraint::~wxIndividualLayoutConstraint ( ) `[virtual]`

21.402.2 Member Function Documentation

void wxIndividualLayoutConstraint::Above (wxWindow * *sibling,* int *margin* = wxLAYOUT_DEFAULT_MARGIN)

void wxIndividualLayoutConstraint::Absolute (int *val*)

void wxIndividualLayoutConstraint::AsIs ( )

void wxIndividualLayoutConstraint::Below (wxWindow * *sibling,* int *margin* = wxLAYOUT_DEFAULT_MARGIN)

bool wxIndividualLayoutConstraint::GetDone ( ) const

int wxIndividualLayoutConstraint::GetEdge (wxEdge *which,* wxWindow * *thisWin,* wxWindow * *other*) const

int wxIndividualLayoutConstraint::GetMargin ( )

wxEdge wxIndividualLayoutConstraint::GetMyEdge ( ) const

int wxIndividualLayoutConstraint::GetOtherEdge ( ) const

wxWindow* wxIndividualLayoutConstraint::GetOtherWindow ( )

int wxIndividualLayoutConstraint::GetPercent ( ) const

wxRelationship wxIndividualLayoutConstraint::GetRelationship ( )

int wxIndividualLayoutConstraint::GetValue ( ) const

void wxIndividualLayoutConstraint::LeftOf (**wxWindow** * *sibling,* int *margin =* **wxLAYOUT_DEFAULT_MARGIN**)

void wxIndividualLayoutConstraint::PercentOf (**wxWindow** * *otherW,* **wxEdge** *wh,* int *per*)

bool wxIndividualLayoutConstraint::ResetIfWin (**wxWindow** * *otherW*)

void wxIndividualLayoutConstraint::RightOf (**wxWindow** * *sibling,* int *margin =* **wxLAYOUT_DEFAULT_MARGIN**)

void wxIndividualLayoutConstraint::SameAs (**wxWindow** * *otherW,* **wxEdge** *edge,* int *margin =* **wxLAYOUT_DEFAULT_MARGIN**)

bool wxIndividualLayoutConstraint::SatisfyConstraint (**wxLayoutConstraints** * *constraints,* **wxWindow** * *win*)

void wxIndividualLayoutConstraint::Set (**wxRelationship** *rel,* **wxWindow** * *otherW,* **wxEdge** *otherE,* int *val =* 0, int *margin =* **wxLAYOUT_DEFAULT_MARGIN**)

void wxIndividualLayoutConstraint::SetDone (bool *d*)

void wxIndividualLayoutConstraint::SetEdge (**wxEdge** *which*)

void wxIndividualLayoutConstraint::SetMargin (int *m*)

void wxIndividualLayoutConstraint::SetRelationship (**wxRelationship** *r*)

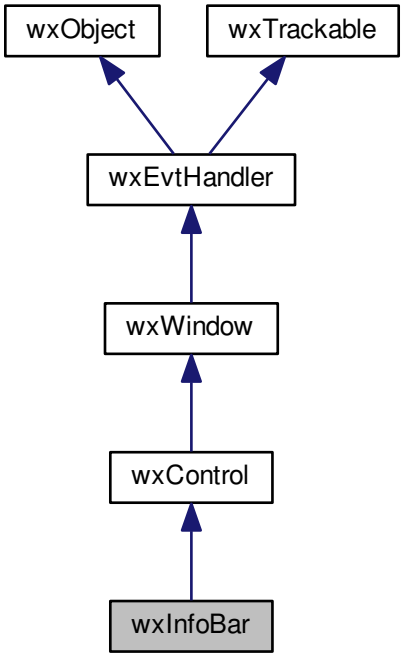
void wxIndividualLayoutConstraint::SetValue (int *v*)

void wxIndividualLayoutConstraint::Unconstrained ( )

21.403 wxInfoBar Class Reference

`#include <wx/infobar.h>`

Inheritance diagram for wxInfoBar:



21.403.1 Detailed Description

An info bar is a transient window shown at top or bottom of its parent window to display non-critical information to the user.

This class provides another way to show messages to the user, intermediate between message boxes and status bar messages. The message boxes are modal and thus interrupt the users work flow and should be used sparingly for this reason. However status bar messages are often too easy not to notice at all. An info bar provides a way to present the messages which has a much higher chance to be noticed by the user but without being annoying.

Info bar may show an icon (on the left), text message and, optionally, buttons allowing the user to react to the information presented. It always has a close button at the right allowing the user to dismiss it so it isn't necessary to provide a button just to close it.

wxInfoBar calls its parent wxWindow::Layout() method and assumes that it will change the parent layout appropriately depending on whether the info bar itself is shown or hidden. Usually this is achieved by simply using a sizer for the parent window layout and adding wxInfoBar to this sizer as one of the items. Considering the usual placement of the info bars, normally this sizer should be a vertical wxBoxSizer and the bar its first or last element so the simplest possible example of using this class would be:

```
class MyFrame : public wxFrame
{
    ...

    wxInfoBar *m_infoBar;
};

MyFrame::MyFrame()
{
    ...
    m_infoBar = new wxInfoBar(this);
```

```
    wxSizer *sizer = new wxBoxSizer(wxVERTICAL);
    sizer->Add(m_infoBar, wxSizerFlags().Expand());
    ... add other frame controls to the sizer ...
    SetSizer(sizer);
}

void MyFrame::SomeMethod()
{
    m_infoBar->ShowMessage("Something happened", wxICON_INFORMATION);
}
```

See the dialogs sample for more sophisticated examples.

Currently this class is implemented generically (i.e. in the same platform-independent way for all ports) and also natively in wxGTK but the native implementation requires a recent – as of this writing – GTK+ 2.18 version.

Library: wxCore

Category: Miscellaneous Windows

See also

wxStatusBar, wxMessageDialog

Since

2.9.1

Public Member Functions

- wxInfoBar ()

  Default constructor.
- wxInfoBar (wxWindow *parent, wxWindowID winid=wxID_ANY)

  Constructor creating the info bar window.
- bool Create (wxWindow *parent, wxWindowID winid=wxID_ANY)

  Create the info bar window.
- void AddButton (wxWindowID btnid, const wxString &label=wxString())

  Add a button to be shown in the info bar.
- virtual void Dismiss ()

  Hide the info bar window.
- void RemoveButton (wxWindowID btnid)

  Remove a button previously added by AddButton().
- void ShowMessage (const wxString &msg, int flags=wxICON_NONE)

  Show a message in the bar.
- virtual size_t GetButtonCount () const

  Returns the number of currently shown buttons.
- virtual wxWindowID GetButtonId (size_t idx) const

  Returns the ID of the button at the given position.
- virtual bool HasButtonId (wxWindowID btnid) const

  Returns whether a button with the given ID is currently shown.

Generic version customization methods.

All these methods exist in the generic version of the class only.

The generic version uses wxWindow::ShowWithEffect() function to progressively show it on the platforms which support it (currently only wxMSW). The methods here allow to change the default effect used (or disable it entirely) and change its duration.

- void SetShowHideEffects (wxShowEffect showEffect, wxShowEffect hideEffect)
  Set the effects to use when showing and hiding the bar.
- wxShowEffect GetShowEffect () const
  Return the effect currently used for showing the bar.
- wxShowEffect GetHideEffect () const
  Return the effect currently used for hiding the bar.
- void SetEffectDuration (int duration)
  Set the duration of the animation used when showing or hiding the bar.
- int GetEffectDuration () const
  Return the effect animation duration currently used.
- virtual bool SetFont (const wxFont &font)
  Overridden base class methods changes the font of the text message.

Additional Inherited Members

21.403.2 Constructor & Destructor Documentation

wxInfoBar::wxInfoBar ( )

Default constructor.

Use Create() for the objects created using this constructor.

wxInfoBar::wxInfoBar (wxWindow * *parent*, wxWindowID *winid* = wxID_ANY)

Constructor creating the info bar window.

See also

Create()

21.403.3 Member Function Documentation

void wxInfoBar::AddButton (wxWindowID *btnid*, const wxString & *label* = wxString())

Add a button to be shown in the info bar.

The button added by this method will be shown to the right of the text (in LTR layout), with each successive button being added to the right of the previous one. If any buttons are added to the info bar using this method, the default "Close" button is not shown as it is assumed that the extra buttons already allow the user to close it.

Clicking the button will generate a normal EVT_COMMAND_BUTTON_CLICKED event which can be handled as usual. The default handler in wxInfoBar itself closes the window whenever a button in it is clicked so if you wish the info bar to be hidden when the button is clicked, simply call `event.Skip()` in the button handler to let the base class handler do it (calling Dismiss() explicitly works too, of course). On the other hand, if you don't skip the event, the info bar will remain opened so make sure to do it for at least some buttons to allow the user to close it.

Notice that the generic wxInfoBar implementation handles the button events itself and so they are not propagated to the info bar parent and you need to either inherit from wxInfoBar and handle them in your derived class or use wxEvtHandler::Connect(), as is done in the dialogs sample, to handle the button events in the parent frame.

Parameters

btnid	Id of the button. It will be used in the button message clicking this button will generate.

label	The label of the button. It may only be empty if *btnid* is one of the stock ids in which case the corresponding stock label (see wxGetStockLabel()) will be used.

bool wxInfoBar::Create (wxWindow * *parent*, wxWindowID *winid* = wxID_ANY)

Create the info bar window.

Notice that unlike most of the other wxWindow-derived classes, wxInfoBar is created hidden and is only shown when ShowMessage() is called. This is more convenient as usually the info bar is created to be shown at some later time and not immediately and so creating it hidden avoids the need to call Hide() explicitly from the code using it.

This should be only called if the object was created using its default constructor.

Parameters

parent	A valid parent window pointer.
winid	The id of the info bar window, usually unused as currently no events are generated by this class.

virtual void wxInfoBar::Dismiss ( ) [virtual]

Hide the info bar window.

This method hides the window and lays out the parent window to account for its disappearance (unlike a simple Hide()).

virtual size_t wxInfoBar::GetButtonCount ( ) const [virtual]

Returns the number of currently shown buttons.

This is simply the number of calls to AddButton() minus the number of calls to RemoveButton() so far.

Returns

The number of currently shown buttons, possibly 0.

Since

3.1.0

virtual wxWindowID wxInfoBar::GetButtonId (size_t *idx*) const [virtual]

Returns the ID of the button at the given position.

The positions of the buttons are counted in order of their addition.

Parameters

idx	The position of the button in 0 to GetButtonCount() range.

Returns

The ID of the button at the given position or wxID_NONE if it is out of range (this also results in an assertion failure).

Since

3.1.0

int wxInfoBar::GetEffectDuration ( ) const

Return the effect animation duration currently used.

wxShowEffect wxInfoBar::GetHideEffect ( ) const

Return the effect currently used for hiding the bar.

wxShowEffect wxInfoBar::GetShowEffect ( ) const

Return the effect currently used for showing the bar.

virtual bool wxInfoBar::HasButtonId (wxWindowID *btnid*) const [virtual]

Returns whether a button with the given ID is currently shown.

Parameters

btnid	ID of the button to check for.

Returns

true if the button with this ID is currently shown.

Since

3.1.0

void wxInfoBar::RemoveButton (wxWindowID *btnid*)

Remove a button previously added by AddButton().

Parameters

btnid	Id of the button to remove. If more than one button with the same id is used in the info bar (which is in any case not recommended), the last, i.e. most recently added, button with this id is removed.

void wxInfoBar::SetEffectDuration (int *duration*)

Set the duration of the animation used when showing or hiding the bar.

By default, 500ms duration is used.

Parameters

duration	Duration of the animation, in milliseconds.

virtual bool wxInfoBar::SetFont (const wxFont & *font*) [virtual]

Overridden base class methods changes the font of the text message.

wxInfoBar overrides this method to use the font passed to it for its text message part. By default a larger and bold version of the standard font is used.

This method is generic-only.

Reimplemented from wxWindow.

void wxInfoBar::SetShowHideEffects (wxShowEffect *showEffect*, wxShowEffect *hideEffect*)

Set the effects to use when showing and hiding the bar.

Either or both of the parameters can be set to wxSHOW_EFFECT_NONE to disable using effects entirely.

By default, the info bar uses wxSHOW_EFFECT_SLIDE_TO_BOTTOM effect for showing itself and wxSHOW_E↩FFECT_SLIDE_TO_TOP for hiding if it is the first element of the containing sizer and reverse effects if it's the last one. If it is neither the first nor the last element, no effect is used to avoid the use of an inappropriate one and this function must be called if an effect is desired.

Parameters

showEffect	The effect to use when showing the bar.
hideEffect	The effect to use when hiding the bar.

void wxInfoBar::ShowMessage (const wxString & *msg*, int *flags* = wxICON_NONE)

Show a message in the bar.

If the bar is currently hidden, it will be shown. Otherwise its message will be updated in place.

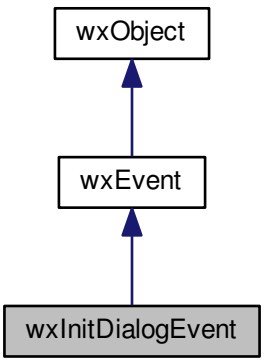
Parameters

msg	The text of the message.
flags	One of wxICON_NONE, wxICON_INFORMATION (default), wxICON_QUESTION, wxICO↩N_WARNING or wxICON_ERROR values. These flags have the same meaning as in wx↩MessageDialog for the generic version, i.e. show (or not, in case of wxICON_NONE) the corresponding icon in the bar but can be interpreted by the native versions. For example, the GTK+ native implementation doesn't show icons at all but uses this parameter to select the appropriate background colour for the notification.

21.404 wxInitDialogEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxInitDialogEvent:



21.404.1 Detailed Description

A wxInitDialogEvent is sent as a dialog or panel is being initialised.

Handlers for this event can transfer data to the window.

The default handler calls wxWindow::TransferDataToWindow.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxInitDialogEvent& event)

Event macros:

- EVT_INIT_DIALOG(func): Process a `wxEVT_INIT_DIALOG` event.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxInitDialogEvent (int id=0)

  Constructor.

Additional Inherited Members

21.404.2 Constructor & Destructor Documentation

wxInitDialogEvent::wxInitDialogEvent (int *id* = `0`)

Constructor.

21.405 wxInitializer Class Reference

```
#include <wx/init.h>
```

21.405.1 Detailed Description

Create an object of this class on the stack to initialize/cleanup the library automatically.

Library: wxBase

Category: Application and Process Management

See also

wxGLContext

Public Member Functions

- wxInitializer (int argc=0, wxChar **argv=NULL)

  Initializes the library.
- bool IsOk () const

  Has the initialization been successful? (explicit test)
- ~wxInitializer ()

  This dtor only does clean up if we initialized the library properly.

21.405.2 Constructor & Destructor Documentation

wxInitializer::wxInitializer (int *argc* = `0`, wxChar ** *argv* = `NULL`)

Initializes the library.

Calls wxInitialize().

wxInitializer::~wxInitializer ( )

This dtor only does clean up if we initialized the library properly.

Calls wxUninitialize().

21.405.3 Member Function Documentation

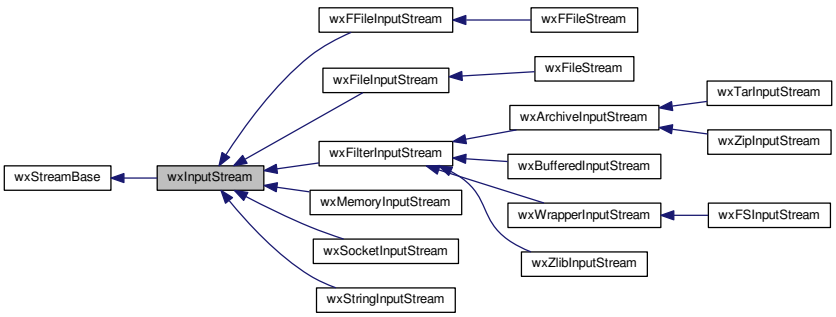
bool wxInitializer::IsOk ( ) const

Has the initialization been successful? (explicit test)

21.406 wxInputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxInputStream:



21.406.1 Detailed Description

wxInputStream is an abstract base class which may not be used directly.

It is the base class of all streams which provide a Read() function, i.e. which can be used to read data from a source (e.g. a file, a socket, etc).

If you want to create your own input stream, you'll need to derive from this class and implement the protected OnSysRead() function only.

Library: wxBase

Category: Streams

Public Member Functions

- wxInputStream ()

  Creates a dummy input stream.
- virtual ~wxInputStream ()

  Destructor.
- virtual bool CanRead () const

  Returns true if some data is available in the stream right now, so that calling Read() wouldn't block.
- virtual bool Eof () const

  Returns true after an attempt has been made to read past the end of the stream.
- int GetC ()

  Returns the first character in the input queue and removes it, blocking until it appears if necessary.
- virtual size_t LastRead () const

  Returns the last number of bytes read.
- virtual char Peek ()

  Returns the first character in the input queue without removing it.
- virtual wxInputStream & Read (void *buffer, size_t size)

  Reads the specified amount of bytes and stores the data in buffer.
- wxInputStream & Read (wxOutputStream &stream_out)

  Reads data from the input queue and stores it in the specified output stream.
- bool ReadAll (void *buffer, size_t size)

  Reads exactly the specified number of bytes into the buffer.
- virtual wxFileOffset SeekI (wxFileOffset pos, wxSeekMode mode=wxFromStart)

  Changes the stream current position.
- virtual wxFileOffset TellI () const

  Returns the current stream position or wxInvalidOffset if it's not available (e.g.
- size_t Ungetch (const void *buffer, size_t size)

  This function is only useful in read mode.
- bool Ungetch (char c)

  This function acts like the previous one except that it takes only one character: it is sometimes shorter to use than the generic function.

Protected Member Functions

- size_t OnSysRead (void *buffer, size_t bufsize)=0

  Internal function.

21.406.2 Constructor & Destructor Documentation

wxInputStream::wxInputStream ( )

Creates a dummy input stream.

virtual wxInputStream::~wxInputStream ( ) [virtual]

Destructor.

21.406.3 Member Function Documentation

virtual bool wxInputStream::CanRead ( ) const [virtual]

Returns true if some data is available in the stream right now, so that calling Read() wouldn't block.

virtual bool wxInputStream::Eof ( ) const [virtual]

Returns true after an attempt has been made to read past the end of the stream.

int wxInputStream::GetC ( )

Returns the first character in the input queue and removes it, blocking until it appears if necessary.

On success returns a value between 0 - 255; on end of file returns `wxEOF`.

virtual size_t wxInputStream::LastRead ( ) const [virtual]

Returns the last number of bytes read.

size_t wxInputStream::OnSysRead (void * *buffer*, size_t *bufsize*) [protected], [pure virtual]

Internal function.

It is called when the stream wants to read data of the specified size *bufsize* and wants it to be placed inside *buffer*.

It should return the size that was actually read or zero if EOF has been reached or an error occurred (in this last case the internal `m_lasterror` variable should be set accordingly as well).

virtual char wxInputStream::Peek ( ) [virtual]

Returns the first character in the input queue without removing it.

virtual wxInputStream& wxInputStream::Read (void * *buffer*, size_t *size*) [virtual]

Reads the specified amount of bytes and stores the data in buffer.

To check if the call was successful you must use LastRead() to check if this call did actually read *size* bytes (if it didn't, GetLastError() should return a meaningful value).

Warning

The buffer absolutely needs to have at least the specified size.

Returns

This function returns a reference on the current object, so the user can test any states of the stream right away.

wxInputStream& wxInputStream::Read (wxOutputStream & *stream_out*)

Reads data from the input queue and stores it in the specified output stream.

The data is read until an error is raised by one of the two streams.

Returns

This function returns a reference on the current object, so the user can test any states of the stream right away.

bool wxInputStream::ReadAll (void * *buffer,* size_t *size*)

Reads exactly the specified number of bytes into the buffer.

Returns true only if the entire amount of data was read, otherwise false is returned and the number of bytes really read can be retrieved using LastRead(), as with Read().

This method uses repeated calls to Read() (which may return after reading less than the requested number of bytes) if necessary.

Warning

The buffer absolutely needs to have at least the specified size.

Since

2.9.5

virtual wxFileOffset wxInputStream::SeekI (wxFileOffset *pos,* wxSeekMode *mode* = wxFromStart) [virtual]

Changes the stream current position.

This operation in general is possible only for seekable streams (see wxStreamBase::IsSeekable()); non-seekable streams support only seeking positive amounts in mode `wxFromCurrent` (this is implemented by reading data and simply discarding it).

Parameters

pos	Offset to seek to.
mode	One of wxFromStart, wxFromEnd, wxFromCurrent.

Returns

The new stream position or wxInvalidOffset on error.

virtual wxFileOffset wxInputStream::TellI ( ) const [virtual]

Returns the current stream position or wxInvalidOffset if it's not available (e.g.

socket streams do not have a size nor a current stream position).

size_t wxInputStream::Ungetch (const void * *buffer*, size_t *size*)

This function is only useful in read mode.

It is the manager of the "Write-Back" buffer. This buffer acts like a temporary buffer where data which has to be read during the next read IO call are put. This is useful when you get a big block of data which you didn't want to read: you can replace them at the top of the input queue by this way.

Be very careful about this call in connection with calling SeekI() on the same stream. Any call to SeekI() will invalidate any previous call to this method (otherwise you could SeekI() to one position, "unread" a few bytes there, SeekI() to another position and data would be either lost or corrupted).

Returns

Returns the amount of bytes saved in the Write-Back buffer.

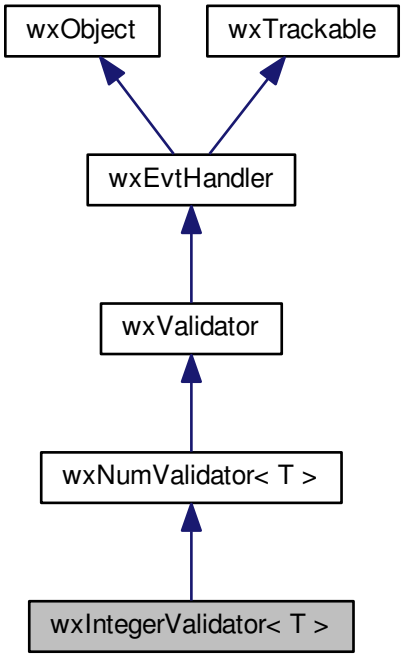
bool wxInputStream::Ungetch (char *c*)

This function acts like the previous one except that it takes only one character: it is sometimes shorter to use than the generic function.

21.407 wxIntegerValidator< T > Class Template Reference

`#include <wx/valnum.h>`

Inheritance diagram for wxIntegerValidator< T >:



21.407.1 Detailed Description

template<typename T>class wxIntegerValidator< T >

Validator for text entries used for integer entry.

This validator can be used with wxTextCtrl or wxComboBox (and potentially any other class implementing wxTextEntry interface) to check that only valid integer values can be entered into them.

This is a template class which can be instantiated for all the integer types (i.e. `short`, `int`, `long` and `long long` if available) as well as their unsigned versions.

By default this validator accepts any integer values in the range appropriate for its type, e.g. `INT_MIN..INT_MAX` for `int` or `0..USHRT_MAX` for `unsigned short`. This range can be restricted further by calling SetMin() and SetMax() or SetRange() methods inherited from the base class.

When the validator displays integers with thousands separators, the character used for the separators (usually "." or ",") depends on the locale set with wxLocale (note that you shouldn't change locale with setlocale() as this can result in a mismatch between the thousands separator used by wxLocale and the one used by the run-time library).

A simple example of using this class:

```
class MyDialog : public wxDialog
{
public:
    MyDialog()
    {
        ...
        // Allow positive integers and display them with thousands
        // separators.
        wxIntegerValidator<unsigned long>
            val(&m_value, wxNUM_VAL_THOUSANDS_SEPARATOR);

        // If the variable were of type "long" and not "unsigned long"
        // we would have needed to call val.SetMin(0) but as it is,
        // this is not needed.

        // Associate it with the text control:
        new wxTextCtrl(this, ..., val);
    }

private:
    unsigned long m_value;
};
```

For more information, please see wxValidator Overview.

Library: wxCore

Category: Validators

See also

wxValidator Overview, wxValidator, wxGenericValidator, wxTextValidator, wxMakeIntegerValidator()

Since

2.9.2

Public Types

- typedef T ValueType

  Type of the values this validator is used with.

Public Member Functions

- wxIntegerValidator (ValueType *value=NULL, int style=wxNUM_VAL_DEFAULT)

  Validator constructor.

Additional Inherited Members

21.407.2 Member Typedef Documentation

template<typename T> typedef T wxIntegerValidator< T >::ValueType

Type of the values this validator is used with.

21.407.3 Constructor & Destructor Documentation

template<typename T> wxIntegerValidator< T >::wxIntegerValidator (ValueType * *value* = NULL, int *style* = wxNUM_VAL_DEFAULT)

Validator constructor.

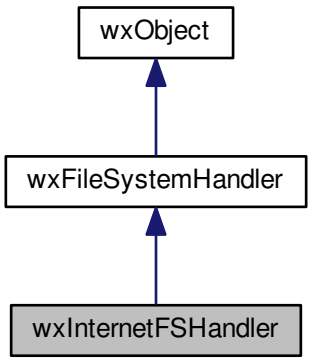
Parameters

value	A pointer to the variable associated with the validator. If non NULL, this variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).
style	A combination of wxNumValidatorStyle enum values with the exception of wxNUM_VAL_NO_TRAILING_ZEROES which can't be used here.

21.408 wxInternetFSHandler Class Reference

`#include <wx/fs_inet.h>`

Inheritance diagram for wxInternetFSHandler:



21.408.1 Detailed Description

A file system handler for accessing files from internet servers.

Public Member Functions

- wxInternetFSHandler ()

Additional Inherited Members

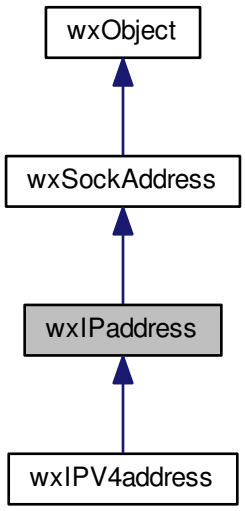
21.408.2 Constructor & Destructor Documentation

wxInternetFSHandler::wxInternetFSHandler ( )

21.409 wxIPaddress Class Reference

`#include <wx/socket.h>`

Inheritance diagram for wxIPaddress:



21.409.1 Detailed Description

wxIPaddress is an abstract base class for all internet protocol address objects.

Currently, only wxIPV4address is implemented. An experimental implementation for IPV6, wxIPV6address, is being developed.

Library: wxNet

Category: Networking

Public Member Functions

- bool AnyAddress ()

  *Internally, this is the same as setting the IP address to **INADDR_ANY**.*
- virtual bool BroadcastAddress ()=0

  *Internally, this is the same as setting the IP address to **INADDR_BROADCAST**.*
- bool Hostname (const wxString &hostname)

  Set the address to hostname, which can be a host name or an IP-style address in a format dependent on implementation.
- wxString Hostname () const

  Returns the hostname which matches the IP address.
- virtual wxString IPAddress () const =0

  Returns a wxString containing the IP address.
- virtual bool IsLocalHost () const =0

  Determines if current address is set to localhost.
- bool LocalHost ()

  Set address to localhost.
- bool Service (const wxString &service)

  Set the port to that corresponding to the specified service.
- bool Service (unsigned short service)

  Set the port to that corresponding to the specified service.
- unsigned short Service () const

  Returns the current service.

Additional Inherited Members

21.409.2 Member Function Documentation

bool wxIPaddress::AnyAddress ( )

Internally, this is the same as setting the IP address to **INADDR_ANY**.

On IPV4 implementations, 0.0.0.0

On IPV6 implementations, ::

Returns

true on success, false if something went wrong.

virtual bool wxIPaddress::BroadcastAddress ( ) [pure virtual]

Internally, this is the same as setting the IP address to **INADDR_BROADCAST**.

On IPV4 implementations, 255.255.255.255

Returns

true on success, false if something went wrong.

bool wxIPaddress::Hostname (const wxString & *hostname*)

Set the address to hostname, which can be a host name or an IP-style address in a format dependent on implementation.

Returns

true on success, false if something goes wrong (invalid hostname or invalid IP address).

wxString wxIPaddress::Hostname ( ) const

Returns the hostname which matches the IP address.

virtual wxString wxIPaddress::IPAddress ( ) const `[pure virtual]`

Returns a wxString containing the IP address.

Implemented in wxIPV4address.

virtual bool wxIPaddress::IsLocalHost ( ) const `[pure virtual]`

Determines if current address is set to localhost.

Returns

true if address is localhost, false if internet address.

bool wxIPaddress::LocalHost ( )

Set address to localhost.

On IPV4 implementations, 127.0.0.1

On IPV6 implementations, ::1

Returns

true on success, false if something went wrong.

bool wxIPaddress::Service (const wxString & *service*)

Set the port to that corresponding to the specified service.

Returns

true on success, false if something goes wrong (invalid *service*).

bool wxIPaddress::Service (unsigned short *service*)

Set the port to that corresponding to the specified service.

Returns

true on success, false if something goes wrong (invalid *service*).

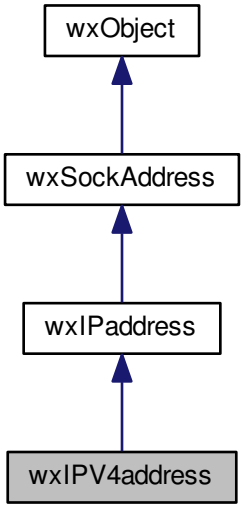
unsigned short wxIPaddress::Service ( ) const

Returns the current service.

21.410 wxIPV4address Class Reference

`#include <wx/socket.h>`

Inheritance diagram for wxIPV4address:



21.410.1 Detailed Description

A class for working with IPv4 network addresses.

Library: wxNet

Category: Networking

Public Member Functions

- bool AnyAddress ()

  Set address to any of the addresses of the current machine.
- bool Hostname (const wxString &hostname)

  Set the address to hostname, which can be a host name or an IP-style address in dot notation(`a.b.c.d`).
- virtual wxString Hostname () const

  Returns the hostname which matches the IP address.
- virtual wxString IPAddress () const

  Returns a wxString containing the IP address in dot quad (127.0.0.1) format.

- bool LocalHost ()

  Set address to localhost (127.0.0.1).
- bool Service (const wxString &service)

  Set the port to that corresponding to the specified service.
- bool Service (unsigned short service)

  Set the port to that corresponding to the specified service.
- unsigned short Service () const

  Returns the current service.

Additional Inherited Members

21.410.2 Member Function Documentation

bool wxIPV4address::AnyAddress ( )

Set address to any of the addresses of the current machine.

Whenever possible, use this function instead of LocalHost(), as this correctly handles multi-homed hosts and avoids other small problems. Internally, this is the same as setting the IP address to **INADDR_ANY**.

Returns

true on success, false if something went wrong.

bool wxIPV4address::Hostname (const wxString & *hostname*)

Set the address to hostname, which can be a host name or an IP-style address in dot notation(`a.b.c.d`).

Returns

true on success, false if something goes wrong (invalid hostname or invalid IP address).

virtual wxString wxIPV4address::Hostname ( ) const `[virtual]`

Returns the hostname which matches the IP address.

virtual wxString wxIPV4address::IPAddress ( ) const `[virtual]`

Returns a wxString containing the IP address in dot quad (127.0.0.1) format.

Implements wxIPaddress.

bool wxIPV4address::LocalHost ( )

Set address to localhost (127.0.0.1).

Whenever possible, use AnyAddress() instead of this one, as that one will correctly handle multi-homed hosts and avoid other small problems.

Returns

true on success, false if something went wrong.

bool wxIPV4address::Service (const wxString & *service*)

Set the port to that corresponding to the specified *service*.

Returns

true on success, false if something goes wrong (invalid *service*).

bool wxIPV4address::Service (unsigned short *service*)

Set the port to that corresponding to the specified *service*.

Returns

true on success, false if something goes wrong (invalid *service*).

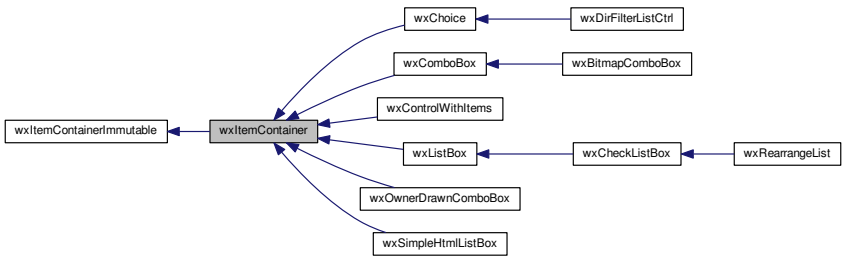
unsigned short wxIPV4address::Service ( ) const

Returns the current service.

21.411 wxItemContainer Class Reference

`#include <wx/ctrlsub.h>`

Inheritance diagram for wxItemContainer:



21.411.1 Detailed Description

This class is an abstract base class for some wxWidgets controls which contain several items such as wxListBox, wxCheckListBox, wxComboBox or wxChoice.

It defines an interface which is implemented by all controls which have string subitems each of which may be selected.

wxItemContainer extends wxItemContainerImmutable interface with methods for adding/removing items.

It defines the methods for accessing the controls items and although each of the derived classes implements them differently, they still all conform to the same interface.

The items in a wxItemContainer have (non-empty) string labels and, optionally, client data associated with them. Client data may be of two different kinds: either simple untyped (`void *`) pointers which are simply stored by the control but not used in any way by it, or typed pointers (wxClientData*) which are owned by the control meaning that the typed client data (and only it) will be deleted when an item is deleted using Delete() or the entire control is cleared using Clear(), which also happens when it is destroyed.

Finally note that in the same control all items must have client data of the same type (typed or untyped), if any. This type is determined by the first call to Append() (the version with client data pointer) or SetClientData().

Note that this is not a control, it's a mixin interface that classes have to derive from in addition to wxControl or wxWindow. Convenience class wxControlWithItems is provided for this purpose.

Library: wxCore

Category: Controls

See also

wxControlWithItems, wxItemContainerImmutable

Public Member Functions

- void Clear ()

  Removes all items from the control.
- void Delete (unsigned int n)

  Deletes an item from the control.
- wxClientData * DetachClientObject (unsigned int n)

  Returns the client object associated with the given item and transfers its ownership to the caller.
- bool HasClientData () const

  Returns true, if either untyped data (`void*`) or object data (wxClientData) is associated with the items of the control.*
- bool HasClientObjectData () const

  Returns true, if object data is associated with the items of the control.
- bool HasClientUntypedData () const

  Returns true, if untyped data (`void*`) is associated with the items of the control.

- int Append (const wxString &item)

  Appends item into the control.
- int Append (const wxString &item, void *clientData)

  Appends item into the control.
- int Append (const wxString &item, wxClientData *clientData)

  Appends item into the control.
- int Append (const wxArrayString &items)

  Appends several items at once into the control.
- template<typename T >
  int Append (const std::vector< T > &items)

  Appends several items at once into the control.
- int Append (const wxArrayString &items, void **clientData)

  Appends several items at once into the control.
- int Append (const wxArrayString &items, wxClientData **clientData)

  Appends several items at once into the control.
- int Append (unsigned int n, const wxString *items)

  Appends several items at once into the control.
- int Append (unsigned int n, const wxString *items, void **clientData)

  Appends several items at once into the control.
- int Append (unsigned int n, const wxString *items, wxClientData **clientData)

  Appends several items at once into the control.

- void * GetClientData (unsigned int n) const

  Returns a pointer to the client data associated with the given item (if any).
- wxClientData * GetClientObject (unsigned int n) const

  Returns a pointer to the client data associated with the given item (if any).
- void SetClientData (unsigned int n, void *data)

  Associates the given untyped client data pointer with the given item.
- void SetClientObject (unsigned int n, wxClientData *data)

  Associates the given typed client data pointer with the given item: the data object will be deleted when the item is deleted (either explicitly by using Delete() or implicitly when the control itself is destroyed).

- int Insert (const wxString &item, unsigned int pos)

  Inserts item into the control.
- int Insert (const wxString &item, unsigned int pos, void *clientData)

  Inserts item into the control.
- int Insert (const wxString &item, unsigned int pos, wxClientData *clientData)

  Inserts item into the control.
- int Insert (const wxArrayString &items, unsigned int pos)

  Inserts several items at once into the control.
- template<typename T >
  int Insert (const std::vector< T > &items)

  Inserts several items at once into the control.
- int Insert (const wxArrayString &items, unsigned int pos, void **clientData)

  Inserts several items at once into the control.
- int Insert (const wxArrayString &items, unsigned int pos, wxClientData **clientData)

  Inserts several items at once into the control.
- int Insert (unsigned int n, const wxString *items, unsigned int pos)

  Inserts several items at once into the control.
- int Insert (unsigned int n, const wxString *items, unsigned int pos, void **clientData)

  Inserts several items at once into the control.
- int Insert (unsigned int n, const wxString *items, unsigned int pos, wxClientData **clientData)

  Inserts several items at once into the control.

- void Set (const wxArrayString &items)

  Replaces the current control contents with the given items.
- template<typename T >
  void Set (const std::vector< T > &items)

  Replaces the current control contents with the given items.
- void Set (const wxArrayString &items, void **clientData)

  Replaces the current control contents with the given items.
- void Set (const wxArrayString &items, wxClientData **clientData)

  Replaces the current control contents with the given items.
- void Set (unsigned int n, const wxString *items)

  Replaces the current control contents with the given items.
- void Set (unsigned int n, const wxString *items, void **clientData)

  Replaces the current control contents with the given items.
- void Set (unsigned int n, const wxString *items, wxClientData **clientData)

  Replaces the current control contents with the given items.

21.411.2 Member Function Documentation

int wxItemContainer::Append (const wxString & *item*)

Appends item into the control.

Parameters

item	String to add.

Returns

The return value is the index of the newly inserted item. Note that this may be different from the last one if the control is sorted (e.g. has `wxLB_SORT` or `wxCB_SORT` style).

int wxItemContainer::Append (const wxString & *item,* void * *clientData*)

Appends item into the control.

Parameters

item	String to add.
clientData	Pointer to client data to associate with the new item.

Returns

The return value is the index of the newly inserted item. Note that this may be different from the last one if the control is sorted (e.g. has `wxLB_SORT` or `wxCB_SORT` style).

int wxItemContainer::Append (const wxString & *item,* wxClientData * *clientData*)

Appends item into the control.

Parameters

item	String to add.
clientData	Pointer to client data to associate with the new item.

Returns

The return value is the index of the newly inserted item. Note that this may be different from the last one if the control is sorted (e.g. has `wxLB_SORT` or `wxCB_SORT` style).

int wxItemContainer::Append (const wxArrayString & *items*)

Appends several items at once into the control.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

items	Array of strings to insert.

template<typename T > int wxItemContainer::Append (const std::vector< T > & *items*)

Appends several items at once into the control.

This is the same as the overload taking wxArrayString, except that it works with the standard vector container.

The template argument `T` can be any type convertible to wxString, including wxString itself but also `std::string`, `char*` or `wchar_t*`.

Since

3.1.0

int wxItemContainer::Append (const wxArrayString & *items*, void ∗∗ *clientData*)

Appends several items at once into the control.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

items	Array of strings to insert.
clientData	Array of client data pointers of the same size as *items* to associate with the new items.

int wxItemContainer::Append (const wxArrayString & *items*, wxClientData ∗∗ *clientData*)

Appends several items at once into the control.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

items	Array of strings to insert.
clientData	Array of client data pointers of the same size as *items* to associate with the new items.

int wxItemContainer::Append (unsigned int *n*, const wxString ∗ *items*)

Appends several items at once into the control.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.

int wxItemContainer::Append (unsigned int *n*, const wxString ∗ *items*, void ∗∗ *clientData*)

Appends several items at once into the control.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.
clientData	Array of client data pointers of size *n* to associate with the new items.

int wxItemContainer::Append (unsigned int *n*, const wxString ∗ *items*, wxClientData ∗∗ *clientData*)

Appends several items at once into the control.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.
clientData	Array of client data pointers of size *n* to associate with the new items.

void wxItemContainer::Clear ( )

Removes all items from the control.

Clear() also deletes the client data of the existing items if it is owned by the control.

void wxItemContainer::Delete (unsigned int *n*)

Deletes an item from the control.

The client data associated with the item will be also deleted if it is owned by the control. Note that it is an error (signalled by an assert failure in debug builds) to remove an item with the index negative or greater or equal than the number of items in the control.

Parameters

n	The zero-based item index.

See also

Clear()

wxClientData* wxItemContainer::DetachClientObject (unsigned int *n*)

Returns the client object associated with the given item and transfers its ownership to the caller.

This method, unlike GetClientObject(), expects the caller to delete the returned pointer. It also replaces the internally stored pointer with NULL, i.e. completely detaches the client object pointer from the control.

It's an error to call this method unless HasClientObjectData() returns true.

Parameters

n	The zero-based item index.

Returns

The associated client object pointer to be deleted by caller or NULL.

Since

2.9.2

void* wxItemContainer::GetClientData (unsigned int *n*) const

Returns a pointer to the client data associated with the given item (if any).

It is an error to call this function for a control which doesn't have untyped client data at all although it is OK to call it even if the given item doesn't have any client data associated with it (but other items do).

Parameters

n	The zero-based position of the item.

Returns

A pointer to the client data, or NULL if not present.

wxClientData* wxItemContainer::GetClientObject (unsigned int *n*) const

Returns a pointer to the client data associated with the given item (if any).

It is an error to call this function for a control which doesn't have typed client data at all although it is OK to call it even if the given item doesn't have any client data associated with it (but other items do).

Notice that the returned pointer is still owned by the control and will be deleted by it, use DetachClientObject() if you want to remove the pointer from the control.

Parameters

n	The zero-based position of the item.

Returns

A pointer to the client data, or NULL if not present.

bool wxItemContainer::HasClientData ( ) const

Returns true, if either untyped data (`void*`) or object data (wxClientData*) is associated with the items of the control.

bool wxItemContainer::HasClientObjectData ( ) const

Returns true, if object data is associated with the items of the control.

Object data pointers have the type `wxClientData*` instead of `void*` and, importantly, are owned by the control, i.e. will be deleted by it, unlike their untyped counterparts.

bool wxItemContainer::HasClientUntypedData ( ) const

Returns true, if untyped data (`void*`) is associated with the items of the control.

int wxItemContainer::Insert (const wxString & *item*, unsigned int *pos*)

Inserts item into the control.

Parameters

item	String to add.
pos	Position to insert item before, zero based.

Returns

The return value is the index of the newly inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (const wxString & *item*, unsigned int *pos*, void * *clientData*)

Inserts item into the control.

Parameters

item	String to add.
pos	Position to insert item before, zero based.
clientData	Pointer to client data to associate with the new item.

Returns

The return value is the index of the newly inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (const wxString & *item*, unsigned int *pos*, wxClientData * *clientData*)

Inserts item into the control.

Parameters

item	String to add.
pos	Position to insert item before, zero based.
clientData	Pointer to client data to associate with the new item.

Returns

The return value is the index of the newly inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (const wxArrayString & *items*, unsigned int *pos*)

Inserts several items at once into the control.

Notice that calling this method is usually much faster than inserting them one by one if you need to insert a lot of items.

Parameters

items	Array of strings to insert.
pos	Position to insert the items before, zero based.

Returns

The return value is the index of the last inserted item. If the insertion failed for some reason, -1 is returned.

template<typename T > int wxItemContainer::Insert (const std::vector< T > & *items*)

Inserts several items at once into the control.

This is the same as the overload taking wxArrayString, except that it works with the standard vector container.

The template argument `T` can be any type convertible to wxString, including wxString itself but also `std::string`, `char*` or `wchar_t*`.

Since

3.1.0

int wxItemContainer::Insert (const wxArrayString & *items*, unsigned int *pos*, void ** *clientData*)

Inserts several items at once into the control.

Notice that calling this method is usually much faster than inserting them one by one if you need to insert a lot of items.

Parameters

items	Array of strings to insert.
pos	Position to insert the items before, zero based.
clientData	Array of client data pointers of the same size as *items* to associate with the new items.

Returns

The return value is the index of the last inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (const wxArrayString & *items,* unsigned int *pos,* wxClientData ∗∗ *clientData*)

Inserts several items at once into the control.

Notice that calling this method is usually much faster than inserting them one by one if you need to insert a lot of items.

Parameters

items	Array of strings to insert.
pos	Position to insert the items before, zero based.
clientData	Array of client data pointers of the same size as *items* to associate with the new items.

Returns

The return value is the index of the last inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (unsigned int *n,* const wxString ∗ *items,* unsigned int *pos*)

Inserts several items at once into the control.

Notice that calling this method is usually much faster than inserting them one by one if you need to insert a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.
pos	Position to insert the items before, zero based.

Returns

The return value is the index of the last inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (unsigned int *n,* const wxString ∗ *items,* unsigned int *pos,* void ∗∗ *clientData*)

Inserts several items at once into the control.

Notice that calling this method is usually much faster than inserting them one by one if you need to insert a lot of items.

Parameters

n	Number of items in the *items* array.

items	Array of strings of size *n*.
pos	Position to insert the new items before, zero based.
clientData	Array of client data pointers of size *n* to associate with the new items.

Returns

The return value is the index of the last inserted item. If the insertion failed for some reason, -1 is returned.

int wxItemContainer::Insert (unsigned int *n*, const wxString * *items*, unsigned int *pos*, wxClientData ** *clientData*)

Inserts several items at once into the control.

Notice that calling this method is usually much faster than inserting them one by one if you need to insert a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.
pos	Position to insert the new items before, zero based.
clientData	Array of client data pointers of size *n* to associate with the new items.

Returns

The return value is the index of the last inserted item. If the insertion failed for some reason, -1 is returned.

void wxItemContainer::Set (const wxArrayString & *items*)

Replaces the current control contents with the given items.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

items	Array of strings to insert.

template<typename T > void wxItemContainer::Set (const std::vector< T > & *items*)

Replaces the current control contents with the given items.

This is the same as the overload taking wxArrayString, except that it works with the standard vector container.

The template argument `T` can be any type convertible to wxString, including wxString itself but also `std::string`, `char*` or `wchar_t*`.

Since

3.1.0

void wxItemContainer::Set (const wxArrayString & *items*, void ** *clientData*)

Replaces the current control contents with the given items.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

items	Array of strings to insert.
clientData	Array of client data pointers of the same size as *items* to associate with the new items.

void wxItemContainer::Set (const wxArrayString & *items*, wxClientData ∗∗ *clientData*)

Replaces the current control contents with the given items.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

items	Array of strings to insert.
clientData	Array of client data pointers of the same size as *items* to associate with the new items.

void wxItemContainer::Set (unsigned int *n*, const wxString ∗ *items*)

Replaces the current control contents with the given items.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.

void wxItemContainer::Set (unsigned int *n*, const wxString ∗ *items*, void ∗∗ *clientData*)

Replaces the current control contents with the given items.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.
clientData	Array of client data pointers of size *n* to associate with the new items.

void wxItemContainer::Set (unsigned int *n*, const wxString ∗ *items*, wxClientData ∗∗ *clientData*)

Replaces the current control contents with the given items.

Notice that calling this method is usually much faster than appending them one by one if you need to add a lot of items.

Parameters

n	Number of items in the *items* array.
items	Array of strings of size *n*.
clientData	Array of client data pointers of size *n* to associate with the new items.

void wxItemContainer::SetClientData (unsigned int *n*, void ∗ *data*)

Associates the given untyped client data pointer with the given item.

Note that it is an error to call this function if any typed client data pointers had been associated with the control items before.

Parameters

n	The zero-based item index.
data	The client data to associate with the item.

void wxItemContainer::SetClientObject (unsigned int *n*, wxClientData * *data*)

Associates the given typed client data pointer with the given item: the *data* object will be deleted when the item is deleted (either explicitly by using Delete() or implicitly when the control itself is destroyed).

Note that it is an error to call this function if any untyped client data pointers had been associated with the control items before.

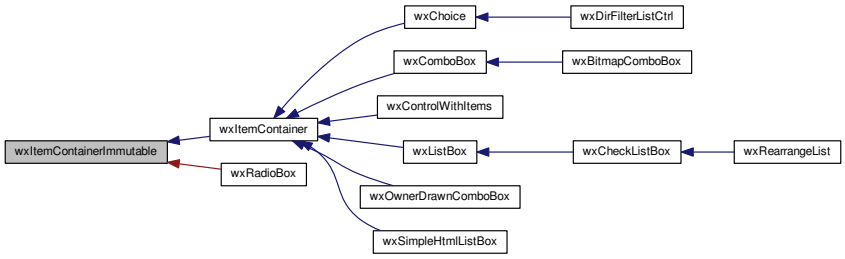
Parameters

n	The zero-based item index.
data	The client data to associate with the item.

21.412 wxItemContainerImmutable Class Reference

`#include <wx/ctrlsub.h>`

Inheritance diagram for wxItemContainerImmutable:



21.412.1 Detailed Description

wxItemContainer defines an interface which is implemented by all controls which have string subitems each of which may be selected.

It is decomposed in wxItemContainerImmutable which omits all methods adding/removing items and is used by wxRadioBox and wxItemContainer itself.

Note that this is not a control, it's a mixin interface that classes have to derive from in addition to wxControl or wxWindow.

Examples: wxListBox, wxCheckListBox, wxChoice and wxComboBox (which implements an extended interface deriving from this one)

Library: wxCore

Category: Controls

See also

wxControlWithItems, wxItemContainer

Public Member Functions

- wxItemContainerImmutable ()

  Constructor.

- virtual unsigned int GetCount () const =0

  Returns the number of items in the control.
- bool IsEmpty () const

  Returns true if the control is empty or false if it has some items.
- virtual wxString GetString (unsigned int n) const =0

  Returns the label of the item with the given index.
- wxArrayString GetStrings () const

  Returns the array of the labels of all items in the control.
- virtual void SetString (unsigned int n, const wxString &string)=0

  Sets the label for the given item.
- virtual int FindString (const wxString &string, bool caseSensitive=false) const

  Finds an item whose label matches the given string.

Selection

- virtual void SetSelection (int n)=0

  Sets the selection to the given item n or removes the selection entirely if n == `wxNOT_FOUND`.
- virtual int GetSelection () const =0

  Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.
- bool SetStringSelection (const wxString &string)

  Selects the item with the specified string in the control.
- virtual wxString GetStringSelection () const

  Returns the label of the selected item or an empty string if no item is selected.
- void Select (int n)

  This is the same as SetSelection() and exists only because it is slightly more natural for controls which support multiple selection.

21.412.2 Constructor & Destructor Documentation

wxItemContainerImmutable::wxItemContainerImmutable ( )

Constructor.

21.412.3 Member Function Documentation

virtual int wxItemContainerImmutable::FindString (const wxString & *string*, bool *caseSensitive* = `false`) const `[virtual]`

Finds an item whose label matches the given string.

Parameters

string	String to find.
caseSensitive	Whether search is case sensitive (default is not).

Returns

The zero-based position of the item, or wxNOT_FOUND if the string was not found.

Reimplemented in wxListBox, wxComboBox, wxRadioBox, and wxChoice.

virtual unsigned int wxItemContainerImmutable::GetCount ( ) const `[pure virtual]`

Returns the number of items in the control.

See also

IsEmpty()

Implemented in wxRadioBox, wxComboBox, wxListBox, and wxChoice.

virtual int wxItemContainerImmutable::GetSelection ( ) const `[pure virtual]`

Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

Returns

The position of the current selection.

Remarks

This method can be used with single selection list boxes only, you should use wxListBox::GetSelections() for the list boxes with wxLB_MULTIPLE style.

See also

SetSelection(), GetStringSelection()

Implemented in wxRadioBox, wxComboBox, wxListBox, and wxChoice.

virtual wxString wxItemContainerImmutable::GetString (unsigned int *n*) const `[pure virtual]`

Returns the label of the item with the given index.

Parameters

n	The zero-based index.

Returns

The label of the item or an empty string if the position was invalid.

Implemented in wxRadioBox, wxListBox, wxComboBox, and wxChoice.

wxArrayString wxItemContainerImmutable::GetStrings ( ) const

Returns the array of the labels of all items in the control.

virtual wxString wxItemContainerImmutable::GetStringSelection ( ) const `[virtual]`

Returns the label of the selected item or an empty string if no item is selected.

See also

GetSelection()

Reimplemented in wxComboBox.

bool wxItemContainerImmutable::IsEmpty ( ) const

Returns true if the control is empty or false if it has some items.

See also

GetCount()

void wxItemContainerImmutable::Select (int *n*)

This is the same as SetSelection() and exists only because it is slightly more natural for controls which support multiple selection.

virtual void wxItemContainerImmutable::SetSelection (int *n*) `[pure virtual]`

Sets the selection to the given item *n* or removes the selection entirely if *n* == `wxNOT_FOUND`.

Note that this does not cause any command events to be emitted nor does it deselect any other items in the controls which support multiple selections.

Parameters

n	The string position to select, starting from zero.

See also

SetString(), SetStringSelection()

Implemented in wxComboBox, wxRadioBox, wxChoice, and wxListBox.

virtual void wxItemContainerImmutable::SetString (unsigned int *n*, const wxString & *string*) `[pure virtual]`

Sets the label for the given item.

Parameters

n	The zero-based item index.
string	The label to set.

Implemented in wxRadioBox, wxComboBox, wxListBox, and wxChoice.

bool wxItemContainerImmutable::SetStringSelection (const wxString & *string*)

Selects the item with the specified string in the control.

This method doesn't cause any command events to be emitted.

Notice that this method is case-insensitive, i.e. the string is compared with all the elements of the control case-insensitively and the first matching entry is selected, even if it doesn't have exactly the same case as this string and there is an exact match afterwards.

Parameters

string	The string to select.

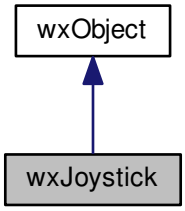
Returns

true if the specified string has been selected, false if it wasn't found in the control.

21.413 wxJoystick Class Reference

`#include <wx/joystick.h>`

Inheritance diagram for wxJoystick:



21.413.1 Detailed Description

wxJoystick allows an application to control one or more joysticks.

Library: wxAdvanced

Category: Miscellaneous

See also

wxJoystickEvent

Public Member Functions

- wxJoystick (int joystick=wxJOYSTICK1)
  Constructor.
- virtual ~wxJoystick ()
  Destroys the wxJoystick object.
- int GetButtonState () const
  Returns the state of the joystick buttons.
- bool GetButtonState (unsigned int id) const
  Returns the state of the specified joystick button.
- int GetManufacturerId () const
  Returns the manufacturer id.

- int GetMovementThreshold () const

  Returns the movement threshold, the number of steps outside which the joystick is deemed to have moved.
- int GetNumberAxes () const

  Returns the number of axes for this joystick.
- int GetNumberButtons () const

  Returns the number of buttons for this joystick.
- int GetPOVCTSPosition () const

  Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units.
- int GetPOVPosition () const

  Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units, but limited to return 0, 9000, 18000 or 27000.
- int GetPollingMax () const

  Returns the maximum polling frequency.
- int GetPollingMin () const

  Returns the minimum polling frequency.
- wxPoint GetPosition () const

  Returns the x, y position of the joystick.
- int GetPosition (unsigned int axis) const

  Returns the position of the specified joystick axis.
- int GetProductId () const

  Returns the product id for the joystick.
- wxString GetProductName () const

  Returns the product name for the joystick.
- int GetRudderMax () const

  Returns the maximum rudder position.
- int GetRudderMin () const

  Returns the minimum rudder position.
- int GetRudderPosition () const

  Returns the rudder position.
- int GetUMax () const

  Returns the maximum U position.
- int GetUMin () const

  Returns the minimum U position.
- int GetUPosition () const

  Gets the position of the fifth axis of the joystick, if it exists.
- int GetVMax () const

  Returns the maximum V position.
- int GetVMin () const

  Returns the minimum V position.
- int GetVPosition () const

  Gets the position of the sixth axis of the joystick, if it exists.
- int GetXMax () const

  Returns the maximum x position.
- int GetXMin () const

  Returns the minimum x position.
- int GetYMax () const

  Returns the maximum y position.
- int GetYMin () const

  Returns the minimum y position.
- int GetZMax () const

  Returns the maximum z position.

- int GetZMin () const

  Returns the minimum z position.
- int GetZPosition () const

  Returns the z position of the joystick.
- bool HasPOV () const

  Returns true if the joystick has a point of view control.
- bool HasPOV4Dir () const

  Returns true if the joystick point-of-view supports discrete values (centered, forward, backward, left, and right).
- bool HasPOVCTS () const

  Returns true if the joystick point-of-view supports continuous degree bearings.
- bool HasRudder () const

  Returns true if there is a rudder attached to the computer.
- bool HasU () const

  Returns true if the joystick has a U axis.
- bool HasV () const

  Returns true if the joystick has a V axis.
- bool HasZ () const

  Returns true if the joystick has a Z axis.
- bool IsOk () const

  Returns true if the joystick is functioning.
- bool ReleaseCapture ()

  *Releases the capture set by **SetCapture**.*
- bool SetCapture (wxWindow *win, int pollingFreq=0)

  Sets the capture to direct joystick events to win.
- void SetMovementThreshold (int threshold)

  Sets the movement threshold, the number of steps outside which the joystick is deemed to have moved.

Static Public Member Functions

- static int GetNumberJoysticks ()

  Returns the number of joysticks currently attached to the computer.

Additional Inherited Members

21.413.2 Constructor & Destructor Documentation

wxJoystick::wxJoystick (int *joystick* = wxJOYSTICK1)

Constructor.

joystick may be one of wxJOYSTICK1, wxJOYSTICK2, indicating the joystick controller of interest.

virtual wxJoystick::~wxJoystick ( ) [virtual]

Destroys the wxJoystick object.

21.413.3 Member Function Documentation

int wxJoystick::GetButtonState ( ) const

Returns the state of the joystick buttons.

Every button is mapped to a single bit in the returned integer, with the first button being mapped to the least significant bit, and so on.

A bitlist of wxJOY_BUTTONn identifiers, where n is 1, 2, 3 or 4 is available for historical reasons.

bool wxJoystick::GetButtonState (unsigned int *id*) const

Returns the state of the specified joystick button.

Parameters

id	The button id to report, from 0 to GetNumberButtons() - 1

int wxJoystick::GetManufacturerId ( ) const

Returns the manufacturer id.

int wxJoystick::GetMovementThreshold ( ) const

Returns the movement threshold, the number of steps outside which the joystick is deemed to have moved.

int wxJoystick::GetNumberAxes ( ) const

Returns the number of axes for this joystick.

int wxJoystick::GetNumberButtons ( ) const

Returns the number of buttons for this joystick.

static int wxJoystick::GetNumberJoysticks ( ) [static]

Returns the number of joysticks currently attached to the computer.

int wxJoystick::GetPollingMax ( ) const

Returns the maximum polling frequency.

int wxJoystick::GetPollingMin ( ) const

Returns the minimum polling frequency.

wxPoint wxJoystick::GetPosition ( ) const

Returns the x, y position of the joystick.

int wxJoystick::GetPosition (unsigned int *axis*) const

Returns the position of the specified joystick axis.

Parameters

axis	The joystick axis to report, from 0 to GetNumberAxes() - 1.

int wxJoystick::GetPOVCTSPosition ( ) const

Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units.

Returns -1 on error.

int wxJoystick::GetPOVPosition ( ) const

Returns the point-of-view position, expressed in continuous, one-hundredth of a degree units, but limited to return 0, 9000, 18000 or 27000.

Returns -1 on error.

int wxJoystick::GetProductId ( ) const

Returns the product id for the joystick.

wxString wxJoystick::GetProductName ( ) const

Returns the product name for the joystick.

int wxJoystick::GetRudderMax ( ) const

Returns the maximum rudder position.

int wxJoystick::GetRudderMin ( ) const

Returns the minimum rudder position.

int wxJoystick::GetRudderPosition ( ) const

Returns the rudder position.

int wxJoystick::GetUMax ( ) const

Returns the maximum U position.

int wxJoystick::GetUMin ( ) const

Returns the minimum U position.

int wxJoystick::GetUPosition ( ) const

Gets the position of the fifth axis of the joystick, if it exists.

int wxJoystick::GetVMax ( ) const

Returns the maximum V position.

int wxJoystick::GetVMin ( ) const

Returns the minimum V position.

int wxJoystick::GetVPosition ( ) const

Gets the position of the sixth axis of the joystick, if it exists.

int wxJoystick::GetXMax ( ) const

Returns the maximum x position.

int wxJoystick::GetXMin ( ) const

Returns the minimum x position.

int wxJoystick::GetYMax ( ) const

Returns the maximum y position.

int wxJoystick::GetYMin ( ) const

Returns the minimum y position.

int wxJoystick::GetZMax ( ) const

Returns the maximum z position.

int wxJoystick::GetZMin ( ) const

Returns the minimum z position.

int wxJoystick::GetZPosition ( ) const

Returns the z position of the joystick.

bool wxJoystick::HasPOV ( ) const

Returns true if the joystick has a point of view control.

bool wxJoystick::HasPOV4Dir ( ) const

Returns true if the joystick point-of-view supports discrete values (centered, forward, backward, left, and right).

bool wxJoystick::HasPOVCTS ( ) const

Returns true if the joystick point-of-view supports continuous degree bearings.

bool wxJoystick::HasRudder ( ) const

Returns true if there is a rudder attached to the computer.

bool wxJoystick::HasU ( ) const

Returns true if the joystick has a U axis.

bool wxJoystick::HasV ( ) const

Returns true if the joystick has a V axis.

bool wxJoystick::HasZ ( ) const

Returns true if the joystick has a Z axis.

bool wxJoystick::IsOk ( ) const

Returns true if the joystick is functioning.

bool wxJoystick::ReleaseCapture ( )

Releases the capture set by **SetCapture**.

Returns

true if the capture release succeeded.

See also

SetCapture(), wxJoystickEvent

bool wxJoystick::SetCapture (wxWindow * *win*, int *pollingFreq* = 0)

Sets the capture to direct joystick events to *win*.

Parameters

win	The window that will receive joystick events.
pollingFreq	If zero, movement events are sent when above the threshold. If greater than zero, events are received every *pollingFreq* milliseconds.

Returns

true if the capture succeeded.

See also

ReleaseCapture(), wxJoystickEvent

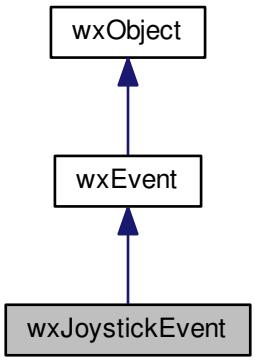
void wxJoystick::SetMovementThreshold (int *threshold*)

Sets the movement threshold, the number of steps outside which the joystick is deemed to have moved.

21.414 wxJoystickEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxJoystickEvent:



21.414.1 Detailed Description

This event class contains information about joystick events, particularly events received by windows.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxJoystickEvent& event)

Event macros:

- EVT_JOY_BUTTON_DOWN(func): Process a `wxEVT_JOY_BUTTON_DOWN` event.
- EVT_JOY_BUTTON_UP(func): Process a `wxEVT_JOY_BUTTON_UP` event.
- EVT_JOY_MOVE(func): Process a `wxEVT_JOY_MOVE` event.
- EVT_JOY_ZMOVE(func): Process a `wxEVT_JOY_ZMOVE` event.
- EVT_JOYSTICK_EVENTS(func): Processes all joystick events.

Library: wxCore

Category: Events

See also

wxJoystick

Public Member Functions

- wxJoystickEvent (wxEventType eventType=wxEVT_NULL, int state=0, int joystick=wxJOYSTICK1, int change=0)

  Constructor.
- bool ButtonDown (int button=wxJOY_BUTTON_ANY) const

  Returns true if the event was a down event from the specified button (or any button).
- bool ButtonIsDown (int button=wxJOY_BUTTON_ANY) const

  Returns true if the specified button (or any button) was in a down state.
- bool ButtonUp (int button=wxJOY_BUTTON_ANY) const

  Returns true if the event was an up event from the specified button (or any button).
- int GetButtonChange () const

  Returns the identifier of the button changing state.
- int GetButtonState () const

  Returns the down state of the buttons.
- int GetJoystick () const

  Returns the identifier of the joystick generating the event - one of wxJOYSTICK1 and wxJOYSTICK2.
- wxPoint GetPosition () const

  Returns the x, y position of the joystick event.
- int GetZPosition () const

  Returns the z position of the joystick event.
- bool IsButton () const

  Returns true if this was a button up or down event (not 'is any button down?').
- bool IsMove () const

  Returns true if this was an x, y move event.
- bool IsZMove () const

  Returns true if this was a z move event.

Additional Inherited Members

21.414.2 Constructor & Destructor Documentation

wxJoystickEvent::wxJoystickEvent (wxEventType *eventType* = wxEVT_NULL, int *state* = 0, int *joystick* = wxJOYSTICK1, int *change* = 0)

Constructor.

21.414.3 Member Function Documentation

bool wxJoystickEvent::ButtonDown (int *button* = wxJOY_BUTTON_ANY) const

Returns true if the event was a down event from the specified button (or any button).

Parameters

button	Can be `wxJOY_BUTTONn` where `n` is 1, 2, 3 or 4; or `wxJOY_BUTTON_ANY` to indicate any button down event.

bool wxJoystickEvent::ButtonIsDown (int *button* = wxJOY_BUTTON_ANY) const

Returns true if the specified button (or any button) was in a down state.

Parameters

button	Can be `wxJOY_BUTTONn` where `n` is 1, 2, 3 or 4; or `wxJOY_BUTTON_ANY` to indicate any button down event.

bool wxJoystickEvent::ButtonUp (int *button* = wxJOY_BUTTON_ANY) const

Returns true if the event was an up event from the specified button (or any button).

Parameters

button	Can be `wxJOY_BUTTONn` where `n` is 1, 2, 3 or 4; or `wxJOY_BUTTON_ANY` to indicate any button down event.

int wxJoystickEvent::GetButtonChange ( ) const

Returns the identifier of the button changing state.

This is a `wxJOY_BUTTONn` identifier, where `n` is one of 1, 2, 3, 4.

int wxJoystickEvent::GetButtonState ( ) const

Returns the down state of the buttons.

This is a `wxJOY_BUTTONn` identifier, where `n` is one of 1, 2, 3, 4.

int wxJoystickEvent::GetJoystick ( ) const

Returns the identifier of the joystick generating the event - one of wxJOYSTICK1 and wxJOYSTICK2.

wxPoint wxJoystickEvent::GetPosition ( ) const

Returns the x, y position of the joystick event.

These coordinates are valid for all the events except wxEVT_JOY_ZMOVE.

int wxJoystickEvent::GetZPosition ( ) const

Returns the z position of the joystick event.

This method can only be used for wxEVT_JOY_ZMOVE events.

bool wxJoystickEvent::IsButton ( ) const

Returns true if this was a button up or down event (*not* 'is any button down?').

bool wxJoystickEvent::IsMove ( ) const

Returns true if this was an x, y move event.

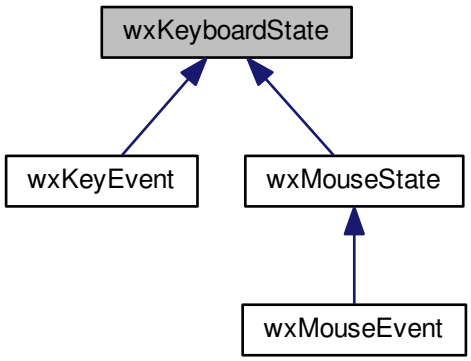
bool wxJoystickEvent::IsZMove ( ) const

Returns true if this was a z move event.

21.415 wxKeyboardState Class Reference

`#include <wx/kbdstate.h>`

Inheritance diagram for wxKeyboardState:



21.415.1 Detailed Description

Provides methods for testing the state of the keyboard modifier keys.

This class is used as a base class of wxKeyEvent and wxMouseState and, hence, indirectly, of wxMouseEvent, so its methods may be used to get information about the modifier keys which were pressed when the event occurred.

This class is implemented entirely inline in <wx/kbdstate.h> and thus has no linking requirements.

Library: None; this class implementation is entirely header-based.

Category: Events

See also

wxKeyEvent, wxMouseState

Public Member Functions

- wxKeyboardState (bool controlDown=false, bool shiftDown=false, bool altDown=false, bool metaDown=false)

Constructor initializes the modifier key settings.

- int GetModifiers () const

  Return the bit mask of all pressed modifier keys.

- bool HasAnyModifiers () const

  Returns true if any modifiers at all are pressed.

- bool HasModifiers () const

  Returns true if Control or Alt are pressed.

- bool ControlDown () const

  Returns true if the Control key or Apple/Command key under OS X is pressed.

- bool RawControlDown () const

  Returns true if the Control key (also under OS X).

- bool ShiftDown () const

  Returns true if the Shift key is pressed.

- bool MetaDown () const

  Returns true if the Meta/Windows/Apple key is pressed.

- bool AltDown () const

  Returns true if the Alt key is pressed.

- bool CmdDown () const

  Returns true if the key used for command accelerators is pressed.

- void SetControlDown (bool down)
- void SetRawControlDown (bool down)
- void SetShiftDown (bool down)
- void SetAltDown (bool down)
- void SetMetaDown (bool down)

21.415.2 Constructor & Destructor Documentation

wxKeyboardState::wxKeyboardState (bool *controlDown* = `false`, bool *shiftDown* = `false`, bool *altDown* = `false`, bool *metaDown* = `false`)

Constructor initializes the modifier key settings.

By default, no modifiers are active.

21.415.3 Member Function Documentation

bool wxKeyboardState::AltDown ( ) const

Returns true if the Alt key is pressed.

Notice that GetModifiers() should usually be used instead of this one.

bool wxKeyboardState::CmdDown ( ) const

Returns true if the key used for command accelerators is pressed.

Same as ControlDown(). Deprecated.

Notice that GetModifiers() should usually be used instead of this one.

bool wxKeyboardState::ControlDown ( ) const

Returns true if the Control key or Apple/Command key under OS X is pressed.

This function doesn't distinguish between right and left control keys.

Notice that GetModifiers() should usually be used instead of this one.

int wxKeyboardState::GetModifiers ( ) const

Return the bit mask of all pressed modifier keys.

The return value is a combination of `wxMOD_ALT`, `wxMOD_CONTROL`, `wxMOD_SHIFT` and `wxMOD_META` bit masks. Additionally, `wxMOD_NONE` is defined as 0, i.e. corresponds to no modifiers (see HasAnyModifiers()) and `wxMOD_CMD` is either `wxMOD_CONTROL` (MSW and Unix) or `wxMOD_META` (Mac), see CmdDown(). See wxKeyModifier for the full list of modifiers.

Notice that this function is easier to use correctly than, for example, ControlDown() because when using the latter you also have to remember to test that none of the other modifiers is pressed:

```
if ( ControlDown() && !AltDown() && !ShiftDown() && !
      MetaDown() )
    ... handle Ctrl-XXX ...
```

and forgetting to do it can result in serious program bugs (e.g. program not working with European keyboard layout where `AltGr` key which is seen by the program as combination of CTRL and ALT is used). On the other hand, you can simply write:

```
if ( GetModifiers() == wxMOD_CONTROL )
    ... handle Ctrl-XXX ...
```

with this function.

bool wxKeyboardState::HasAnyModifiers ( ) const

Returns true if any modifiers at all are pressed.

This is equivalent to `GetModifiers() != wxMOD_NONE`.

Notice that this is different from HasModifiers() method which doesn't take e.g. Shift modifier into account. This method is most suitable for mouse events when any modifier, including Shift, can change the interpretation of the event.

Since

2.9.5

bool wxKeyboardState::HasModifiers ( ) const

Returns true if Control or Alt are pressed.

Checks if Control, Alt or, under OS X only, Command key are pressed (notice that the real Control key is still taken into account under OS X too).

This method returns false if only Shift is pressed for compatibility reasons and also because pressing Shift usually doesn't change the interpretation of key events, see HasAnyModifiers() if you want to take Shift into account as well.

bool wxKeyboardState::MetaDown ( ) const

Returns true if the Meta/Windows/Apple key is pressed.

This function tests the state of the key traditionally called Meta under Unix systems, Windows keys under MSW Notice that GetModifiers() should usually be used instead of this one.

See also

CmdDown()

bool wxKeyboardState::RawControlDown ( ) const

Returns true if the Control key (also under OS X).

This function doesn't distinguish between right and left control keys.

Notice that GetModifiers() should usually be used instead of this one.

void wxKeyboardState::SetAltDown (bool *down*)

void wxKeyboardState::SetControlDown (bool *down*)

void wxKeyboardState::SetMetaDown (bool *down*)

void wxKeyboardState::SetRawControlDown (bool *down*)

void wxKeyboardState::SetShiftDown (bool *down*)

bool wxKeyboardState::ShiftDown ( ) const

Returns true if the Shift key is pressed.

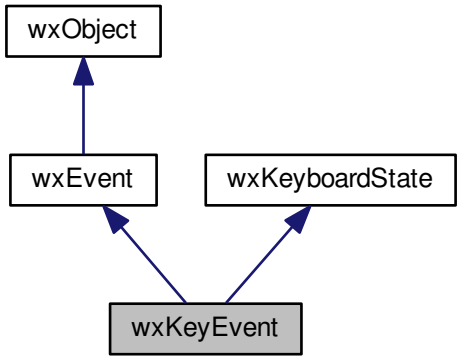
This function doesn't distinguish between right and left shift keys.

Notice that GetModifiers() should usually be used instead of this one.

21.416 wxKeyEvent Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxKeyEvent:



21.416.1 Detailed Description

This event class contains information about key press and release events.

The main information carried by this event is the key being pressed or released. It can be accessed using either GetKeyCode() function or GetUnicodeKey(). For the printable characters, the latter should be used as it works for any keys, including non-Latin-1 characters that can be entered when using national keyboard layouts. GetKeyCode() should be used to handle special characters (such as cursor arrows keys or `HOME` or `INS` and so on) which correspond to wxKeyCode enum elements above the `WXK_START` constant. While GetKeyCode() also returns the character code for Latin-1 keys for compatibility, it doesn't work for Unicode characters in general and will return `WXK_NONE` for any non-Latin-1 ones. For this reason, it's recommended to always use GetUnicodeKey() and only fall back to GetKeyCode() if GetUnicodeKey() returned `WXK_NONE` meaning that the event corresponds to a non-printable special keys.

While both of these functions can be used with the events of `wxEVT_KEY_DOWN`, `wxEVT_KEY_UP` and `wxEVT_CHAR` types, the values returned by them are different for the first two events and the last one. For the latter, the key returned corresponds to the character that would appear in e.g. a text zone if the user pressed the key in it. As such, its value depends on the current state of the Shift key and, for the letters, on the state of Caps Lock modifier. For example, if `A` key is pressed without Shift being held down, wxKeyEvent of type `wxEVT_CHAR` generated for this key press will return (from either GetKeyCode() or GetUnicodeKey() as their meanings coincide for ASCII characters) key code of 97 corresponding the ASCII value of `a`. And if the same key is pressed but with Shift being held (or Caps Lock being active), then the key could would be 65, i.e. ASCII value of capital `A`.

However for the key down and up events the returned key code will instead be `A` independently of the state of the modifier keys i.e. it depends only on physical key being pressed and is not translated to its logical representation using the current keyboard state. Such untranslated key codes are defined as follows:

- For the letters they correspond to the *upper* case value of the letter.
- For the other alphanumeric keys (e.g. `7` or `+`), the untranslated key code corresponds to the character produced by the key when it is pressed without Shift. E.g. in standard US keyboard layout the untranslated key code for the key `=/+` in the upper right corner of the keyboard is 61 which is the ASCII value of `=`.
- For the rest of the keys (i.e. special non-printable keys) it is the same as the normal key code as no translation is used anyhow.

Notice that the first rule applies to all Unicode letters, not just the usual Latin-1 ones. However for non-Latin-1 letters only GetUnicodeKey() can be used to retrieve the key code as GetKeyCode() just returns `WXK_NONE` in this case.

To summarize: you should handle wxEVT_CHAR if you need the translated key and wxEVT_KEY_DOWN if you only need the value of the key itself, independent of the current keyboard state.

Note

Not all key down events may be generated by the user. As an example, wxEVT_KEY_DOWN with = key code can be generated using the standard US keyboard layout but not using the German one because the = key corresponds to Shift-0 key combination in this layout and the key code for it is 0, not =. Because of this you should avoid requiring your users to type key events that might be impossible to enter on their keyboard.

Another difference between key and char events is that another kind of translation is done for the latter ones when the Control key is pressed: char events for ASCII letters in this case carry codes corresponding to the ASCII value of Ctrl-Latter, i.e. 1 for Ctrl-A, 2 for Ctrl-B and so on until 26 for Ctrl-Z. This is convenient for terminal-like applications and can be completely ignored by all the other ones (if you need to handle Ctrl-A it is probably a better idea to use the key event rather than the char one). Notice that currently no translation is done for the presses of [, \,], ^ and _ keys which might be mapped to ASCII values from 27 to 31. Since version 2.9.2, the enum values WXK_CONTROL_A - WXK_CONTROL_Z can be used instead of the non-descriptive constant values 1-26.

Finally, modifier keys only generate key events but no char events at all. The modifiers keys are WXK_SHIFT, WXK_CONTROL, WXK_ALT and various WXK_WINDOWS_XXX from wxKeyCode enum.

Modifier keys events are special in one additional aspect: usually the keyboard state associated with a key press is well defined, e.g. wxKeyboardState::ShiftDown() returns true only if the Shift key was held pressed when the key that generated this event itself was pressed. There is an ambiguity for the key press events for Shift key itself however. By convention, it is considered to be already pressed when it is pressed and already released when it is released. In other words, wxEVT_KEY_DOWN event for the Shift key itself will have wxMOD_SHIFT in GetModifiers() and ShiftDown() will return true while the wxEVT_KEY_UP event for Shift itself will not have wxMOD_SHIFT in its modifiers and ShiftDown() will return false.

Tip: You may discover the key codes and modifiers generated by all the keys on your system interactively by running the Key Event Sample wxWidgets sample and pressing some keys in it.

Note

If a key down (EVT_KEY_DOWN) event is caught and the event handler does not call event.Skip() then the corresponding char event (EVT_CHAR) will not happen. This is by design and enables the programs that handle both types of events to avoid processing the same key twice. As a consequence, if you do not want to suppress the wxEVT_CHAR events for the keys you handle, always call event.Skip() in your wxEVT_KEY_DOWN handler. Not doing may also prevent accelerators defined using this key from working.
If a key is maintained in a pressed state, you will typically get a lot of (automatically generated) key down events but only one key up one at the end when the key is released so it is wrong to assume that there is one up event corresponding to each down one.
For Windows programmers: The key and char events in wxWidgets are similar to but slightly different from Windows WM_KEYDOWN and WM_CHAR events. In particular, Alt-x combination will generate a char event in wxWidgets (unless it is used as an accelerator) and almost all keys, including ones without ASCII equivalents, generate char events too.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxKeyEvent& event)

Event macros:

- EVT_KEY_DOWN(func): Process a wxEVT_KEY_DOWN event (any key has been pressed). If this event is handled and not skipped, wxEVT_CHAR will not be generated at all for this key press (but wxEVT_KEY_UP will be).
- EVT_KEY_UP(func): Process a wxEVT_KEY_UP event (any key has been released).
- EVT_CHAR(func): Process a wxEVT_CHAR event.

- EVT_CHAR_HOOK(func): Process a `wxEVT_CHAR_HOOK` event. Unlike all the other key events, this event is propagated upwards the window hierarchy which allows intercepting it in the parent window of the focused window to which it is sent initially (if there is no focused window, this event is sent to the wxApp global object). It is also generated before any other key events and so gives the parent window an opportunity to modify the keyboard handling of its children, e.g. it is used internally by wxWidgets in some ports to intercept pressing Esc key in any child of a dialog to close the dialog itself when it's pressed. By default, if this event is handled, i.e. the handler doesn't call wxEvent::Skip(), neither `wxEVT_KEY_DOWN` nor `wxEVT_CHAR` events will be generated (although `wxEVT_KEY_UP` still will be), i.e. it replaces the normal key events. However by calling the special DoAllowNextEvent() method you can handle `wxEVT_CHAR_HOOK` and still allow normal events generation. This is something that is rarely useful but can be required if you need to prevent a parent `wxEVT_CHAR_HOOK` handler from running without suppressing the normal key events. Finally notice that this event is not generated when the mouse is captured as it is considered that the window which has the capture should receive all the keyboard events too without allowing its parent wxTopLevelWindow to interfere with their processing.

See also

wxKeyboardState

Library: wxCore

Category: Events

Public Member Functions

- wxKeyEvent (wxEventType keyEventType=wxEVT_NULL)

  Constructor.
- int GetKeyCode () const

  Returns the key code of the key that generated this event.
- bool IsKeyInCategory (int category) const

  Returns true if the key is in the given key category.
- wxUint32 GetRawKeyCode () const

  Returns the raw key code for this event.
- wxUint32 GetRawKeyFlags () const

  Returns the low level key flags for this event.
- wxChar GetUnicodeKey () const

  Returns the Unicode character corresponding to this key event.
- wxCoord GetX () const

  Returns the X position (in client coordinates) of the event.
- wxCoord GetY () const

  Returns the Y position (in client coordinates) of the event.
- void DoAllowNextEvent ()

  Allow normal key events generation.
- bool IsNextEventAllowed () const

  Returns true if DoAllowNextEvent() had been called, false by default.

- wxPoint GetPosition () const

  Obtains the position (in client coordinates) at which the key was pressed.
- void GetPosition (wxCoord *x, wxCoord *y) const

  Obtains the position (in client coordinates) at which the key was pressed.

Additional Inherited Members

21.416.2 Constructor & Destructor Documentation

wxKeyEvent::wxKeyEvent (wxEventType *keyEventType* = wxEVT_NULL)

Constructor.

Currently, the only valid event types are `wxEVT_CHAR` and `wxEVT_CHAR_HOOK`.

21.416.3 Member Function Documentation

void wxKeyEvent::DoAllowNextEvent ( )

Allow normal key events generation.

Can be called from `wxEVT_CHAR_HOOK` handler to indicate that the generation of normal events should *not* be suppressed, as it happens by default when this event is handled.

The intended use of this method is to allow some window object to prevent `wxEVT_CHAR_HOOK` handler in its parent window from running by defining its own handler for this event. Without calling this method, this would result in not generating `wxEVT_KEY_DOWN` nor `wxEVT_CHAR` events at all but by calling it you can ensure that these events would still be generated, even if `wxEVT_CHAR_HOOK` event was handled.

Since

2.9.3

int wxKeyEvent::GetKeyCode ( ) const

Returns the key code of the key that generated this event.

ASCII symbols return normal ASCII values, while events from special keys such as "left cursor arrow" (`WXK_LEFT`) return values outside of the ASCII range. See wxKeyCode for a full list of the virtual key codes.

Note that this method returns a meaningful value only for special non-alphanumeric keys or if the user entered a Latin-1 character (this includes ASCII and the accented letters found in Western European languages but not letters of other alphabets such as e.g. Cyrillic). Otherwise it simply method returns `WXK_NONE` and GetUnicodeKey() should be used to obtain the corresponding Unicode character.

Using GetUnicodeKey() is in general the right thing to do if you are interested in the characters typed by the user, GetKeyCode() should be only used for special keys (for which GetUnicodeKey() returns `WXK_NONE`). To handle both kinds of keys you might write:

```
void MyHandler::OnChar(wxKeyEvent& event)
{
    wxChar uc = event.GetUnicodeKey();
    if ( uc != WXK_NONE )
    {
        // It's a "normal" character. Notice that this includes
        // control characters in 1..31 range, e.g. WXK_RETURN or
        // WXK_BACK, so check for them explicitly.
        if ( uc >= 32 )
        {
            wxLogMessage("You pressed '%c'", uc);
        }
        else
        {
            // It's a control character
            ...
        }
    }
    else // No Unicode equivalent.
    {
        // It's a special key, deal with all the known ones:
        switch ( event.GetKeyCode() )
```

```
        {
            case WXK_LEFT:
            case WXK_RIGHT:
                ... move cursor ...
                break;

            case WXK_F1:
                ... give help ...
                break;
        }
    }
}
```

wxPoint wxKeyEvent::GetPosition ( ) const

Obtains the position (in client coordinates) at which the key was pressed.

Notice that under most platforms this position is simply the current mouse pointer position and has no special relationship to the key event itself.

x and *y* may be NULL if the corresponding coordinate is not needed.

void wxKeyEvent::GetPosition (wxCoord * *x*, wxCoord * *y*) const

Obtains the position (in client coordinates) at which the key was pressed.

Notice that under most platforms this position is simply the current mouse pointer position and has no special relationship to the key event itself.

x and *y* may be NULL if the corresponding coordinate is not needed.

wxUint32 wxKeyEvent::GetRawKeyCode ( ) const

Returns the raw key code for this event.

The flags are platform-dependent and should only be used if the functionality provided by other wxKeyEvent methods is insufficient.

Under MSW, the raw key code is the value of `wParam` parameter of the corresponding message.

Under GTK, the raw key code is the `keyval` field of the corresponding GDK event.

Under OS X, the raw key code is the `keyCode` field of the corresponding NSEvent.

Note

Currently the raw key codes are not supported by all ports, use #ifdef wxHAS_RAW_KEY_CODES to determine if this feature is available.

wxUint32 wxKeyEvent::GetRawKeyFlags ( ) const

Returns the low level key flags for this event.

The flags are platform-dependent and should only be used if the functionality provided by other wxKeyEvent methods is insufficient.

Under MSW, the raw flags are just the value of `lParam` parameter of the corresponding message.

Under GTK, the raw flags contain the `hardware_keycode` field of the corresponding GDK event.

Under OS X, the raw flags contain the modifiers state.

Note

Currently the raw key flags are not supported by all ports, use #ifdef wxHAS_RAW_KEY_CODES to determine if this feature is available.

wxChar wxKeyEvent::GetUnicodeKey ( ) const

Returns the Unicode character corresponding to this key event.

If the key pressed doesn't have any character value (e.g. a cursor key) this method will return `WXK_NONE`. In this case you should use GetKeyCode() to retrieve the value of the key.

This function is only available in Unicode build, i.e. when `wxUSE_UNICODE` is 1.

wxCoord wxKeyEvent::GetX ( ) const

Returns the X position (in client coordinates) of the event.

See also

GetPosition()

wxCoord wxKeyEvent::GetY ( ) const

Returns the Y position (in client coordinates) of the event.

See also

GetPosition()

bool wxKeyEvent::IsKeyInCategory (int *category*) const

Returns true if the key is in the given key category.

Parameters

category	A bitwise combination of named wxKeyCategoryFlags constants.

Since

2.9.1

bool wxKeyEvent::IsNextEventAllowed ( ) const

Returns true if DoAllowNextEvent() had been called, false by default.

This method is used by wxWidgets itself to determine whether the normal key events should be generated after `wxEVT_CHAR_HOOK` processing.

Since

2.9.3

21.417 wxLanguageInfo Struct Reference

```
#include <wx/intl.h>
```

21.417.1 Detailed Description

Encapsulates a wxLanguage identifier together with OS-specific information related to that language.

wxPerl Note: In wxPerl `Wx::LanguageInfo` has only one method:

- Wx::LanguageInfo->new(language, canonicalName, WinLang, WinSubLang, Description)

Public Member Functions

- wxUint32 GetLCID () const

  Return the LCID corresponding to this language.
- wxString GetLocaleName () const

  Return the locale name corresponding to this language usable with `setlocale()` on the current system.

Public Attributes

- int Language

  wxLanguage id.
- wxString CanonicalName

  Canonical name of the language, e.g. `fr_FR`.
- wxString Description

  Human-readable name of the language.
- wxLayoutDirection LayoutDirection

  The layout direction used for this language.

- wxUint32 WinLang

  Win32 language identifiers (LANG_xxxx, SUBLANG_xxxx).
- wxUint32 WinSublang

  Win32 language identifiers (LANG_xxxx, SUBLANG_xxxx).

21.417.2 Member Function Documentation

wxUint32 wxLanguageInfo::GetLCID ( ) const

Return the LCID corresponding to this language.

Availability: only available for the wxMSW port.

wxString wxLanguageInfo::GetLocaleName ( ) const

Return the locale name corresponding to this language usable with `setlocale()` on the current system.

21.417.3 Member Data Documentation

wxString wxLanguageInfo::CanonicalName

Canonical name of the language, e.g. `fr_FR`.

wxString wxLanguageInfo::Description

Human-readable name of the language.

int wxLanguageInfo::Language

wxLanguage id.

It should be greater than `wxLANGUAGE_USER_DEFINED` when defining your own language info structure.

wxLayoutDirection wxLanguageInfo::LayoutDirection

The layout direction used for this language.

wxUint32 wxLanguageInfo::WinLang

Win32 language identifiers (LANG_xxxx, SUBLANG_xxxx).

Availability: only available for the wxMSW port.

wxUint32 wxLanguageInfo::WinSublang

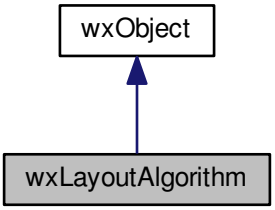
Win32 language identifiers (LANG_xxxx, SUBLANG_xxxx).

Availability: only available for the wxMSW port.

21.418 wxLayoutAlgorithm Class Reference

`#include <wx/laywin.h>`

Inheritance diagram for wxLayoutAlgorithm:



21.418.1 Detailed Description

wxLayoutAlgorithm implements layout of subwindows in MDI or SDI frames.

It sends a wxCalculateLayoutEvent event to children of the frame, asking them for information about their size. For MDI parent frames, the algorithm allocates the remaining space to the MDI client window (which contains the MDI child frames).

For SDI (normal) frames, a 'main' window is specified as taking up the remaining space.

Because the event system is used, this technique can be applied to any windows, which are not necessarily 'aware' of the layout classes (no virtual functions in wxWindow refer to wxLayoutAlgorithm or its events). However, you may wish to use wxSashLayoutWindow for your subwindows since this class provides handlers for the required

events, and accessors to specify the desired size of the window. The sash behaviour in the base class can be used, optionally, to make the windows user-resizable.

wxLayoutAlgorithm is typically used in IDE (integrated development environment) applications, where there are several resizable windows in addition to the MDI client window, or other primary editing window. Resizable windows might include toolbars, a project window, and a window for displaying error and warning messages.

When a window receives an OnCalculateLayout event, it should call SetRect in the given event object, to be the old supplied rectangle minus whatever space the window takes up. It should also set its own size accordingly. wxSash↩ LayoutWindow::OnCalculateLayout generates an OnQueryLayoutInfo event which it sends to itself to determine the orientation, alignment and size of the window, which it gets from internal member variables set by the application.

The algorithm works by starting off with a rectangle equal to the whole frame client area. It iterates through the frame children, generating wxLayoutAlgorithm::OnCalculateLayout events which subtract the window size and return the remaining rectangle for the next window to process. It is assumed (by wxSashLayoutWindow::OnCalculateLayout) that a window stretches the full dimension of the frame client, according to the orientation it specifies. For example, a horizontal window will stretch the full width of the remaining portion of the frame client area. In the other orientation, the window will be fixed to whatever size was specified by wxLayoutAlgorithm::OnQueryLayoutInfo. An alignment setting will make the window 'stick' to the left, top, right or bottom of the remaining client area. This scheme implies that order of window creation is important. Say you wish to have an extra toolbar at the top of the frame, a project window to the left of the MDI client window, and an output window above the status bar. You should therefore create the windows in this order: toolbar, output window, project window. This ensures that the toolbar and output window take up space at the top and bottom, and then the remaining height in-between is used for the project window.

wxLayoutAlgorithm is quite independent of the way in which wxLayoutAlgorithm::OnCalculateLayout chooses to interpret a window's size and alignment. Therefore you could implement a different window class with a new wx↩ LayoutAlgorithm::OnCalculateLayout event handler, that has a more sophisticated way of laying out the windows. It might allow specification of whether stretching occurs in the specified orientation, for example, rather than always assuming stretching. (This could, and probably should, be added to the existing implementation).

Note

wxLayoutAlgorithm has nothing to do with wxLayoutConstraints. It is an alternative way of specifying layouts for which the normal constraint system is unsuitable.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxQueryLayoutInfoEvent& event) or void handlerFuncName(wxCalculateLayout↩ Event& event)

Event macros for events emitted by this class:

- EVT_QUERY_LAYOUT_INFO(func): Process a `wxEVT_QUERY_LAYOUT_INFO` event, to get size, orientation and alignment from a window. See wxQueryLayoutInfoEvent.
- EVT_CALCULATE_LAYOUT(func): Process a `wxEVT_CALCULATE_LAYOUT` event, which asks the window to take a 'bite' out of a rectangle provided by the algorithm. See wxCalculateLayoutEvent.

Note that the algorithm object does not respond to events, but itself generates the previous events in order to calculate window sizes.

Library: wxAdvanced

Category: Window Layout

See also

wxSashEvent, wxSashLayoutWindow, Events and Event Handling

Public Member Functions

- wxLayoutAlgorithm ()

  Default constructor.
- virtual ~wxLayoutAlgorithm ()

  Destructor.
- bool LayoutFrame (wxFrame *frame, wxWindow *mainWindow=NULL)

  Lays out the children of a normal frame.
- bool LayoutMDIFrame (wxMDIParentFrame *frame, wxRect *rect=NULL)

  Lays out the children of an MDI parent frame.
- bool LayoutWindow (wxWindow *parent, wxWindow *mainWindow=NULL)

  Lays out the children of a normal frame or other window.

Additional Inherited Members

21.418.2 Constructor & Destructor Documentation

wxLayoutAlgorithm::wxLayoutAlgorithm ( )

Default constructor.

virtual wxLayoutAlgorithm::~wxLayoutAlgorithm ( ) [virtual]

Destructor.

21.418.3 Member Function Documentation

bool wxLayoutAlgorithm::LayoutFrame (wxFrame * *frame*, wxWindow * *mainWindow* = NULL)

Lays out the children of a normal frame.

mainWindow is set to occupy the remaining space. This function simply calls LayoutWindow().

bool wxLayoutAlgorithm::LayoutMDIFrame (wxMDIParentFrame * *frame*, wxRect * *rect* = NULL)

Lays out the children of an MDI parent frame.

If *rect* is non-NULL, the given rectangle will be used as a starting point instead of the frame's client area. The MDI client window is set to occupy the remaining space.

bool wxLayoutAlgorithm::LayoutWindow (wxWindow * *parent*, wxWindow * *mainWindow* = NULL)

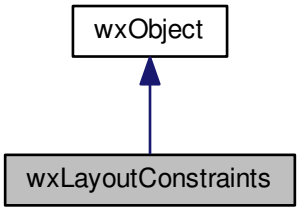
Lays out the children of a normal frame or other window.

mainWindow is set to occupy the remaining space. If this is not specified, then the last window that responds to a calculate layout event in query mode will get the remaining space (that is, a non-query OnCalculateLayout event will not be sent to this window and the window will be set to the remaining size).

21.419 wxLayoutConstraints Class Reference

```
#include <wx/layout.h>
```

Inheritance diagram for wxLayoutConstraints:



Public Member Functions

- wxLayoutConstraints ()
- virtual ~wxLayoutConstraints ()
- bool SatisfyConstraints (wxWindow *win, int *noChanges)
- bool AreSatisfied () const

Public Attributes

- wxIndividualLayoutConstraint left
- wxIndividualLayoutConstraint top
- wxIndividualLayoutConstraint right
- wxIndividualLayoutConstraint bottom
- wxIndividualLayoutConstraint width
- wxIndividualLayoutConstraint height
- wxIndividualLayoutConstraint centreX
- wxIndividualLayoutConstraint centreY

Additional Inherited Members

21.419.1 Constructor & Destructor Documentation

wxLayoutConstraints::wxLayoutConstraints ( )

virtual wxLayoutConstraints::~wxLayoutConstraints ( ) `[virtual]`

21.419.2 Member Function Documentation

bool wxLayoutConstraints::AreSatisfied ( ) const

bool wxLayoutConstraints::SatisfyConstraints (wxWindow * *win*, int * *noChanges*)

21.419.3 Member Data Documentation

wxIndividualLayoutConstraint wxLayoutConstraints::bottom

wxIndividualLayoutConstraint wxLayoutConstraints::centreX

wxIndividualLayoutConstraint wxLayoutConstraints::centreY

wxIndividualLayoutConstraint wxLayoutConstraints::height

wxIndividualLayoutConstraint wxLayoutConstraints::left

wxIndividualLayoutConstraint wxLayoutConstraints::right

wxIndividualLayoutConstraint wxLayoutConstraints::top

wxIndividualLayoutConstraint wxLayoutConstraints::width

21.420 wxLinuxDistributionInfo Struct Reference

`#include <wx/platinfo.h>`

21.420.1 Detailed Description

A structure containing information about a Linux distribution as returned by the `lsb_release` utility.

See wxGetLinuxDistributionInfo() or wxPlatformInfo::GetLinuxDistributionInfo() for more info.

Public Member Functions

- bool operator== (const wxLinuxDistributionInfo &ldi) const
- bool operator!= (const wxLinuxDistributionInfo &ldi) const

Public Attributes

- wxString Id

  The id of the distribution; e.g. "Ubuntu".
- wxString Release

  The version of the distribution; e.g. "9.04".
- wxString CodeName

  The code name of the distribution; e.g. "jaunty".
- wxString Description

  The description of the distribution; e.g. "Ubuntu 9.04".

21.420.2 Member Function Documentation

bool wxLinuxDistributionInfo::operator!= (const wxLinuxDistributionInfo & *ldi*) const

bool wxLinuxDistributionInfo::operator== (const wxLinuxDistributionInfo & *ldi*) const

21.420.3 Member Data Documentation

wxString wxLinuxDistributionInfo::CodeName

The code name of the distribution; e.g. "jaunty".

wxString wxLinuxDistributionInfo::Description

The description of the distribution; e.g. "Ubuntu 9.04".

wxString wxLinuxDistributionInfo::Id

The id of the distribution; e.g. "Ubuntu".

wxString wxLinuxDistributionInfo::Release

The version of the distribution; e.g. "9.04".

21.421 wxList< T > Class Template Reference

`#include <wx/list.h>`

21.421.1 Detailed Description

template<typename T>class wxList< T >

The wxList<T> class provides linked list functionality.

This class has been rewritten to be type safe and to provide the full API of the STL std::list container and should be used like it. The exception is that wxList<T> actually stores pointers and therefore its iterators return pointers and not references to the actual objects in the list (see example below) and *value_type* is defined as *T**. wxList<T> destroys an object after removing it only if wxList<T>::DeleteContents has been called.

wxList<T> is not a real template and it requires that you declare and define each wxList<T> class in your program. This is done with *WX_DECLARE_LIST* and *WX_DEFINE_LIST* macros (see example). We hope that we'll be able to provide a proper template class providing both the STL `std::list` and the old wxList API in the future.

Please refer to the STL `std::list` documentation (see http://www.cppreference.com/wiki/stl/list/start) for further information on how to use the class. Below we documented both the supported STL and the legacy API that originated from the old wxList class and which can still be used alternatively for the same class.

Note that if you compile wxWidgets in STL mode (`wxUSE_STL` defined as 1) then wxList<T> will actually derive from `std::list` and just add a legacy compatibility layer for the old wxList class.

```
// this part might be in a header or source (.cpp) file
class MyListElement
{
    ... // whatever
};

// this macro declares and partly implements MyList class
WX_DECLARE_LIST(MyListElement, MyList);

...

// the only requirement for the rest is to be AFTER the full declaration of
// MyListElement (for WX_DECLARE_LIST forward declaration is enough), but
// usually it will be found in the source file and not in the header

#include <wx/listimpl.cpp>
WX_DEFINE_LIST(MyList);


MyList list;
MyListElement element;
list.Append(&element);     // ok
list.Append(17);           // error: incorrect type

// let's iterate over the list in STL syntax
MyList::iterator iter;
for (iter = list.begin(); iter != list.end(); ++iter)
```

```
{
    MyListElement *current = *iter;

    ...process the current element...
}

// the same with the legacy API from the old wxList class
MyList::compatibility_iterator node = list.GetFirst();
while (node)
{
    MyListElement *current = node->GetData();

    ...process the current element...

    node = node->GetNext();
}
```

For compatibility with previous versions wxList and wxStringList classes are still defined, but their usage is deprecated and they will disappear in the future versions completely. The use of the latter is especially discouraged as it is not only unsafe but is also much less efficient than wxArrayString class.

Template Parameters

T	The type stored in the wxList nodes.

Library: wxBase

Category: Containers

See also

wxArray<T>, wxVector<T>, wxNode<T>

Public Member Functions

- wxList ()

  Default constructor.
- wxList (size_t count, T *elements[])

  Constructor which initialized the list with an array of count elements.
- ~wxList ()

  Destroys the list, but does not delete the objects stored in the list unless you called DeleteContents(true).
- wxList< T >::compatibility_iterator Append (T *object)

  Appends the pointer to object to the list.
- void Clear ()

  Clears the list.
- void DeleteContents (bool destroy)

  If destroy is true, instructs the list to call delete on objects stored in the list whenever they are removed.
- bool DeleteNode (const compatibility_iterator &iter)

  Deletes the given element referred to by iter from the list if iter is a valid iterator.
- bool DeleteObject (T *object)

  Finds the given object and removes it from the list, returning true if successful.
- void Erase (const compatibility_iterator &iter)

  Removes element referred to be iter.
- wxList< T >::compatibility_iterator Find (T *object) const

  Returns the iterator referring to object or NULL if none found.
- size_t GetCount () const

  Returns the number of elements in the list.
- wxList< T >::compatibility_iterator GetFirst () const

Returns the first iterator in the list (NULL if the list is empty).

- wxList< T >::compatibility_iterator GetLast () const

  Returns the last iterator in the list (NULL if the list is empty).

- int IndexOf (T *obj) const

  Returns the index of obj within the list or `wxNOT_FOUND` if obj is not found in the list.

- wxList< T >::compatibility_iterator Insert (T *object)

  Inserts object at the beginning of the list.

- wxList< T >::compatibility_iterator Insert (size_t position, T *object)

  Inserts object at position.

- wxList< T >::compatibility_iterator Insert (compatibility_iterator iter, T *object)

  Inserts object before the object referred to be iter.

- bool IsEmpty () const

  Returns true if the list is empty, false otherwise.

- wxList< T >::compatibility_iterator Item (size_t index) const

  Returns the iterator referring to the object at the given index in the list.

- bool Member (T *object) const

  Check if the object is present in the list.

- wxList< T >::compatibility_iterator Nth (int n) const
- int Number () const
- void Sort (wxSortCompareFunction compfunc)

  Allows the sorting of arbitrary lists by giving a function to compare two list elements.

- void assign (const_iterator first, const const_iterator &last)

  Clears the list and item from first to last from another list to it.

- void assign (size_type n, const_reference v=value_type())

  Clears the list and adds n items with value v to it.

- reference back ()

  Returns the last item of the list.

- const_reference back () const

  Returns the last item of the list as a const reference.

- iterator begin ()

  Returns an iterator pointing to the beginning of the list.

- const_iterator begin () const

  Returns a const iterator pointing to the beginning of the list.

- void clear ()

  Removes all items from the list.

- bool empty () const

  Returns true if the list is empty.

- const_iterator end () const

  Returns a const iterator pointing at the end of the list.

- iterator end () const

  Returns a iterator pointing at the end of the list.

- iterator erase (const iterator &it)

  Erases the given item.

- iterator erase (const iterator &first, const iterator &last)

  Erases the items from first to last.

- reference front () const

  Returns the first item in the list.

- const_reference front () const

  Returns the first item in the list as a const reference.

- iterator insert (const iterator &it)

  Inserts an item at the head of the list.

- void insert (const iterator &it, size_type n)

  Inserts an item at the given position.
- void insert (const iterator &it, const_iterator first, const const_iterator &last)

  Inserts several items at the given position.
- size_type max_size () const

  Returns the largest possible size of the list.
- void pop_back ()

  Removes the last item from the list.
- void pop_front ()

  Removes the first item from the list.
- void push_back (const_reference v=value_type())

  Adds an item to end of the list.
- void push_front (const_reference v=value_type())

  Adds an item to the front of the list.
- reverse_iterator rbegin ()

  Returns a reverse iterator pointing to the beginning of the reversed list.
- const_reverse_iterator rbegin () const

  Returns a const reverse iterator pointing to the beginning of the reversed list.
- void remove (const_reference v)

  Removes an item from the list.
- reverse_iterator rend ()

  Returns a reverse iterator pointing to the end of the reversed list.
- const_reverse_iterator rend () const

  Returns a const reverse iterator pointing to the end of the reversed list.
- void resize (size_type n, value_type v=value_type())

  Resizes the list.
- void reverse ()

  Reverses the list.
- size_type size () const

  Returns the size of the list.
- wxVector< T > AsVector () const

  Returns a wxVector holding the list elements.

21.421.2 Constructor & Destructor Documentation

template<typename T > wxList< T >::wxList ( )

Default constructor.

template<typename T > wxList< T >::wxList (size_t *count*, T * *elements[]*)

Constructor which initialized the list with an array of *count* elements.

template<typename T > wxList< T >::~wxList ( )

Destroys the list, but does not delete the objects stored in the list unless you called DeleteContents(true).

21.421.3 Member Function Documentation

template<typename T > wxList<T>::compatibility_iterator wxList< T >::Append (T * *object*)

Appends the pointer to *object* to the list.

template<typename T > void wxList< T >::assign (const_iterator *first,* const const_iterator & *last*)

Clears the list and item from *first* to *last* from another list to it.

template<typename T > void wxList< T >::assign (size_type *n,* const_reference *v =* `value_type()`)

Clears the list and adds *n* items with value *v* to it.

template<typename T > wxVector<T> wxList< T >::AsVector ( ) const

Returns a wxVector holding the list elements.

Since

2.9.5

template<typename T > reference wxList< T >::back ( )

Returns the last item of the list.

template<typename T > const_reference wxList< T >::back ( ) const

Returns the last item of the list as a const reference.

template<typename T > iterator wxList< T >::begin ( )

Returns an iterator pointing to the beginning of the list.

template<typename T > const_iterator wxList< T >::begin ( ) const

Returns a const iterator pointing to the beginning of the list.

template<typename T > void wxList< T >::Clear ( )

Clears the list.

Deletes the actual objects if DeleteContents(true) was called previously.

template<typename T > void wxList< T >::clear ( )

Removes all items from the list.

template<typename T > void wxList< T >::DeleteContents (bool *destroy*)

If *destroy* is true, instructs the list to call *delete* on objects stored in the list whenever they are removed.
The default is false.

template<typename T > bool wxList< T >::DeleteNode (const compatibility_iterator & *iter*)

Deletes the given element referred to by *iter* from the list if *iter* is a valid iterator.
Returns true if successful.
Deletes the actual object if DeleteContents(true) was called previously.

template<typename T > bool wxList< T >::DeleteObject (T * *object*)

Finds the given *object* and removes it from the list, returning true if successful.
Deletes *object* if DeleteContents(true) was called previously.

template<typename T > bool wxList< T >::empty ( ) const

Returns true if the list is empty.

template<typename T > const_iterator wxList< T >::end ( ) const

Returns a const iterator pointing at the end of the list.

template<typename T > iterator wxList< T >::end ( ) const

Returns a iterator pointing at the end of the list.

template<typename T > void wxList< T >::Erase (const compatibility_iterator & *iter*)

Removes element referred to be *iter*.
Deletes the actual object if DeleteContents(true) was called previously.

template<typename T > iterator wxList< T >::erase (const iterator & *it*)

Erases the given item.

template<typename T > iterator wxList< T >::erase (const iterator & *first,* const iterator & *last*)

Erases the items from *first* to *last*.

template<typename T > wxList<T>::compatibility_iterator wxList< T >::Find (T * *object*) const

Returns the iterator referring to *object* or NULL if none found.

template<typename T > reference wxList< T >::front ( ) const

Returns the first item in the list.

template<typename T> const_reference wxList< T >::front ( ) const

Returns the first item in the list as a const reference.

template<typename T> size_t wxList< T >::GetCount ( ) const

Returns the number of elements in the list.

template<typename T> wxList<T>::compatibility_iterator wxList< T >::GetFirst ( ) const

Returns the first iterator in the list (NULL if the list is empty).

template<typename T> wxList<T>::compatibility_iterator wxList< T >::GetLast ( ) const

Returns the last iterator in the list (NULL if the list is empty).

template<typename T> int wxList< T >::IndexOf (T * *obj*) const

Returns the index of *obj* within the list or `wxNOT_FOUND` if *obj* is not found in the list.

template<typename T> wxList<T>::compatibility_iterator wxList< T >::Insert (T * *object*)

Inserts *object* at the beginning of the list.

template<typename T> wxList<T>::compatibility_iterator wxList< T >::Insert (size_t *position,* T * *object*)

Inserts *object* at *position*.

template<typename T> wxList<T>::compatibility_iterator wxList< T >::Insert (compatibility_iterator *iter,* T * *object*)

Inserts *object* before the object referred to be *iter*.

template<typename T> iterator wxList< T >::insert (const iterator & *it*)

Inserts an item at the head of the list.

template<typename T> void wxList< T >::insert (const iterator & *it,* size_type *n*)

Inserts an item at the given position.

template<typename T> void wxList< T >::insert (const iterator & *it,* const_iterator *first,* const const_iterator & *last*)

Inserts several items at the given position.

template<typename T> bool wxList< T >::IsEmpty ( ) const

Returns true if the list is empty, false otherwise.

template<typename T > wxList<T>::compatibility_iterator wxList< T >::Item (size_t *index*) const

Returns the iterator referring to the object at the given *index* in the list.

template<typename T > size_type wxList< T >::max_size ( ) const

Returns the largest possible size of the list.

template<typename T > bool wxList< T >::Member (T * *object*) const

Check if the object is present in the list.

See also

Find()

template<typename T > wxList<T>::compatibility_iterator wxList< T >::Nth (int *n*) const

Deprecated This function is deprecated, use Item() instead.

template<typename T > int wxList< T >::Number ( ) const

Deprecated This function is deprecated, use wxList::GetCount instead. Returns the number of elements in the list.

template<typename T > void wxList< T >::pop_back ( )

Removes the last item from the list.

template<typename T > void wxList< T >::pop_front ( )

Removes the first item from the list.

template<typename T > void wxList< T >::push_back (const_reference *v* = `value_type()`)

Adds an item to end of the list.

template<typename T > void wxList< T >::push_front (const_reference *v* = `value_type()`)

Adds an item to the front of the list.

template<typename T > reverse_iterator wxList< T >::rbegin ( )

Returns a reverse iterator pointing to the beginning of the reversed list.

template<typename T > const_reverse_iterator wxList< T >::rbegin ( ) const

Returns a const reverse iterator pointing to the beginning of the reversed list.

template<typename T > void wxList< T >::remove (const_reference *v*)

Removes an item from the list.

template<typename T > reverse_iterator wxList< T >::rend ( )

Returns a reverse iterator pointing to the end of the reversed list.

template<typename T > const_reverse_iterator wxList< T >::rend ( ) const

Returns a const reverse iterator pointing to the end of the reversed list.

template<typename T > void wxList< T >::resize (size_type *n*, value_type *v* = `value_type()`)

Resizes the list.

If the list is longer than *n*, then items are removed until the list becomes long *n*. If the list is shorter than *n* items with the value *v* are appended to the list until the list becomes long *n*.

template<typename T > void wxList< T >::reverse ( )

Reverses the list.

template<typename T > size_type wxList< T >::size ( ) const

Returns the size of the list.

template<typename T > void wxList< T >::Sort (wxSortCompareFunction *compfunc*)

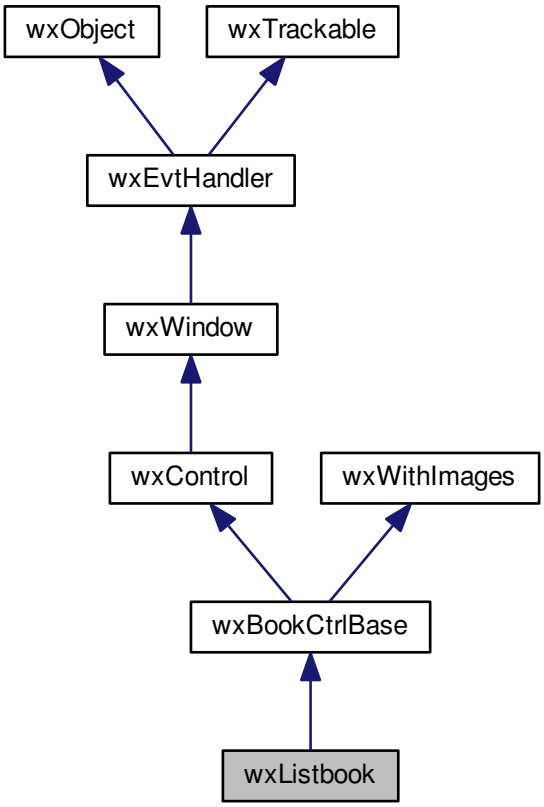
Allows the sorting of arbitrary lists by giving a function to compare two list elements.

We use the system **qsort** function for the actual sorting process.

21.422 wxListbook Class Reference

`#include <wx/listbook.h>`

Inheritance diagram for wxListbook:



21.422.1 Detailed Description

wxListbook is a class similar to wxNotebook but which uses a wxListCtrl to show the labels instead of the tabs.

The underlying wxListCtrl displays page labels in a one-column report view by default. Calling wxBookCtrl::Set↩ ImageList will implicitly switch the control to use an icon view.

For usage documentation of this class, please refer to the base abstract class wxBookCtrl. You can also use the Notebook Sample to see wxListbook in action.

Styles

This class supports the following styles:

- wxLB_DEFAULT: Choose the default location for the labels depending on the current platform (left everywhere except Mac where it is top).
- wxLB_TOP: Place labels above the page area.
- wxLB_LEFT: Place labels on the left side.
- wxLB_RIGHT: Place labels on the right side.
- wxLB_BOTTOM: Place labels below the page area.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxBookCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_LISTBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_LISTBOOK_PAGE_CHANGED` event.
- EVT_LISTBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_LISTBOOK_PAGE_CHANGING` event. This event can be vetoed.

Library: wxCore

Category: Book Controls

See also

wxBookCtrl, wxNotebook, Notebook Sample

Public Member Functions

- wxListbook ()

  Default ctor.
- wxListbook (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Constructs a listbook control.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Create the list book control that has already been constructed with the default constructor.
- wxListView * GetListView () const

  Returns the wxListView associated with the control.

Additional Inherited Members

21.422.2 Constructor & Destructor Documentation

wxListbook::wxListbook ( )

Default ctor.

wxListbook::wxListbook (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Constructs a listbook control.

21.422.3 Member Function Documentation

bool wxListbook::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Create the list book control that has already been constructed with the default constructor.

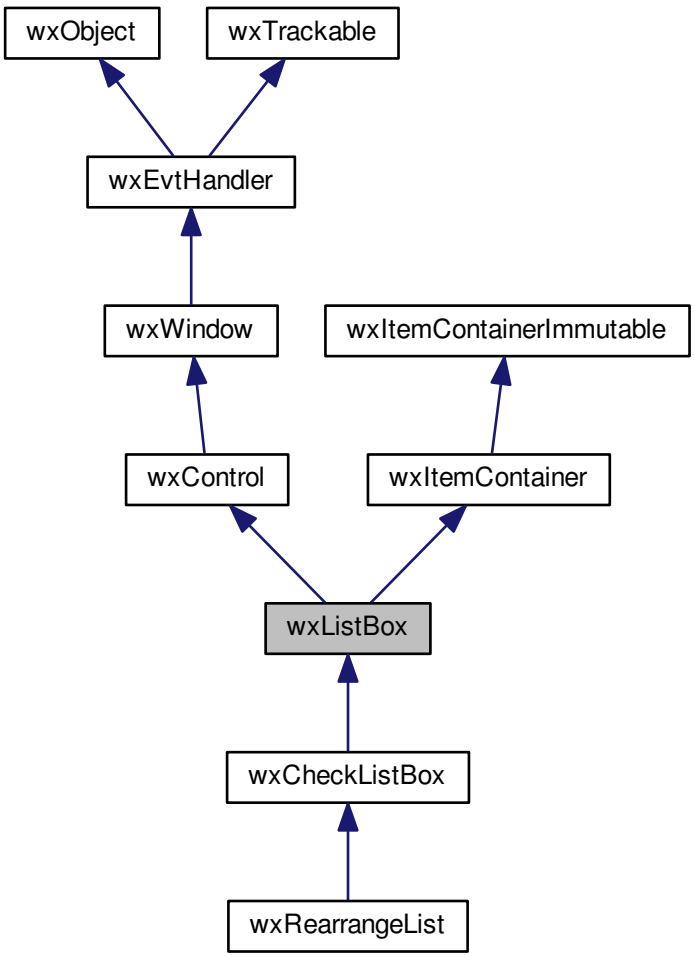
wxListView* wxListbook::GetListView ( ) const

Returns the wxListView associated with the control.

21.423 wxListBox Class Reference

`#include <wx/listbox.h>`

Inheritance diagram for wxListBox:



21.423.1 Detailed Description

A listbox is used to select one or more of a list of strings.

The strings are displayed in a scrolling box, with the selected string(s) marked in reverse video. A listbox can be single selection (if an item is selected, the previous selection is removed) or multiple selection (clicking an item toggles the item on or off independently of other selections).

List box elements are numbered from zero and while the maximal number of elements is unlimited, it is usually

better to use a virtual control, not requiring to add all the items to it at once, such as wxDataViewCtrl or wxListCtrl with `wxLC_VIRTUAL` style, once more than a few hundreds items need to be displayed because this control is not optimized, neither from performance nor from user interface point of view, for large number of items.

Notice that currently `TAB` characters in list box items text are not handled consistently under all platforms, so they should be replaced by spaces to display strings properly everywhere. The list box doesn't support any other control characters at all.

Styles

This class supports the following styles:

- wxLB_SINGLE: Single-selection list.
- wxLB_MULTIPLE: Multiple-selection list: the user can toggle multiple items on and off. This is the same as wxLB_EXTENDED in wxGTK2 port.
- wxLB_EXTENDED: Extended-selection list: the user can extend the selection by using `SHIFT` or `CTRL` keys together with the cursor movement keys or the mouse.
- wxLB_HSCROLL: Create horizontal scrollbar if contents are too wide (Windows only).
- wxLB_ALWAYS_SB: Always show a vertical scrollbar.
- wxLB_NEEDED_SB: Only create a vertical scrollbar if needed.
- wxLB_NO_SB: Don't create vertical scrollbar (wxMSW only).
- wxLB_SORT: The listbox contents are sorted in alphabetical order.

Note that `wxLB_SINGLE`, `wxLB_MULTIPLE` and `wxLB_EXTENDED` styles are mutually exclusive and you can specify at most one of them (single selection is the default). See also Window Styles.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_LISTBOX(id, func): Process a `wxEVT_LISTBOX` event, when an item on the list is selected or the selection changes.
- EVT_LISTBOX_DCLICK(id, func): Process a `wxEVT_LISTBOX_DCLICK` event, when the listbox is double-clicked.

Library: wxCore

Category: Controls

See also

wxEditableListBox, wxChoice, wxComboBox, wxListCtrl, wxCommandEvent

Public Member Functions

- wxListBox ()

  Default constructor.
- wxListBox (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListBoxNameStr)

  Constructor, creating and showing a list box.
- wxListBox (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListBoxNameStr)

  Constructor, creating and showing a list box.
- virtual ~wxListBox ()

  Destructor, destroying the list box.
- void Deselect (int n)

  Deselects an item in the list box.
- virtual void SetSelection (int n)

  Sets the selection to the given item n or removes the selection entirely if n == wxNOT_FOUND.
- virtual int GetSelection () const

  Returns the index of the selected item or wxNOT_FOUND if no item is selected.
- virtual bool SetStringSelection (const wxString &s, bool select)
- virtual bool SetStringSelection (const wxString &s)
- virtual int GetSelections (wxArrayInt &selections) const

  Fill an array of ints with the positions of the currently selected items.
- int HitTest (const wxPoint &point) const

  Returns the item located at point, or wxNOT_FOUND if there is no item located at point.
- int HitTest (int x, int y) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void InsertItems (unsigned int nItems, const wxString *items, unsigned int pos)

  Insert the given number of strings before the specified position.
- void InsertItems (const wxArrayString &items, unsigned int pos)

  Insert the given number of strings before the specified position.
- virtual bool IsSelected (int n) const

  Determines whether an item is selected.
- void SetFirstItem (int n)

  Set the specified item to be the first visible item.
- void SetFirstItem (const wxString &string)

  Set the specified item to be the first visible item.
- virtual void EnsureVisible (int n)

  Ensure that the item with the given index is currently shown.
- virtual bool IsSorted () const

  Return true if the listbox has wxLB_SORT style.
- virtual unsigned int GetCount () const

  Returns the number of items in the control.
- virtual wxString GetString (unsigned int n) const

  Returns the label of the item with the given index.
- virtual void SetString (unsigned int n, const wxString &s)

  Sets the label for the given item.
- virtual int FindString (const wxString &s, bool bCase=false) const

  Finds an item whose label matches the given string.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListBoxNameStr)

  Creates the listbox for two-step construction.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListBoxNameStr)

  Creates the listbox for two-step construction.

Additional Inherited Members

21.423.2 Constructor & Destructor Documentation

wxListBox::wxListBox ( )

Default constructor.

wxListBox::wxListBox (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxListBoxNameStr)

Constructor, creating and showing a list box.

Parameters

parent	The parent window.
id	The ID of this control. A value of wxID_ANY indicates a default value.
pos	The initial position. If wxDefaultPosition is specified then a default position is chosen.
size	The initial size. If wxDefaultSize is specified then the window is sized appropriately.
n	Number of strings with which to initialise the control.
choices	The strings to use to initialize the control.
style	Window style. See wxListBox.
validator	The validator for this control.
name	The name of this class.

wxPerl Note: Not supported by wxPerl.

wxListBox::wxListBox (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxListBoxNameStr)

Constructor, creating and showing a list box.

See the other wxListBox() constructor; the only difference is that this overload takes a wxArrayString instead of a pointer to an array of wxString.

wxPerl Note: Use an array reference for the *choices* parameter.

virtual wxListBox::~wxListBox ( ) [virtual]

Destructor, destroying the list box.

21.423.3 Member Function Documentation

bool wxListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxListBoxNameStr)

Creates the listbox for two-step construction.

See wxListBox() for further details.

bool wxListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxListBoxNameStr)

Creates the listbox for two-step construction.

See wxListBox() for further details.

void wxListBox::Deselect (int *n*)

Deselects an item in the list box.

Parameters

n	The zero-based item to deselect.

Remarks

This applies to multiple selection listboxes only.

virtual void wxListBox::EnsureVisible (int *n*) [virtual]

Ensure that the item with the given index is currently shown.

This method scrolls the listbox only if necessary and doesn't do anything if this item is already shown, unlike SetFirstItem().

virtual int wxListBox::FindString (const wxString & *string*, bool *caseSensitive* = false) const [virtual]

Finds an item whose label matches the given string.

Parameters

string	String to find.
caseSensitive	Whether search is case sensitive (default is not).

Returns

The zero-based position of the item, or wxNOT_FOUND if the string was not found.

Reimplemented from wxItemContainerImmutable.

virtual unsigned int wxListBox::GetCount ( ) const [virtual]

Returns the number of items in the control.

See also

IsEmpty()

Implements wxItemContainerImmutable.

virtual int wxListBox::GetSelection ( ) const `[virtual]`

Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

Returns

The position of the current selection.

Remarks

This method can be used with single selection list boxes only, you should use wxListBox::GetSelections() for the list boxes with wxLB_MULTIPLE style.

See also

SetSelection(), GetStringSelection()

Implements wxItemContainerImmutable.

virtual int wxListBox::GetSelections (wxArrayInt & *selections*) const `[virtual]`

Fill an array of ints with the positions of the currently selected items.

Parameters

selections	A reference to an wxArrayInt instance that is used to store the result of the query.

Returns

The number of selections.

Remarks

Use this with a multiple selection listbox.

wxPerl Note: In wxPerl this method takes no parameters and return the selected items as a list.

See also

wxControlWithItems::GetSelection, wxControlWithItems::GetStringSelection, wxControlWithItems::SetSelection

virtual wxString wxListBox::GetString (unsigned int *n*) const `[virtual]`

Returns the label of the item with the given index.

Parameters

n	The zero-based index.

Returns

The label of the item or an empty string if the position was invalid.

Implements wxItemContainerImmutable.

int wxListBox::HitTest (const wxPoint & *point*) const

Returns the item located at *point*, or `wxNOT_FOUND` if there is no item located at *point*.

It is currently implemented for wxMSW, wxMac and wxGTK2 ports.

Parameters

point	Point of item (in client coordinates) to obtain

Returns

Item located at point, or wxNOT_FOUND if unimplemented or the item does not exist.

Since

2.7.0

int wxListBox::HitTest (int *x*, int *y*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxListBox::InsertItems (unsigned int *nItems*, const wxString * *items*, unsigned int *pos*)

Insert the given number of strings before the specified position.

Parameters

nItems	Number of items in the array items
items	Labels of items to be inserted
pos	Position before which to insert the items: if pos is 0 the items will be inserted in the beginning of the listbox

wxPerl Note: Not supported by wxPerl.

void wxListBox::InsertItems (const wxArrayString & *items*, unsigned int *pos*)

Insert the given number of strings before the specified position.

Parameters

items	Labels of items to be inserted
pos	Position before which to insert the items: if pos is 0 the items will be inserted in the beginning of the listbox

wxPerl Note: Use an array reference for the *items* parameter.

virtual bool wxListBox::IsSelected (int *n*) const [virtual]

Determines whether an item is selected.

Parameters

n	The zero-based item index.

Returns

true if the given item is selected, false otherwise.

virtual bool wxListBox::IsSorted ( ) const [virtual]

Return true if the listbox has wxLB_SORT style.

This method is mostly meant for internal use only.

void wxListBox::SetFirstItem (int *n*)

Set the specified item to be the first visible item.

Parameters

n	The zero-based item index that should be visible.

void wxListBox::SetFirstItem (const wxString & *string*)

Set the specified item to be the first visible item.

Parameters

string	The string that should be visible.

virtual void wxListBox::SetSelection (int *n*) [virtual]

Sets the selection to the given item *n* or removes the selection entirely if *n* == `wxNOT_FOUND`.

Note that this does not cause any command events to be emitted nor does it deselect any other items in the controls which support multiple selections.

Parameters

n	The string position to select, starting from zero.

See also

SetString(), SetStringSelection()

Implements wxItemContainerImmutable.

virtual void wxListBox::SetString (unsigned int *n,* const wxString & *string*) [virtual]

Sets the label for the given item.

Parameters

n	The zero-based item index.
string	The label to set.

Implements wxItemContainerImmutable.

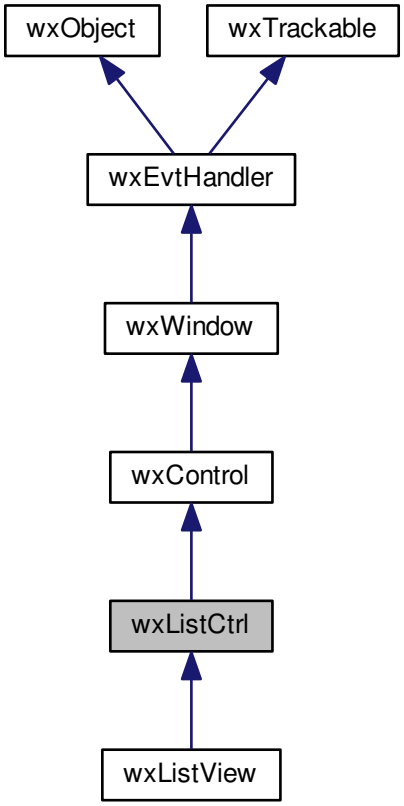
virtual bool wxListBox::SetStringSelection (const wxString & *s,* bool *select*) [virtual]

virtual bool wxListBox::SetStringSelection (const wxString & *s*) [virtual]

21.424 wxListCtrl Class Reference

```
#include <wx/listctrl.h>
```

Inheritance diagram for wxListCtrl:



21.424.1 Detailed Description

A list control presents lists in a number of formats: list view, report view, icon view and small icon view.

In any case, elements are numbered from zero. For all these modes, the items are stored in the control and must be added to it using wxListCtrl::InsertItem method.

A special case of report view quite different from the other modes of the list control is a virtual control in which the items data (including text, images and attributes) is managed by the main program and is requested by the control itself only when needed which allows to have controls with millions of items without consuming much memory. To use virtual list control you must use wxListCtrl::SetItemCount first and override at least wxListCtrl::OnGetItemText (and optionally wxListCtrl::OnGetItemImage or wxListCtrl::OnGetItemColumnImage and wxListCtrl::OnGetItemAttr) to return the information about the items when the control requests it.

Virtual list control can be used as a normal one except that no operations which can take time proportional to the number of items in the control happen – this is required to allow having a practically infinite number of items. For example, in a multiple selection virtual list control, the selections won't be sent when many items are selected at once because this could mean iterating over all the items.

Using many of wxListCtrl features is shown in the corresponding sample.

To intercept events from a list control, use the event table macros described in wxListEvent.

wxMac Note: Starting with wxWidgets 2.8, wxListCtrl uses a native implementation for report mode, and uses a generic implementation for other modes. You can use the generic implementation for report mode as well by setting

the `mac.listctrl.always_use_generic` system option (see wxSystemOptions) to 1.

Styles

This class supports the following styles:

- wxLC_LIST: Multicolumn list view, with optional small icons. Columns are computed automatically, i.e. you don't set columns as in `wxLC_REPORT`. In other words, the list wraps, unlike a wxListBox.
- wxLC_REPORT: Single or multicolumn report view, with optional header.
- wxLC_VIRTUAL: The application provides items text on demand. May only be used with `wxLC_REPORT`.
- wxLC_ICON: Large icon view, with optional labels.
- wxLC_SMALL_ICON: Small icon view, with optional labels.
- wxLC_ALIGN_TOP: Icons align to the top. Win32 default, Win32 only.
- wxLC_ALIGN_LEFT: Icons align to the left.
- wxLC_AUTOARRANGE: Icons arrange themselves. Win32 only.
- wxLC_EDIT_LABELS: Labels are editable: the application will be notified when editing starts.
- wxLC_NO_HEADER: No header in report mode.
- wxLC_SINGLE_SEL: Single selection (default is multiple).
- wxLC_SORT_ASCENDING: Sort in ascending order. (You must still supply a comparison callback in wxListCtrl::SortItems.)
- wxLC_SORT_DESCENDING: Sort in descending order. (You must still supply a comparison callback in wxListCtrl::SortItems.)
- wxLC_HRULES: Draws light horizontal rules between rows in report mode.
- wxLC_VRULES: Draws light vertical rules between columns in report mode.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxListEvent& event)

Event macros for events emitted by this class:

- EVT_LIST_BEGIN_DRAG(id, func): Begin dragging with the left mouse button. Processes a `wxEVT_LIST_BEGIN_DRAG` event type.
- EVT_LIST_BEGIN_RDRAG(id, func): Begin dragging with the right mouse button. Processes a `wxEVT_LIST_BEGIN_RDRAG` event type.
- EVT_BEGIN_LABEL_EDIT(id, func): Begin editing a label. This can be prevented by calling Veto(). Processes a `wxEVT_LIST_BEGIN_LABEL_EDIT` event type.
- EVT_LIST_END_LABEL_EDIT(id, func): Finish editing a label. This can be prevented by calling Veto(). Processes a `wxEVT_LIST_END_LABEL_EDIT` event type.
- EVT_LIST_DELETE_ITEM(id, func): An item was deleted. Processes a `wxEVT_LIST_DELETE_ITEM` event type.
- EVT_LIST_DELETE_ALL_ITEMS(id, func): All items were deleted. Processes a `wxEVT_LIST_DELETE_ALL_ITEMS` event type.

- EVT_LIST_ITEM_SELECTED(id, func): The item has been selected. Processes a `wxEVT_LIST_ITEM_SELECTED` event type.
- EVT_LIST_ITEM_DESELECTED(id, func): The item has been deselected. Processes a `wxEVT_LIST_ITEM_DESELECTED` event type.
- EVT_LIST_ITEM_ACTIVATED(id, func): The item has been activated (ENTER or double click). Processes a `wxEVT_LIST_ITEM_ACTIVATED` event type.
- EVT_LIST_ITEM_FOCUSED(id, func): The currently focused item has changed. Processes a `wxEVT_LIST_ITEM_FOCUSED` event type.
- EVT_LIST_ITEM_MIDDLE_CLICK(id, func): The middle mouse button has been clicked on an item. This is only supported by the generic control. Processes a `wxEVT_LIST_ITEM_MIDDLE_CLICK` event type.
- EVT_LIST_ITEM_RIGHT_CLICK(id, func): The right mouse button has been clicked on an item. Processes a `wxEVT_LIST_ITEM_RIGHT_CLICK` event type.
- EVT_LIST_KEY_DOWN(id, func): A key has been pressed. Processes a `wxEVT_LIST_KEY_DOWN` event type.
- EVT_LIST_INSERT_ITEM(id, func): An item has been inserted. Processes a `wxEVT_LIST_INSERT_ITEM` event type.
- EVT_LIST_COL_CLICK(id, func): A column (m_col) has been left-clicked. Processes a `wxEVT_LIST_COL_CLICK` event type.
- EVT_LIST_COL_RIGHT_CLICK(id, func): A column (m_col) has been right-clicked. Processes a `wxEVT_LIST_COL_RIGHT_CLICK` event type.
- EVT_LIST_COL_BEGIN_DRAG(id, func): The user started resizing a column - can be vetoed. Processes a `wxEVT_LIST_COL_BEGIN_DRAG` event type.
- EVT_LIST_COL_DRAGGING(id, func): The divider between columns is being dragged. Processes a `wxEVT_LIST_COL_DRAGGING` event type.
- EVT_LIST_COL_END_DRAG(id, func): A column has been resized by the user. Processes a `wxEVT_LIST_COL_END_DRAG` event type.
- EVT_LIST_CACHE_HINT(id, func): Prepare cache for a virtual list control. Processes a `wxEVT_LIST_CACHE_HINT` event type.

Library: wxCore

Category: Controls

See also

wxListCtrl Overview, wxListView, wxListBox, wxTreeCtrl, wxImageList, wxListEvent, wxListItem, wxEditableListBox

Public Member Functions

- wxListCtrl ()

  Default constructor.
- wxListCtrl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxLC_ICON, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListCtrlNameStr)

  Constructor, creating and showing a list control.
- virtual ~wxListCtrl ()

Destructor, destroying the list control.

- long AppendColumn (const wxString &heading, wxListColumnFormat format=wxLIST_FORMAT_LEFT, int width=-1)

  Adds a new column to the list control in report view mode.
- bool Arrange (int flag=wxLIST_ALIGN_DEFAULT)

  Arranges the items in icon or small icon view.
- void AssignImageList (wxImageList *imageList, int which)

  Sets the image list associated with the control and takes ownership of it (i.e.
- void ClearAll ()

  Deletes all items and all columns.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxLC_ICON, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListCtrlNameStr)

  Creates the list control.
- bool DeleteAllItems ()

  Deletes all items in the list control.
- bool DeleteColumn (int col)

  Deletes a column.
- bool DeleteItem (long item)

  Deletes the specified item.
- wxTextCtrl * EditLabel (long item, wxClassInfo *textControlClass=wxCLASSINFO(wxTextCtrl))

  Starts editing the label of the given item.
- void EnableAlternateRowColours (bool enable=true)

  Enable alternating row background colours (also called zebra striping).
- void EnableBellOnNoMatch (bool on=true)

  Enable or disable a beep if there is no match for the currently entered text when searching for the item from keyboard.
- bool EndEditLabel (bool cancel)

  Finish editing the label.
- bool EnsureVisible (long item)

  Ensures this item is visible.
- long FindItem (long start, const wxString &str, bool partial=false)

  Find an item whose label matches this string, starting from start or the beginning if start is `-1`.
- long FindItem (long start, wxUIntPtr data)

  Find an item whose data matches this data, starting from start or the beginning if 'start' is `-1`.
- long FindItem (long start, const wxPoint &pt, int direction)

  Find an item nearest this position in the specified direction, starting from start or the beginning if start is -1.
- bool GetColumn (int col, wxListItem &item) const

  Gets information about this column.
- int GetColumnCount () const

  Returns the number of columns.
- int GetColumnIndexFromOrder (int pos) const

  Gets the column index from its position in visual order.
- int GetColumnOrder (int col) const

  Gets the column visual order position.
- int GetColumnWidth (int col) const

  Gets the column width (report view only).
- wxArrayInt GetColumnsOrder () const

  Returns the array containing the orders of all columns.
- int GetCountPerPage () const

  Gets the number of items that can fit vertically in the visible area of the list control (list or report view) or the total number of items in the list control (icon or small icon view).

- wxTextCtrl * GetEditControl () const

  Returns the edit control being currently used to edit a label.
- wxImageList * GetImageList (int which) const

  Returns the specified image list.
- bool GetItem (wxListItem &info) const

  Gets information about the item.
- wxColour GetItemBackgroundColour (long item) const

  Returns the colour for this item.
- int GetItemCount () const

  Returns the number of items in the list control.
- wxUIntPtr GetItemData (long item) const

  Gets the application-defined data associated with this item.
- wxFont GetItemFont (long item) const

  Returns the item's font.
- bool GetItemPosition (long item, wxPoint &pos) const

  Returns the position of the item, in icon or small icon view.
- bool GetItemRect (long item, wxRect &rect, int code=wxLIST_RECT_BOUNDS) const

  Returns the rectangle representing the item's size and position, in physical coordinates.
- wxSize GetItemSpacing () const

  Retrieves the spacing between icons in pixels: horizontal spacing is returned as x component of the wxSize object and the vertical spacing as its y component.
- int GetItemState (long item, long stateMask) const

  Gets the item state.
- wxString GetItemText (long item, int col=0) const

  Gets the item text for this item.
- wxColour GetItemTextColour (long item) const

  Returns the colour for this item.
- long GetNextItem (long item, int geometry=wxLIST_NEXT_ALL, int state=wxLIST_STATE_DONTCARE) const

  Searches for an item with the given geometry or state, starting from item but excluding the item itself.
- int GetSelectedItemCount () const

  Returns the number of selected items in the list control.
- bool GetSubItemRect (long item, long subItem, wxRect &rect, int code=wxLIST_RECT_BOUNDS) const

  Returns the rectangle representing the size and position, in physical coordinates, of the given subitem, i.e.
- wxColour GetTextColour () const

  Gets the text colour of the list control.
- long GetTopItem () const

  Gets the index of the topmost visible item when in list or report view.
- wxRect GetViewRect () const

  Returns the rectangle taken by all items in the control.
- void SetAlternateRowColour (const wxColour &colour)

  Set the alternative row background colour to a specific colour.
- long HitTest (const wxPoint &point, int &flags, long *ptrSubItem=NULL) const

  Determines which item (if any) is at the specified point, giving details in flags.
- bool InReportView () const

  Returns true if the control is currently using wxLC_REPORT style.
- long InsertColumn (long col, const wxListItem &info)

  For report view mode (only), inserts a column.
- long InsertColumn (long col, const wxString &heading, int format=wxLIST_FORMAT_LEFT, int width=wxLIST_AUTOSIZE)

  For report view mode (only), inserts a column.

- long InsertItem (wxListItem &info)

  Inserts an item, returning the index of the new item if successful, -1 otherwise.
- long InsertItem (long index, const wxString &label)

  Insert an string item.
- long InsertItem (long index, int imageIndex)

  Insert an image item.
- long InsertItem (long index, const wxString &label, int imageIndex)

  Insert an image/string item.
- bool IsVirtual () const

  Returns true if the control is currently in virtual report view.
- void RefreshItem (long item)

  Redraws the given item.
- void RefreshItems (long itemFrom, long itemTo)

  Redraws the items between itemFrom and itemTo.
- bool ScrollList (int dx, int dy)

  Scrolls the list control.
- virtual bool SetBackgroundColour (const wxColour &col)

  Sets the background colour.
- bool SetColumn (int col, wxListItem &item)

  Sets information about this column.
- bool SetColumnWidth (int col, int width)

  Sets the column width.
- bool SetColumnsOrder (const wxArrayInt &orders)

  Changes the order in which the columns are shown.
- void SetImageList (wxImageList *imageList, int which)

  Sets the image list associated with the control.
- bool SetItem (wxListItem &info)

  Sets the data of an item.
- long SetItem (long index, int column, const wxString &label, int imageId=-1)

  Sets an item string field at a particular column.
- void SetItemBackgroundColour (long item, const wxColour &col)

  Sets the background colour for this item.
- bool SetItemColumnImage (long item, long column, int image)

  Sets the image associated with the item.
- void SetItemCount (long count)

  This method can only be used with virtual list controls.
- bool SetItemData (long item, long data)

  Associates application-defined data with this item.
- void SetItemFont (long item, const wxFont &font)

  Sets the item's font.
- bool SetItemImage (long item, int image, int selImage=-1)

  Sets the unselected and selected images associated with the item.
- bool SetItemPosition (long item, const wxPoint &pos)

  Sets the position of the item, in icon or small icon view.
- bool SetItemPtrData (long item, wxUIntPtr data)

  Associates application-defined data with this item.
- bool SetItemState (long item, long state, long stateMask)

  Sets the item state.
- void SetItemText (long item, const wxString &text)

  Sets the item text for this item.
- void SetItemTextColour (long item, const wxColour &col)

Sets the colour for this item.

- void SetSingleStyle (long style, bool add=true)

  Adds or removes a single window style.

- void SetTextColour (const wxColour &col)

  Sets the text colour of the list control.

- void SetWindowStyleFlag (long style)

  Sets the whole window style, deleting all items.

- bool SortItems (wxListCtrlCompare fnSortCallBack, wxIntPtr data)

  Call this function to sort the items in the list control.

Protected Member Functions

- virtual wxListItemAttr * OnGetItemAttr (long item) const

  This function may be overridden in the derived class for a control with `wxLC_VIRTUAL` style.

- virtual wxListItemAttr * OnGetItemColumnAttr (long item, long column) const

  This function may be overridden in the derived class for a control with `wxLC_VIRTUAL` style.

- virtual int OnGetItemColumnImage (long item, long column) const

  Override this function in the derived class for a control with `wxLC_VIRTUAL` and `wxLC_REPORT` styles in order to specify the image index for the given line and column.

- virtual int OnGetItemImage (long item) const

  This function must be overridden in the derived class for a control with `wxLC_VIRTUAL` style having an "image list" (see SetImageList(); if the control doesn't have an image list, it is not necessary to override it).

- virtual wxString OnGetItemText (long item, long column) const

  *This function **must** be overridden in the derived class for a control with `wxLC_VIRTUAL` style.*

Additional Inherited Members

21.424.2 Constructor & Destructor Documentation

wxListCtrl::wxListCtrl ( )

Default constructor.

wxListCtrl::wxListCtrl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxLC_ICON, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxListCtrlNameStr`)

Constructor, creating and showing a list control.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxListCtrl.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxListCtrl::~wxListCtrl ( ) [virtual]

Destructor, destroying the list control.

21.424.3 Member Function Documentation

long wxListCtrl::AppendColumn (const wxString & *heading*, wxListColumnFormat *format* = wxLIST_FORMAT_LEFT, int *width* = -1)

Adds a new column to the list control in report view mode.

This is just a convenient wrapper for InsertColumn() which adds the new column after all the existing ones without having to specify its position explicitly.

Since

2.9.4

bool wxListCtrl::Arrange (int *flag* = wxLIST_ALIGN_DEFAULT)

Arranges the items in icon or small icon view.

This only has effect on Win32. *flag* is one of:

- wxLIST_ALIGN_DEFAULT: Default alignment.
- wxLIST_ALIGN_LEFT: Align to the left side of the control.
- wxLIST_ALIGN_TOP: Align to the top side of the control.
- wxLIST_ALIGN_SNAP_TO_GRID: Snap to grid.

void wxListCtrl::AssignImageList (wxImageList * *imageList*, int *which*)

Sets the image list associated with the control and takes ownership of it (i.e.

the control will, unlike when using SetImageList(), delete the list when destroyed). *which* is one of `wxIMAGE_LIST_NORMAL`, `wxIMAGE_LIST_SMALL`, `wxIMAGE_LIST_STATE` (the last is unimplemented).

See also

SetImageList()

void wxListCtrl::ClearAll ( )

Deletes all items and all columns.

Note

This sends an event of type `wxEVT_LIST_DELETE_ALL_ITEMS` under all platforms.

bool wxListCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxLC_ICON, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxListCtrlNameStr`)

Creates the list control.

See wxListCtrl() for further details.

bool wxListCtrl::DeleteAllItems ( )

Deletes all items in the list control.

This function does *not* send the `wxEVT_LIST_DELETE_ITEM` event because deleting many items from the control would be too slow then (unlike wxListCtrl::DeleteItem) but it does send the special `wxEVT_LIST_DELETE_ALL_ITEMS` event if the control was not empty. If it was already empty, nothing is done and no event is sent.

Returns

true if the items were successfully deleted or if the control was already empty, false if an error occurred while deleting the items.

bool wxListCtrl::DeleteColumn (int *col*)

Deletes a column.

bool wxListCtrl::DeleteItem (long *item*)

Deletes the specified item.

This function sends the `wxEVT_LIST_DELETE_ITEM` event for the item being deleted.

See also

DeleteAllItems()

wxTextCtrl* wxListCtrl::EditLabel (long *item*, wxClassInfo * *textControlClass* = wxCLASSINFO(wxTextCtrl))

Starts editing the label of the given item.

This function generates a `EVT_LIST_BEGIN_LABEL_EDIT` event which can be vetoed so that no text control will appear for in-place editing.

If the user changed the label (i.e. s/he does not press ESC or leave the text control without changes, a `EVT_LIST_END_LABEL_EDIT` event will be sent which can be vetoed as well.

void wxListCtrl::EnableAlternateRowColours (bool *enable* = `true`)

Enable alternating row background colours (also called zebra striping).

This method can only be called for the control in virtual report mode, i.e. having wxLC_REPORT and wxLC_VIRTUAL styles.

When enabling alternating colours, the appropriate colour for the even rows is chosen automatically depending on the default foreground and background colours which are used for the odd rows.

Parameters

enable	If true, enable alternating row background colours, i.e. different colours for the odd and even rows. If false, disable this feature and use the same background colour for all rows.

Since

2.9.5

See also

SetAlternateRowColour()

void wxListCtrl::EnableBellOnNoMatch (bool *on* = `true`)

Enable or disable a beep if there is no match for the currently entered text when searching for the item from keyboard.

The default is to not beep in this case except in wxMSW where the beep is always generated by the native control and cannot be disabled, i.e. calls to this function do nothing there.

Since

2.9.5

bool wxListCtrl::EndEditLabel (bool *cancel*)

Finish editing the label.

This method allows to programmatically end editing a list control item in place. Usually it will only be called when editing is in progress, i.e. if GetEditControl() returns non-NULL. In particular, do not call it from EVT_LIST_BEGIN_LABEL_EDIT handler as the edit control is not yet fully created by then, just veto the event in this handler instead to prevent the editing from even starting.

Notice that calling this method will result in EVT_LIST_END_LABEL_EDIT event being generated.

Currently only implemented in wxMSW.

Parameters

cancel	If true, discard the changes made by user, as if `Escape` key was pressed. Otherwise, accept the changes as if `Return` was pressed.

Returns

true if item editing was finished or false if no item as being edited.

bool wxListCtrl::EnsureVisible (long *item*)

Ensures this item is visible.

long wxListCtrl::FindItem (long *start*, const wxString & *str*, bool *partial* = `false`)

Find an item whose label matches this string, starting from start or the beginning if start is `-1`.

The string comparison is case insensitive.

If *partial* is true then this method will look for items which begin with *str*.

Returns

The next matching item if any or `-1` (wxNOT_FOUND) otherwise.

long wxListCtrl::FindItem (long *start*, wxUIntPtr *data*)

Find an item whose data matches this data, starting from start or the beginning if 'start' is `-1`.

wxPerl Note: In wxPerl this method is implemented as FindItemData(start, data).

Returns

The next matching item if any or `-1` (wxNOT_FOUND) otherwise.

long wxListCtrl::FindItem (long *start,* const wxPoint & *pt,* int *direction*)

Find an item nearest this position in the specified direction, starting from *start* or the beginning if *start* is -1.

wxPerl Note: In wxPerl this method is implemented as FindItemAtPos(start, pt, direction).

Returns

The next matching item if any or `-1` (wxNOT_FOUND) otherwise.

bool wxListCtrl::GetColumn (int *col,* wxListItem & *item*) const

Gets information about this column.

See SetItem() for more information.

wxPerl Note: In wxPerl this method takes only the *col* parameter and returns a `Wx::ListItem` (or `undef`).

int wxListCtrl::GetColumnCount ( ) const

Returns the number of columns.

int wxListCtrl::GetColumnIndexFromOrder (int *pos*) const

Gets the column index from its position in visual order.

After calling SetColumnsOrder(), the index returned by this function corresponds to the value of the element number *pos* in the array returned by GetColumnsOrder().

Please see SetColumnsOrder() documentation for an example and additional remarks about the columns ordering.

See also

GetColumnOrder()

int wxListCtrl::GetColumnOrder (int *col*) const

Gets the column visual order position.

This function returns the index of the column which appears at the given visual position, e.g. calling it with *col* equal to 0 returns the index of the first shown column.

Please see SetColumnsOrder() documentation for an example and additional remarks about the columns ordering.

See also

GetColumnsOrder(), GetColumnIndexFromOrder()

wxArrayInt wxListCtrl::GetColumnsOrder ( ) const

Returns the array containing the orders of all columns.

On error, an empty array is returned.

Please see SetColumnsOrder() documentation for an example and additional remarks about the columns ordering.

See also

GetColumnOrder(), GetColumnIndexFromOrder()

int wxListCtrl::GetColumnWidth (int *col*) const

Gets the column width (report view only).

int wxListCtrl::GetCountPerPage ( ) const

Gets the number of items that can fit vertically in the visible area of the list control (list or report view) or the total number of items in the list control (icon or small icon view).

wxTextCtrl* wxListCtrl::GetEditControl ( ) const

Returns the edit control being currently used to edit a label.

Returns NULL if no label is being edited.

Note

It is currently only implemented for wxMSW and the generic version, not for the native Mac OS X version.

wxImageList* wxListCtrl::GetImageList (int *which*) const

Returns the specified image list.

which may be one of:

- wxIMAGE_LIST_NORMAL: The normal (large icon) image list.
- wxIMAGE_LIST_SMALL: The small icon image list.
- wxIMAGE_LIST_STATE: The user-defined state image list (unimplemented).

bool wxListCtrl::GetItem (wxListItem & *info*) const

Gets information about the item.

See SetItem() for more information.

You must call *info.SetId()* to set the ID of item you're interested in before calling this method, and *info.SetMask()* with the flags indicating what fields you need to retrieve from *info*.

wxPerl Note: In wxPerl this method takes as parameter the ID of the item and (optionally) the column, and returns a Wx::ListItem object.

wxColour wxListCtrl::GetItemBackgroundColour (long *item*) const

Returns the colour for this item.

If the item has no specific colour, returns an invalid colour (and not the default background control of the control itself).

See also

GetItemTextColour()

int wxListCtrl::GetItemCount ( ) const

Returns the number of items in the list control.

wxUIntPtr wxListCtrl::GetItemData (long *item*) const

Gets the application-defined data associated with this item.

wxFont wxListCtrl::GetItemFont (long *item*) const

Returns the item's font.

bool wxListCtrl::GetItemPosition (long *item*, wxPoint & *pos*) const

Returns the position of the item, in icon or small icon view.

wxPerl Note: In wxPerl this method takes only the *item* parameter and returns a `Wx::Point` (or `undef`).

bool wxListCtrl::GetItemRect (long *item*, wxRect & *rect*, int *code* = wxLIST_RECT_BOUNDS) const

Returns the rectangle representing the item's size and position, in physical coordinates.

code is one of wxLIST_RECT_BOUNDS, wxLIST_RECT_ICON, wxLIST_RECT_LABEL.

wxPerl Note: In wxPerl this method takes only the *item* and *code* parameters and returns a `Wx::Rect` (or `undef`).

wxSize wxListCtrl::GetItemSpacing ( ) const

Retrieves the spacing between icons in pixels: horizontal spacing is returned as `x` component of the wxSize object and the vertical spacing as its `y` component.

int wxListCtrl::GetItemState (long *item*, long *stateMask*) const

Gets the item state.

For a list of state flags, see SetItem(). The *stateMask* indicates which state flags are of interest.

wxString wxListCtrl::GetItemText (long *item*, int *col* = `0`) const

Gets the item text for this item.

Parameters

item	Item (zero-based) index.
col	Item column (zero-based) index. Column 0 is the default. This parameter is new in wxWidgets 2.9.1.

wxColour wxListCtrl::GetItemTextColour (long *item*) const

Returns the colour for this item.

If the item has no specific colour, returns an invalid colour (and not the default foreground control of the control itself as this wouldn't allow distinguishing between items having the same colour as the current control foreground and items with default colour which, hence, have always the same colour as the control).

long wxListCtrl::GetNextItem (long *item*, int *geometry* = wxLIST_NEXT_ALL, int *state* = wxLIST_STATE_DONTCARE) const

Searches for an item with the given geometry or state, starting from *item* but excluding the *item* itself.

If *item* is -1, the first item that matches the specified flags will be returned. Returns the first item with given state following *item* or -1 if no such item found. This function may be used to find all selected items in the control like this:

```
long item = -1;
for ( ;; )
{
    item = listctrl->GetNextItem(item,
                                 wxLIST_NEXT_ALL,
                                 wxLIST_STATE_SELECTED);
    if ( item == -1 )
        break;

    // this item is selected - do whatever is needed with it
    wxLogMessage("Item %ld is selected.", item);
}
```

geometry can be one of:

- wxLIST_NEXT_ABOVE: Searches for an item above the specified item.
- wxLIST_NEXT_ALL: Searches for subsequent item by index.
- wxLIST_NEXT_BELOW: Searches for an item below the specified item.
- wxLIST_NEXT_LEFT: Searches for an item to the left of the specified item.
- wxLIST_NEXT_RIGHT: Searches for an item to the right of the specified item.

Note

this parameter is only supported by wxMSW currently and ignored on other platforms.

state can be a bitlist of the following:

- wxLIST_STATE_DONTCARE: Don't care what the state is.
- wxLIST_STATE_DROPHILITED: The item indicates it is a drop target.
- wxLIST_STATE_FOCUSED: The item has the focus.
- wxLIST_STATE_SELECTED: The item is selected.
- wxLIST_STATE_CUT: The item is selected as part of a cut and paste operation.

int wxListCtrl::GetSelectedItemCount ( ) const

Returns the number of selected items in the list control.

bool wxListCtrl::GetSubItemRect (long *item*, long *subItem*, wxRect & *rect*, int *code* = wxLIST_RECT_BOUNDS) const

Returns the rectangle representing the size and position, in physical coordinates, of the given subitem, i.e.

the part of the row *item* in the column *subItem*.

This method is only meaningful when the wxListCtrl is in the report mode. If *subItem* parameter is equal to the special value `wxLIST_GETSUBITEMRECT_WHOLEITEM` the return value is the same as for GetItemRect().

code can be one of `wxLIST_RECT_BOUNDS`, `wxLIST_RECT_ICON` or `wxLIST_RECT_LABEL`.

Since

2.7.0

wxColour wxListCtrl::GetTextColour ( ) const

Gets the text colour of the list control.

long wxListCtrl::GetTopItem ( ) const

Gets the index of the topmost visible item when in list or report view.

wxRect wxListCtrl::GetViewRect ( ) const

Returns the rectangle taken by all items in the control.

In other words, if the controls client size were equal to the size of this rectangle, no scrollbars would be needed and no free space would be left.

Note that this function only works in the icon and small icon views, not in list or report views (this is a limitation of the native Win32 control).

long wxListCtrl::HitTest (const wxPoint & *point*, int & *flags*, long * *ptrSubItem* = NULL) const

Determines which item (if any) is at the specified point, giving details in *flags*.

Returns index of the item or wxNOT_FOUND if no item is at the specified point.

flags will be a combination of the following flags:

- wxLIST_HITTEST_ABOVE: Above the client area.
- wxLIST_HITTEST_BELOW: Below the client area.
- wxLIST_HITTEST_NOWHERE: In the client area but below the last item.
- wxLIST_HITTEST_ONITEMICON: On the bitmap associated with an item.
- wxLIST_HITTEST_ONITEMLABEL: On the label (string) associated with an item.
- wxLIST_HITTEST_ONITEMRIGHT: In the area to the right of an item.
- wxLIST_HITTEST_ONITEMSTATEICON: On the state icon for a tree view item that is in a user-defined state.
- wxLIST_HITTEST_TOLEFT: To the right of the client area.
- wxLIST_HITTEST_TORIGHT: To the left of the client area.
- wxLIST_HITTEST_ONITEM: Combination of wxLIST_HITTEST_ONITEMICON, wxLIST_HITTEST_ONITEMLABEL, wxLIST_HITTEST_ONITEMSTATEICON.

If *ptrSubItem* is not NULL and the wxListCtrl is in the report mode the subitem (or column) number will also be provided. This feature is only available in version 2.7.0 or higher and is currently only implemented under wxMSW and requires at least comctl32.dll of version 4.70 on the host system or the value stored in *ptrSubItem* will be always -1. To compile this feature into wxWidgets library you need to have access to commctrl.h of version 4.70 that is provided by Microsoft.

wxPerl Note: In wxPerl this method only takes the *point* parameter and returns a 2-element list (item, flags).

bool wxListCtrl::InReportView ( ) const

Returns true if the control is currently using wxLC_REPORT style.

long wxListCtrl::InsertColumn (long *col,* const wxListItem & *info*)

For report view mode (only), inserts a column.

For more details, see SetItem(). Also see InsertColumn(long, const wxString&, int, int) overload for a usually more convenient alternative to this method and the description of how the item width is interpreted by this method.

long wxListCtrl::InsertColumn (long *col,* const wxString & *heading,* int *format* = wxLIST_FORMAT_LEFT, int *width* = wxLIST_AUTOSIZE)

For report view mode (only), inserts a column.

Insert a new column in the list control in report view mode at the given position specifying its most common attributes.

Notice that to set the image for the column you need to use Insert(long, const wxListItem&) overload and specify wxLIST_MASK_IMAGE in the item mask.

Parameters

col	The index where the column should be inserted. Valid indices are from 0 up to GetColumnCount() inclusive and the latter can be used to append the new column after the last existing one.
heading	The string specifying the column heading.
format	The flags specifying the control heading text alignment.
width	If positive, the width of the column in pixels. Otherwise it can be `wxLIST_AUTOSIZE` to choose the default size for the column or `wxLIST_AUTOSIZE_USEHEADER` to fit the column width to *heading* or to extend to fill all the remaining space for the last column. Notice that in case of `wxLIST_AUTOSIZE` fixed width is used as there are no items in this column to use for determining its best size yet. If you want to fit the column to its contents, use SetColumnWidth() after adding the items with values in this column.

Returns

The index of the inserted column or -1 if adding it failed.

long wxListCtrl::InsertItem (wxListItem & *info*)

Inserts an item, returning the index of the new item if successful, -1 otherwise.

Parameters

info	wxListItem object

long wxListCtrl::InsertItem (long *index,* const wxString & *label*)

Insert an string item.

Parameters

index	Index of the new item, supplied by the application
label	String label

wxPerl Note: In wxPerl this method is implemented as InsertStringItem(index, label).

long wxListCtrl::InsertItem (long *index,* int *imageIndex*)

Insert an image item.

Parameters

index	Index of the new item, supplied by the application
imageIndex	Index into the image list associated with this control and view style

wxPerl Note: In wxPerl this method is implemented as InsertImageItem(index, imageIndex).

long wxListCtrl::InsertItem (long *index,* const wxString & *label,* int *imageIndex*)

Insert an image/string item.

Parameters

index	Index of the new item, supplied by the application
label	String label
imageIndex	Index into the image list associated with this control and view style

wxPerl Note: In wxPerl this method is implemented as InsertImageStringItem(index, label, imageIndex).

bool wxListCtrl::IsVirtual ( ) const

Returns true if the control is currently in virtual report view.

virtual wxListItemAttr* wxListCtrl::OnGetItemAttr (long *item*) const [protected], [virtual]

This function may be overridden in the derived class for a control with `wxLC_VIRTUAL` style.

It should return the attribute for the specified `item` or NULL to use the default appearance parameters.

wxListCtrl will not delete the pointer or keep a reference of it. You can return the same wxListItemAttr pointer for every OnGetItemAttr() call.

The base class version always returns NULL.

See also

OnGetItemImage(), OnGetItemColumnImage(), OnGetItemText(), OnGetItemColumnAttr()

virtual wxListItemAttr* wxListCtrl::OnGetItemColumnAttr (long *item,* long *column*) const [protected], [virtual]

This function may be overridden in the derived class for a control with `wxLC_VIRTUAL` style.

It should return the attribute for the for the specified *item* and *column* or NULL to use the default appearance parameters.

The base class version returns `OnGetItemAttr(item)`.

Note

Currently this function is only called under wxMSW, the other ports only support OnGetItemAttr()

See also

OnGetItemAttr(), OnGetItemText(), OnGetItemImage(), OnGetItemColumnImage(),

virtual int wxListCtrl::OnGetItemColumnImage (long *item,* long *column*) const `[protected], [virtual]`

Override this function in the derived class for a control with `wxLC_VIRTUAL` and `wxLC_REPORT` styles in order to specify the image index for the given line and column.

The base class version always calls OnGetItemImage() for the first column, else it returns -1.

See also

OnGetItemText(), OnGetItemImage(), OnGetItemAttr(), OnGetItemColumnAttr()

virtual int wxListCtrl::OnGetItemImage (long *item*) const `[protected], [virtual]`

This function must be overridden in the derived class for a control with `wxLC_VIRTUAL` style having an "image list" (see SetImageList(); if the control doesn't have an image list, it is not necessary to override it).

It should return the index of the items image in the controls image list or -1 for no image.

In a control with `wxLC_REPORT` style, OnGetItemImage() only gets called for the first column of each line.

The base class version always returns -1.

See also

OnGetItemText(), OnGetItemColumnImage(), OnGetItemAttr()

virtual wxString wxListCtrl::OnGetItemText (long *item,* long *column*) const `[protected], [virtual]`

This function **must** be overridden in the derived class for a control with `wxLC_VIRTUAL` style.

It should return the string containing the text of the given *column* for the specified `item`.

See also

SetItemCount(), OnGetItemImage(), OnGetItemColumnImage(), OnGetItemAttr()

void wxListCtrl::RefreshItem (long *item*)

Redraws the given *item*.

This is only useful for the virtual list controls as without calling this function the displayed value of the item doesn't change even when the underlying data does change.

See also

RefreshItems()

void wxListCtrl::RefreshItems (long *itemFrom,* long *itemTo*)

Redraws the items between *itemFrom* and *itemTo*.

The starting item must be less than or equal to the ending one.

Just as RefreshItem() this is only useful for virtual list controls.

bool wxListCtrl::ScrollList (int *dx,* int *dy*)

Scrolls the list control.

If in icon, small icon or report view mode, *dx* specifies the number of pixels to scroll. If in list view mode, *dx* specifies the number of columns to scroll. *dy* always specifies the number of pixels to scroll vertically.

Note

This method is currently only implemented in the Windows version.

void wxListCtrl::SetAlternateRowColour (const wxColour & *colour*)

Set the alternative row background colour to a specific colour.

It is recommended to call EnableAlternateRowColours() instead of using these methods as native implementations of this control might support alternating row colours but not setting the exact colour to be used for them.

As EnableAlternateRowColours(), this method can only be used with controls having wxLC_REPORT and wxLC_VIRTUAL styles.

Parameters

colour	A valid alternative row background colour to enable alternating rows or invalid colour to disable them and use the same colour for all rows.

Since

2.9.5

virtual bool wxListCtrl::SetBackgroundColour (const wxColour & *col*) [virtual]

Sets the background colour.

Note that the wxWindow::GetBackgroundColour() function of wxWindow base class can be used to retrieve the current background colour.

Reimplemented from wxWindow.

bool wxListCtrl::SetColumn (int *col,* wxListItem & *item*)

Sets information about this column.

See SetItem() for more information.

bool wxListCtrl::SetColumnsOrder (const wxArrayInt & *orders*)

Changes the order in which the columns are shown.

By default, the columns of a list control appear on the screen in order of their indices, i.e. the column 0 appears first, the column 1 next and so on. However by using this function it is possible to arbitrarily reorder the columns visual order and the user can also rearrange the columns interactively by dragging them. In this case, the index of the column is not the same as its order and the functions GetColumnOrder() and GetColumnIndexFromOrder() should be used to translate between them. Notice that all the other functions still work with the column indices, i.e. the visual positioning of the columns on screen doesn't affect the code setting or getting their values for example.

The *orders* array must have the same number elements as the number of columns and contain each position exactly once. Its n-th element contains the index of the column to be shown in n-th position, so for a control with three columns passing an array with elements 2, 0 and 1 results in the third column being displayed first, the first one next and the second one last.

Example of using it:

```
wxListCtrl *list = new wxListCtrl(...);
for ( int i = 0; i < 3; i++ )
    list->InsertColumn(i, wxString::Format("Column %d", i));

wxArrayInt order(3);
order[0] = 2;
order[1] = 0;
order[2] = 1;
list->SetColumnsOrder(order);

// now list->GetColumnsOrder() will return order and
// list->GetColumnIndexFromOrder(n) will return order[n] and
// list->GetColumnOrder() will return 1, 2 and 0 for the column 0,
// 1 and 2 respectively
```

Please notice that this function makes sense for report view only and currently is only implemented in wxMSW port. To avoid explicit tests for **wxMSW** in your code, please use `wxHAS_LISTCTRL_COLUMN_ORDER` as this will allow it to start working under the other platforms when support for the column reordering is added there.

See also

GetColumnsOrder()

bool wxListCtrl::SetColumnWidth (int *col,* int *width*)

Sets the column width.

width can be a width in pixels or `wxLIST_AUTOSIZE` (-1) or `wxLIST_AUTOSIZE_USEHEADER` (-2).

`wxLIST_AUTOSIZE` will resize the column to the length of its longest item.

`wxLIST_AUTOSIZE_USEHEADER` will resize the column to the length of the header (Win32) or 80 pixels (other platforms).

In small or normal icon view, *col* must be -1, and the column width is set for all columns.

void wxListCtrl::SetImageList (**wxImageList** * *imageList,* int *which*)

Sets the image list associated with the control.

which is one of `wxIMAGE_LIST_NORMAL`, `wxIMAGE_LIST_SMALL`, `wxIMAGE_LIST_STATE` (the last is unimplemented).

This method does not take ownership of the image list, you have to delete it yourself.

See also

AssignImageList()

bool wxListCtrl::SetItem (**wxListItem** & *info*)

Sets the data of an item.

Using the wxListItem's mask and state mask, you can change only selected attributes of a wxListCtrl item.

long wxListCtrl::SetItem (long *index,* int *column,* const **wxString** & *label,* int *imageId* = -1)

Sets an item string field at a particular column.

void wxListCtrl::SetItemBackgroundColour (long *item,* const wxColour & *col*)

Sets the background colour for this item.

This function only works in report view mode. The colour can be retrieved using GetItemBackgroundColour().

bool wxListCtrl::SetItemColumnImage (long *item,* long *column,* int *image*)

Sets the image associated with the item.

In report view, you can specify the column. The image is an index into the image list associated with the list control.

void wxListCtrl::SetItemCount (long *count*)

This method can only be used with virtual list controls.

It is used to indicate to the control the number of items it contains. After calling it, the main program should be ready to handle calls to various item callbacks (such as wxListCtrl::OnGetItemText) for all items in the range from 0 to *count*.

Notice that the control is not necessarily redrawn after this call as it may be undesirable if an item which is not visible on the screen anyhow was added to or removed from a control displaying many items, if you do need to refresh the display you can just call Refresh() manually.

bool wxListCtrl::SetItemData (long *item,* long *data*)

Associates application-defined data with this item.

Notice that this function cannot be used to associate pointers with the control items, use SetItemPtrData() instead.

void wxListCtrl::SetItemFont (long *item,* const wxFont & *font*)

Sets the item's font.

bool wxListCtrl::SetItemImage (long *item,* int *image,* int *selImage* = `-1`)

Sets the unselected and selected images associated with the item.

The images are indices into the image list associated with the list control.

bool wxListCtrl::SetItemPosition (long *item,* const wxPoint & *pos*)

Sets the position of the item, in icon or small icon view.

Windows only.

bool wxListCtrl::SetItemPtrData (long *item,* wxUIntPtr *data*)

Associates application-defined data with this item.

The *data* parameter may be either an integer or a pointer cast to the `wxUIntPtr` type which is guaranteed to be large enough to be able to contain all integer types and pointers.

Since

2.8.4

bool wxListCtrl::SetItemState (long *item,* long *state,* long *stateMask*)

Sets the item state.

The *stateMask* is a combination of `wxLIST_STATE_XXX` constants described in wxListItem documentation. For each of the bits specified in *stateMask*, the corresponding state is set or cleared depending on whether *state* argument contains the same bit or not.

So to select an item you can use

```
list->SetItemState(item, wxLIST_STATE_SELECTED,
      wxLIST_STATE_SELECTED);
```

while to deselect it you should use

```
list->SetItemState(item, 0, wxLIST_STATE_SELECTED);
```

Consider using wxListView if possible to avoid dealing with this error-prone and confusing method.

void wxListCtrl::SetItemText (long *item,* const wxString & *text*)

Sets the item text for this item.

void wxListCtrl::SetItemTextColour (long *item,* const wxColour & *col*)

Sets the colour for this item.

This function only works in report view. The colour can be retrieved using GetItemTextColour().

void wxListCtrl::SetSingleStyle (long *style,* bool *add* = `true`)

Adds or removes a single window style.

void wxListCtrl::SetTextColour (const wxColour & *col*)

Sets the text colour of the list control.

void wxListCtrl::SetWindowStyleFlag (long *style*) `[virtual]`

Sets the whole window style, deleting all items.

Reimplemented from wxWindow.

bool wxListCtrl::SortItems (wxListCtrlCompare *fnSortCallBack,* wxIntPtr *data*)

Call this function to sort the items in the list control.

Sorting is done using the specified *fnSortCallBack* function. This function must have the following prototype:

```
int wxCALLBACK wxListCompareFunction(wxIntPtr item1, wxIntPtr item2,
      wxIntPtr sortData)
```

It is called each time when the two items must be compared and should return 0 if the items are equal, negative value if the first item is less than the second one and positive value if the first one is greater than the second one (the same convention as used by `qsort(3)`).

The parameter *item1* is the client data associated with the first item (**NOT** the index). The parameter *item2* is the client data associated with the second item (**NOT** the index). The parameter *data* is the value passed to SortItems() itself.

Notice that the control may only be sorted on client data associated with the items, so you must use SetItemData if you want to be able to sort the items in the control.

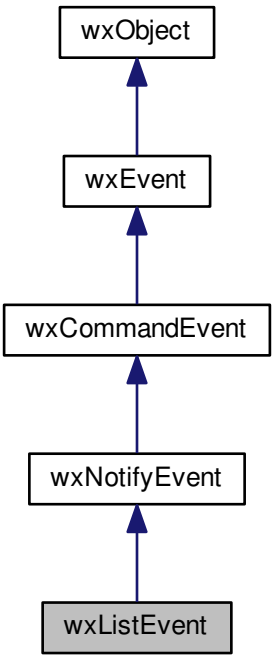
Please see the List Control Sample for an example of using this function.

wxPerl Note: In wxPerl the comparison function must take just two parameters; however, you may use a closure to achieve an effect similar to the SortItems third parameter.

21.425 wxListEvent Class Reference

`#include <wx/listctrl.h>`

Inheritance diagram for wxListEvent:



21.425.1 Detailed Description

A list event holds information about events associated with wxListCtrl objects.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxListEvent& event)

Event macros:

- EVT_LIST_BEGIN_DRAG(id, func): Begin dragging with the left mouse button.
- EVT_LIST_BEGIN_RDRAG(id, func): Begin dragging with the right mouse button.
- EVT_LIST_BEGIN_LABEL_EDIT(id, func): Begin editing a label. This can be prevented by calling Veto().
- EVT_LIST_END_LABEL_EDIT(id, func): Finish editing a label. This can be prevented by calling Veto().
- EVT_LIST_DELETE_ITEM(id, func): Delete an item.
- EVT_LIST_DELETE_ALL_ITEMS(id, func): Delete all items.
- EVT_LIST_ITEM_SELECTED(id, func): The item has been selected.
- EVT_LIST_ITEM_DESELECTED(id, func): The item has been deselected.
- EVT_LIST_ITEM_ACTIVATED(id, func): The item has been activated (ENTER or double click).
- EVT_LIST_ITEM_FOCUSED(id, func): The currently focused item has changed.
- EVT_LIST_ITEM_MIDDLE_CLICK(id, func): The middle mouse button has been clicked on an item.
- EVT_LIST_ITEM_RIGHT_CLICK(id, func): The right mouse button has been clicked on an item.
- EVT_LIST_KEY_DOWN(id, func): A key has been pressed. GetIndex() may be -1 if no item is selected.
- EVT_LIST_INSERT_ITEM(id, func): An item has been inserted.
- EVT_LIST_COL_CLICK(id, func): A column (m_col) has been left-clicked.
- EVT_LIST_COL_RIGHT_CLICK(id, func): A column (m_col) (which can be -1 if the click occurred outside any column) has been right-clicked.
- EVT_LIST_COL_BEGIN_DRAG(id, func): The user started resizing a column - can be vetoed.
- EVT_LIST_COL_DRAGGING(id, func): The divider between columns is being dragged.
- EVT_LIST_COL_END_DRAG(id, func): A column has been resized by the user.
- EVT_LIST_CACHE_HINT(id, func): Prepare cache for a virtual list control

Library: wxCore

Category: Events

See also

wxListCtrl

Public Member Functions

- wxListEvent (wxEventType commandType=wxEVT_NULL, int id=0)

  Constructor.
- long GetCacheFrom () const

  For `EVT_LIST_CACHE_HINT` event only: return the first item which the list control advises us to cache.
- long GetCacheTo () const

  For `EVT_LIST_CACHE_HINT` event only: return the last item (inclusive) which the list control advises us to cache.
- int GetColumn () const

  The column position: it is only used with `COL` events.
- wxUIntPtr GetData () const

  The data.

- int GetImage () const

  The image.
- long GetIndex () const

  The item index.
- const wxListItem & GetItem () const

  An item object, used by some events.
- int GetKeyCode () const

  Key code if the event is a keypress event.
- const wxString & GetLabel () const

  The (new) item label for `EVT_LIST_END_LABEL_EDIT` event.
- long GetMask () const

  The mask.
- wxPoint GetPoint () const

  The position of the mouse pointer if the event is a drag event.
- const wxString & GetText () const

  The text.
- bool IsEditCancelled () const

  This method only makes sense for `EVT_LIST_END_LABEL_EDIT` message and returns true if it the label editing has been cancelled by the user (GetLabel() returns an empty string in this case but it doesn't allow the application to distinguish between really cancelling the edit and the admittedly rare case when the user wants to rename it to an empty string).

Additional Inherited Members

21.425.2 Constructor & Destructor Documentation

wxListEvent::wxListEvent (wxEventType *commandType* = wxEVT_NULL, int *id* = `0`)

Constructor.

21.425.3 Member Function Documentation

long wxListEvent::GetCacheFrom ( ) const

For `EVT_LIST_CACHE_HINT` event only: return the first item which the list control advises us to cache.

long wxListEvent::GetCacheTo ( ) const

For `EVT_LIST_CACHE_HINT` event only: return the last item (inclusive) which the list control advises us to cache.

int wxListEvent::GetColumn ( ) const

The column position: it is only used with `COL` events.

For the column dragging events, it is the column to the left of the divider being dragged, for the column click events it may be -1 if the user clicked in the list control header outside any column.

wxUIntPtr wxListEvent::GetData ( ) const

The data.

int wxListEvent::GetImage ( ) const

The image.

long wxListEvent::GetIndex ( ) const

The item index.

const wxListItem& wxListEvent::GetItem ( ) const

An item object, used by some events.

See also wxListCtrl::SetItem.

int wxListEvent::GetKeyCode ( ) const

Key code if the event is a keypress event.

const wxString& wxListEvent::GetLabel ( ) const

The (new) item label for `EVT_LIST_END_LABEL_EDIT` event.

long wxListEvent::GetMask ( ) const

The mask.

wxPoint wxListEvent::GetPoint ( ) const

The position of the mouse pointer if the event is a drag event.

const wxString& wxListEvent::GetText ( ) const

The text.

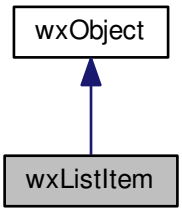
bool wxListEvent::IsEditCancelled ( ) const

This method only makes sense for `EVT_LIST_END_LABEL_EDIT` message and returns true if it the label editing has been cancelled by the user (GetLabel() returns an empty string in this case but it doesn't allow the application to distinguish between really cancelling the edit and the admittedly rare case when the user wants to rename it to an empty string).

21.426 wxListItem Class Reference

```
#include <wx/listctrl.h>
```

Inheritance diagram for wxListItem:



21.426.1 Detailed Description

This class stores information about a wxListCtrl item or column.

wxListItem is a class which contains information about:

- Zero based item position; see SetId() and GetId().
- Zero based column index; see SetColumn() and GetColumn().
- The label (or header for columns); see SetText() and GetText().
- The zero based index into an image list; see GetImage() and SetImage().
- Application defined data; see SetData() and GetData().
- For columns only: the width of the column; see SetWidth() and GetWidth().
- For columns only: the format of the column; one of `wxLIST_FORMAT_LEFT`, `wxLIST_FORMAT_RIGHT`, `wxLIST_FORMAT_CENTRE`. See SetAlign() and GetAlign().
- The state of the item; see SetState() and GetState(). This is a bitlist of the following flags:
  - `wxLIST_STATE_FOCUSED`: The item has the focus.
  - `wxLIST_STATE_SELECTED`: The item is selected.
  - `wxLIST_STATE_DONTCARE`: Don't care what the state is. Win32 only.
  - `wxLIST_STATE_DROPHILITED`: The item is highlighted to receive a drop event. Win32 only.
  - `wxLIST_STATE_CUT`: The item is in the cut state. Win32 only.
- A mask indicating which state flags are valid; this is a bitlist of the flags reported above for the item state. See SetStateMask() and GetStateMask().
- A mask indicating which fields of this class are valid; see SetMask() and GetMask(). This is a bitlist of the following flags:
  - `wxLIST_MASK_STATE`: The state field is valid.
  - `wxLIST_MASK_TEXT`: The label field is valid.
  - `wxLIST_MASK_IMAGE`: The image field is valid.
  - `wxLIST_MASK_DATA`: The application-defined data field is valid.
  - `wxLIST_MASK_WIDTH`: The column width field is valid.
  - `wxLIST_MASK_FORMAT`: The column format field is valid.

The wxListItem object can also contain item-specific colour and font information: for this you need to call one of SetTextColour(), SetBackgroundColour() or SetFont() functions on it passing it the colour/font to use. If the colour/font is not specified, the default list control colour/font is used.

Library: wxCore

Category: Data Structures

See also

wxListCtrl

Public Member Functions

- wxListItem ()

  Constructor.
- void Clear ()

  Resets the item state to the default.
- wxListColumnFormat GetAlign () const

  Returns the alignment for this item.
- wxColour GetBackgroundColour () const

  Returns the background colour for this item.
- int GetColumn () const

  Returns the zero-based column; meaningful only in report mode.
- wxUIntPtr GetData () const

  Returns client data associated with the control.
- wxFont GetFont () const

  Returns the font used to display the item.
- long GetId () const

  Returns the zero-based item position.
- int GetImage () const

  Returns the zero-based index of the image associated with the item into the image list.
- long GetMask () const

  Returns a bit mask indicating which fields of the structure are valid.
- long GetState () const

  Returns a bit field representing the state of the item.
- const wxString & GetText () const

  Returns the label/header text.
- wxColour GetTextColour () const

  Returns the text colour.
- int GetWidth () const

  Meaningful only for column headers in report mode.
- void SetAlign (wxListColumnFormat align)

  Sets the alignment for the item.
- void SetBackgroundColour (const wxColour &colBack)

  Sets the background colour for the item.
- void SetColumn (int col)

  Sets the zero-based column.
- void SetFont (const wxFont &font)

  Sets the font for the item.
- void SetId (long id)

  Sets the zero-based item position.
- void SetImage (int image)

  Sets the zero-based index of the image associated with the item into the image list.

- void SetMask (long mask)

  Sets the mask of valid fields.
- void SetState (long state)

  Sets the item state flags (note that the valid state flags are influenced by the value of the state mask, see wxListItem::SetStateMask).
- void SetStateMask (long stateMask)

  Sets the bitmask that is used to determine which of the state flags are to be set.
- void SetText (const wxString &text)

  Sets the text label for the item.
- void SetTextColour (const wxColour &colText)

  Sets the text colour for the item.
- void SetWidth (int width)

  Meaningful only for column headers in report mode.

- void SetData (long data)

  Sets client data for the item.
- void SetData (void *data)

  Sets client data for the item.

Additional Inherited Members

21.426.2 Constructor & Destructor Documentation

wxListItem::wxListItem ( )

Constructor.

21.426.3 Member Function Documentation

void wxListItem::Clear ( )

Resets the item state to the default.

wxListColumnFormat wxListItem::GetAlign ( ) const

Returns the alignment for this item.

Can be one of `wxLIST_FORMAT_LEFT`, `wxLIST_FORMAT_RIGHT` or `wxLIST_FORMAT_CENTRE`.

wxColour wxListItem::GetBackgroundColour ( ) const

Returns the background colour for this item.

int wxListItem::GetColumn ( ) const

Returns the zero-based column; meaningful only in report mode.

wxUIntPtr wxListItem::GetData ( ) const

Returns client data associated with the control.

Please note that client data is associated with the item and not with subitems.

wxFont wxListItem::GetFont ( ) const

Returns the font used to display the item.

long wxListItem::GetId ( ) const

Returns the zero-based item position.

int wxListItem::GetImage ( ) const

Returns the zero-based index of the image associated with the item into the image list.

long wxListItem::GetMask ( ) const

Returns a bit mask indicating which fields of the structure are valid.

Can be any combination of the following values:

- wxLIST_MASK_STATE: **GetState** is valid.
- wxLIST_MASK_TEXT: **GetText** is valid.
- wxLIST_MASK_IMAGE: **GetImage** is valid.
- wxLIST_MASK_DATA: **GetData** is valid.
- wxLIST_MASK_WIDTH: **GetWidth** is valid.
- wxLIST_MASK_FORMAT: **GetFormat** is valid.

long wxListItem::GetState ( ) const

Returns a bit field representing the state of the item.

Can be any combination of:

- wxLIST_STATE_DONTCARE: Don't care what the state is. Win32 only.
- wxLIST_STATE_DROPHILITED: The item is highlighted to receive a drop event. Win32 only.
- wxLIST_STATE_FOCUSED: The item has the focus.
- wxLIST_STATE_SELECTED: The item is selected.
- wxLIST_STATE_CUT: The item is in the cut state. Win32 only.

const wxString& wxListItem::GetText ( ) const

Returns the label/header text.

wxColour wxListItem::GetTextColour ( ) const

Returns the text colour.

int wxListItem::GetWidth ( ) const

Meaningful only for column headers in report mode.

Returns the column width.

void wxListItem::SetAlign (wxListColumnFormat *align*)

Sets the alignment for the item.

See also GetAlign()

void wxListItem::SetBackgroundColour (const wxColour & *colBack*)

Sets the background colour for the item.

void wxListItem::SetColumn (int *col*)

Sets the zero-based column.

Meaningful only in report mode.

void wxListItem::SetData (long *data*)

Sets client data for the item.

Please note that client data is associated with the item and not with subitems.

void wxListItem::SetData (void * *data*)

Sets client data for the item.

Please note that client data is associated with the item and not with subitems.

void wxListItem::SetFont (const wxFont & *font*)

Sets the font for the item.

void wxListItem::SetId (long *id*)

Sets the zero-based item position.

void wxListItem::SetImage (int *image*)

Sets the zero-based index of the image associated with the item into the image list.

void wxListItem::SetMask (long *mask*)

Sets the mask of valid fields.

See GetMask().

void wxListItem::SetState (long *state*)

Sets the item state flags (note that the valid state flags are influenced by the value of the state mask, see wxListItem::SetStateMask).

See GetState() for valid flag values.

void wxListItem::SetStateMask (long *stateMask*)

Sets the bitmask that is used to determine which of the state flags are to be set.

See also SetState().

void wxListItem::SetText (const wxString & *text*)

Sets the text label for the item.

void wxListItem::SetTextColour (const wxColour & *colText*)

Sets the text colour for the item.

void wxListItem::SetWidth (int *width*)

Meaningful only for column headers in report mode.

Sets the column width.

21.427 wxListItemAttr Class Reference

```
#include <wx/listctrl.h>
```

21.427.1 Detailed Description

Represents the attributes (color, font, ...) of a wxListCtrl's wxListItem.

Library: wxCore

Category: Data Structures

See also

wxListCtrl Overview, wxListCtrl, wxListItem

Public Member Functions

- wxListItemAttr ()

  Default Constructor.
- wxListItemAttr (const wxColour &colText, const wxColour &colBack, const wxFont &font)

  Construct a wxListItemAttr with the specified foreground and background colors and font.
- const wxColour & GetBackgroundColour () const

  Returns the currently set background color.

- const wxFont & GetFont () const

  Returns the currently set font.
- const wxColour & GetTextColour () const

  Returns the currently set text color.
- bool HasBackgroundColour () const

  Returns true if the currently set background color is valid.
- bool HasFont () const

  Returns true if the currently set font is valid.
- bool HasTextColour () const

  Returns true if the currently set text color is valid.
- void SetBackgroundColour (const wxColour &colour)

  Sets a new background color.
- void SetFont (const wxFont &font)

  Sets a new font.
- void SetTextColour (const wxColour &colour)

  Sets a new text color.

21.427.2 Constructor & Destructor Documentation

wxListItemAttr::wxListItemAttr ( )

Default Constructor.

wxListItemAttr::wxListItemAttr (const wxColour & *colText,* const wxColour & *colBack,* const wxFont & *font*)

Construct a wxListItemAttr with the specified foreground and background colors and font.

21.427.3 Member Function Documentation

const wxColour& wxListItemAttr::GetBackgroundColour ( ) const

Returns the currently set background color.

const wxFont& wxListItemAttr::GetFont ( ) const

Returns the currently set font.

const wxColour& wxListItemAttr::GetTextColour ( ) const

Returns the currently set text color.

bool wxListItemAttr::HasBackgroundColour ( ) const

Returns true if the currently set background color is valid.

bool wxListItemAttr::HasFont ( ) const

Returns true if the currently set font is valid.

bool wxListItemAttr::HasTextColour ( ) const

Returns true if the currently set text color is valid.

void wxListItemAttr::SetBackgroundColour (const wxColour & *colour*)

Sets a new background color.

void wxListItemAttr::SetFont (const wxFont & *font*)

Sets a new font.

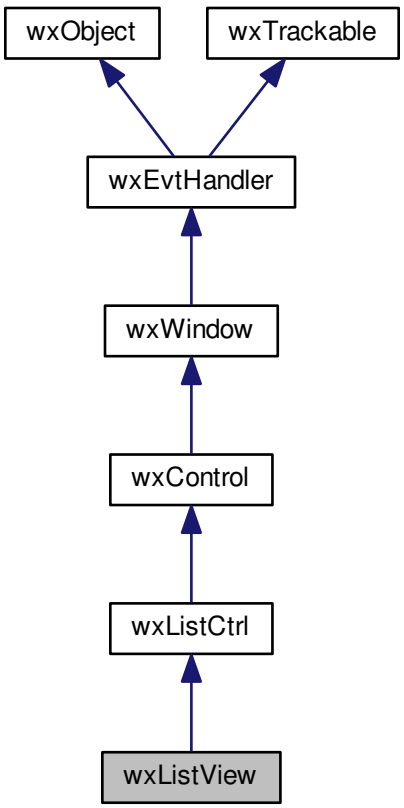
void wxListItemAttr::SetTextColour (const wxColour & *colour*)

Sets a new text color.

21.428 wxListView Class Reference

`#include <wx/listctrl.h>`

Inheritance diagram for wxListView:



21.428.1 Detailed Description

This class currently simply presents a simpler to use interface for the wxListCtrl – it can be thought of as a *façade* for that complicated class.

Using it is preferable to using wxListCtrl directly whenever possible because in the future some ports might implement wxListView but not the full set of wxListCtrl features.

Other than different interface, this class is identical to wxListCtrl. In particular, it uses the same events, same window styles and so on.

Library: wxCore

Category: Controls

See also

wxListView::SetColumnImage

Public Member Functions

- wxListView ()

  Default constructor.
- wxListView (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxLC_ICON, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxListCtrlNameStr)

  Constructor, creating and showing a listview control.
- virtual ~wxListView ()

  Destructor, destroying the listview control.
- void ClearColumnImage (int col)

  Resets the column image – after calling this function, no image will be shown.
- void Focus (long index)

  Sets focus to the item with the given index.
- long GetFirstSelected () const

  Returns the first selected item in a (presumably) multiple selection control.
- long GetFocusedItem () const

  Returns the currently focused item or -1 if none.
- long GetNextSelected (long item) const

  Used together with GetFirstSelected() to iterate over all selected items in the control.
- bool IsSelected (long index) const

  Returns true if the item with the given index is selected, false otherwise.
- void Select (long n, bool on=true)

  Selects or unselects the given item.
- void SetColumnImage (int col, int image)

  Sets the column image for the specified column.

Additional Inherited Members

21.428.2 Constructor & Destructor Documentation

wxListView::wxListView ( )

Default constructor.

wxListView::wxListView (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxLC_ICON, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxListCtrlNameStr`)

Constructor, creating and showing a listview control.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxListCtrl.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxListView::~wxListView ( ) [virtual]

Destructor, destroying the listview control.

21.428.3 Member Function Documentation

void wxListView::ClearColumnImage (int *col*)

Resets the column image – after calling this function, no image will be shown.

Parameters

col	the column to clear image for

See also

SetColumnImage()

void wxListView::Focus (long *index*)

Sets focus to the item with the given *index*.

long wxListView::GetFirstSelected ( ) const

Returns the first selected item in a (presumably) multiple selection control.

Together with GetNextSelected() it can be used to iterate over all selected items in the control.

Returns

The first selected item, if any, -1 otherwise.

long wxListView::GetFocusedItem ( ) const

Returns the currently focused item or -1 if none.

See also

IsSelected(), Focus()

long wxListView::GetNextSelected (long *item*) const

Used together with GetFirstSelected() to iterate over all selected items in the control.

Returns

Returns the next selected item or -1 if there are no more of them.

bool wxListView::IsSelected (long *index*) const

Returns true if the item with the given *index* is selected, false otherwise.

See also

GetFirstSelected(), GetNextSelected()

void wxListView::Select (long *n*, bool *on* = `true`)

Selects or unselects the given item.

Parameters

n	the item to select or unselect
on	if true (default), selects the item, otherwise unselects it

See also

wxListCtrl::SetItemState

void wxListView::SetColumnImage (int *col*, int *image*)

Sets the column image for the specified column.

To use the column images, the control must have a valid image list with at least one image.

Parameters

col	the column to set image for
image	the index of the column image in the controls image list

21.429 wxLocale Class Reference

`#include <wx/intl.h>`

21.429.1 Detailed Description

wxLocale class encapsulates all language-dependent settings and is a generalization of the C locale concept.

In wxWidgets this class manages current locale. It also initializes and activates wxTranslations object that manages message catalogs.

For a list of the supported languages, please see wxLanguage enum values. These constants may be used to specify the language in wxLocale::Init and are returned by wxLocale::GetSystemLanguage.

wxPerl Note: In wxPerl you can't use the '_' function name, so the `Wx::Locale` module can export the `gettext` and `gettext_noop` under any given name.

```
# this imports gettext ( equivalent to Wx::GetTranslation
# and gettext_noop ( a noop )
# into your module
use Wx::Locale qw(:default);

# ....

# use the functions
print gettext( "Panic!" );

button = Wx::Button-new( window, -1, gettext( "Label" ) );
```

If you need to translate a lot of strings, then adding gettext( ) around each one is a long task (that is why _( ) was introduced), so just choose a shorter name for gettext:

```
use Wx::Locale 'gettext' = 't',
               'gettext_noop' = 'gettext_noop';

# ...

# use the functions
print t( "Panic!!" );

# ...
```

Library: wxBase

Category: Application and System configuration

See also

Internationalization, Internationalization Sample, wxXLocale, wxTranslations

Public Member Functions

- wxLocale ()

  This is the default constructor and it does nothing to initialize the object: Init() must be used to do that.
- wxLocale (int language, int flags=wxLOCALE_LOAD_DEFAULT)

  See Init() for parameters description.
- wxLocale (const wxString &name, const wxString &shortName=wxEmptyString, const wxString &locale=wxEmptyString, bool bLoadDefault=true)

  See Init() for parameters description.
- virtual ~wxLocale ()

  The destructor, like the constructor, also has global side effects: the previously set locale is restored and so the changes described in Init() documentation are rolled back.
- bool AddCatalog (const wxString &domain)

  Calls wxTranslations::AddCatalog(const wxString&).
- bool AddCatalog (const wxString &domain, wxLanguage msgIdLanguage)

  Calls wxTranslations::AddCatalog(const wxString&, wxLanguage).
- bool AddCatalog (const wxString &domain, wxLanguage msgIdLanguage, const wxString &msgIdCharset)

  Calls wxTranslations::AddCatalog(const wxString&, wxLanguage, const wxString&).
- wxString GetCanonicalName () const

  Returns the canonical form of current locale name.
- wxString GetHeaderValue (const wxString &header, const wxString &domain=wxEmptyString) const

  Calls wxTranslations::GetHeaderValue().
- int GetLanguage () const

  Returns the wxLanguage constant of current language.
- const wxString & GetLocale () const

Returns the locale name as passed to the constructor or Init().

- const wxString & GetName () const

  Returns the current short name for the locale (as given to the constructor or the Init() function).

- virtual const wxString & GetString (const wxString &origString, const wxString &domain=wxEmptyString) const

  Calls wxGetTranslation(const wxString&, const wxString&).

- virtual const wxString & GetString (const wxString &origString, const wxString &origString2, unsigned n, const wxString &domain=wxEmptyString) const

  Calls wxGetTranslation(const wxString&, const wxString&, unsigned, const wxString&).

- wxString GetSysName () const

  Returns current platform-specific locale name as passed to setlocale().

- bool Init (int language=wxLANGUAGE_DEFAULT, int flags=wxLOCALE_LOAD_DEFAULT)

  Initializes the wxLocale instance.

- bool Init (const wxString &name, const wxString &shortName=wxEmptyString, const wxString &locale=wxEmptyString, bool bLoadDefault=true)

- bool IsLoaded (const wxString &domain) const

  Calls wxTranslations::IsLoaded().

- bool IsOk () const

  Returns true if the locale could be set successfully.

Static Public Member Functions

- static void AddCatalogLookupPathPrefix (const wxString &prefix)

  Calls wxFileTranslationsLoader::AddCatalogLookupPathPrefix().

- static void AddLanguage (const wxLanguageInfo &info)

  Adds custom, user-defined language to the database of known languages.

- static const wxLanguageInfo * FindLanguageInfo (const wxString &locale)

  This function may be used to find the language description structure for the given locale, specified either as a two letter ISO language code (for example, "pt"), a language code followed by the country code ("pt_BR") or a full, human readable, language description ("Portuguese-Brazil").

- static const wxLanguageInfo * GetLanguageInfo (int lang)

  Returns a pointer to wxLanguageInfo structure containing information about the given language or NULL if this language is unknown.

- static wxString GetLanguageName (int lang)

  Returns English name of the given language or empty string if this language is unknown.

- static wxString GetLanguageCanonicalName (int lang)

  Returns canonical name (see GetCanonicalName()) of the given language or empty string if this language is unknown.

- static wxFontEncoding GetSystemEncoding ()

  Tries to detect the user's default font encoding.

- static wxString GetSystemEncodingName ()

  Tries to detect the name of the user's default font encoding.

- static int GetSystemLanguage ()

  Tries to detect the user's default locale setting.

- static wxString GetInfo (wxLocaleInfo index, wxLocaleCategory cat=wxLOCALE_CAT_DEFAULT)

  Get the values of the given locale-dependent datum.

- static wxString GetOSInfo (wxLocaleInfo index, wxLocaleCategory cat=wxLOCALE_CAT_DEFAULT)

  Get the values of a locale datum in the OS locale.

- static bool IsAvailable (int lang)

  Check whether the operating system and/or C run time environment supports this locale.

21.429.2 Constructor & Destructor Documentation

wxLocale::wxLocale ( )

This is the default constructor and it does nothing to initialize the object: Init() must be used to do that.

wxLocale::wxLocale (int *language*, int *flags* = `wxLOCALE_LOAD_DEFAULT`)

See Init() for parameters description.

wxLocale::wxLocale (const wxString & *name*, const wxString & *shortName* = wxEmptyString, const wxString & *locale* = wxEmptyString, bool *bLoadDefault* = `true`)

See Init() for parameters description.

The call of this function has several global side effects which you should understand: first of all, the application locale is changed - note that this will affect many of standard C library functions such as printf() or strftime(). Second, this wxLocale object becomes the new current global locale for the application and so all subsequent calls to wxGetTranslation() will try to translate the messages using the message catalogs for this locale.

virtual wxLocale::~wxLocale ( ) `[virtual]`

The destructor, like the constructor, also has global side effects: the previously set locale is restored and so the changes described in Init() documentation are rolled back.

21.429.3 Member Function Documentation

bool wxLocale::AddCatalog (const wxString & *domain*)

Calls wxTranslations::AddCatalog(const wxString&).

bool wxLocale::AddCatalog (const wxString & *domain*, wxLanguage *msgIdLanguage*)

Calls wxTranslations::AddCatalog(const wxString&, wxLanguage).

bool wxLocale::AddCatalog (const wxString & *domain*, wxLanguage *msgIdLanguage*, const wxString & *msgIdCharset*)

Calls wxTranslations::AddCatalog(const wxString&, wxLanguage, const wxString&).

static void wxLocale::AddCatalogLookupPathPrefix (const wxString & *prefix*) `[static]`

Calls wxFileTranslationsLoader::AddCatalogLookupPathPrefix().

static void wxLocale::AddLanguage (const wxLanguageInfo & *info*) `[static]`

Adds custom, user-defined language to the database of known languages.

This database is used in conjunction with the first form of Init().

static const wxLanguageInfo* wxLocale::FindLanguageInfo (const wxString & *locale*) [static]

This function may be used to find the language description structure for the given locale, specified either as a two letter ISO language code (for example, "pt"), a language code followed by the country code ("pt_BR") or a full, human readable, language description ("Portuguese-Brazil").

Returns the information for the given language or NULL if this language is unknown. Note that even if the returned pointer is valid, the caller should *not* delete it.

See also

GetLanguageInfo()

wxString wxLocale::GetCanonicalName ( ) const

Returns the canonical form of current locale name.

Canonical form is the one that is used on UNIX systems: it is a two- or five-letter string in xx or xx_YY format, where xx is ISO 639 code of language and YY is ISO 3166 code of the country. Examples are "en", "en_GB", "en_US" or "fr_FR". This form is internally used when looking up message catalogs. Compare GetSysName().

wxString wxLocale::GetHeaderValue (const wxString & *header,* const wxString & *domain* = wxEmptyString) const

Calls wxTranslations::GetHeaderValue().

static wxString wxLocale::GetInfo (wxLocaleInfo *index,* wxLocaleCategory *cat* = wxLOCALE_CAT_DEFAULT) [static]

Get the values of the given locale-dependent datum.

This function returns the value of the locale-specific option specified by the given *index*.

Parameters

index	One of the elements of wxLocaleInfo enum.
cat	The category to use with the given index or wxLOCALE_CAT_DEFAULT if the index can only apply to a single category.

Returns

The option value or empty string if the function failed.

int wxLocale::GetLanguage ( ) const

Returns the wxLanguage constant of current language.

Note that you can call this function only if you used the form of Init() that takes wxLanguage argument.

static wxString wxLocale::GetLanguageCanonicalName (int *lang*) [static]

Returns canonical name (see GetCanonicalName()) of the given language or empty string if this language is unknown.

See GetLanguageInfo() for a remark about special meaning of `wxLANGUAGE_DEFAULT`.

Since

2.9.1

static const wxLanguageInfo* wxLocale::GetLanguageInfo (int *lang*) `[static]`

Returns a pointer to wxLanguageInfo structure containing information about the given language or NULL if this language is unknown.

Note that even if the returned pointer is valid, the caller should *not* delete it.

See AddLanguage() for the wxLanguageInfo description. As with Init(), `wxLANGUAGE_DEFAULT` has the special meaning if passed as an argument to this function and in this case the result of GetSystemLanguage() is used.

static wxString wxLocale::GetLanguageName (int *lang*) `[static]`

Returns English name of the given language or empty string if this language is unknown.

See GetLanguageInfo() for a remark about special meaning of `wxLANGUAGE_DEFAULT`.

const wxString& wxLocale::GetLocale ( ) const

Returns the locale name as passed to the constructor or Init().

This is a full, human-readable name, e.g. "English" or "French".

const wxString& wxLocale::GetName ( ) const

Returns the current short name for the locale (as given to the constructor or the Init() function).

static wxString wxLocale::GetOSInfo (wxLocaleInfo *index*, wxLocaleCategory *cat* = wxLOCALE_CAT_DEFAULT) `[static]`

Get the values of a locale datum in the OS locale.

This function is similar to GetInfo() and, in fact, identical to it under non-MSW systems. Under MSW it differs from it when no locale had been explicitly set: GetInfo() returns the values corresponding to the "C" locale used by the standard library functions, while this method returns the values used by the OS which, in Windows case, correspond to the user settings in the control panel.

Since

3.1.0

virtual const wxString& wxLocale::GetString (const wxString & *origString*, const wxString & *domain* = wxEmptyString) const `[virtual]`

Calls wxGetTranslation(const wxString&, const wxString&).

virtual const wxString& wxLocale::GetString (const wxString & *origString*, const wxString & *origString2*, unsigned *n*, const wxString & *domain* = wxEmptyString) const `[virtual]`

Calls wxGetTranslation(const wxString&, const wxString&, unsigned, const wxString&).

wxString wxLocale::GetSysName ( ) const

Returns current platform-specific locale name as passed to setlocale().

Compare GetCanonicalName().

static wxFontEncoding wxLocale::GetSystemEncoding ( ) [static]

Tries to detect the user's default font encoding.

Returns wxFontEncoding() value or `wxFONTENCODING_SYSTEM` if it couldn't be determined.

static wxString wxLocale::GetSystemEncodingName ( ) [static]

Tries to detect the name of the user's default font encoding.

This string isn't particularly useful for the application as its form is platform-dependent and so you should probably use GetSystemEncoding() instead.

Returns a user-readable string value or an empty string if it couldn't be determined.

static int wxLocale::GetSystemLanguage ( ) [static]

Tries to detect the user's default locale setting.

Returns the wxLanguage value or `wxLANGUAGE_UNKNOWN` if the language-guessing algorithm failed.

Note

This function works with *locales* and returns the user's default locale. This may be, and usually is, the same as their preferred UI language, but it's not the same thing. Use wxTranslation to obtain *language* information.

See also

wxTranslations::GetBestTranslation().

bool wxLocale::Init (int *language* = wxLANGUAGE_DEFAULT, int *flags* = `wxLOCALE_LOAD_DEFAULT`)

Initializes the wxLocale instance.

The call of this function has several global side effects which you should understand: first of all, the application locale is changed - note that this will affect many of standard C library functions such as printf() or strftime(). Second, this wxLocale object becomes the new current global locale for the application and so all subsequent calls to wxGetTranslation() will try to translate the messages using the message catalogs for this locale.

Parameters

language	wxLanguage identifier of the locale. `wxLANGUAGE_DEFAULT` has special meaning – wxLocale will use system's default language (see GetSystemLanguage()).
flags	Combination of the following: • wxLOCALE_LOAD_DEFAULT: Load the message catalog for the given locale containing the translations of standard wxWidgets messages automatically. • wxLOCALE_DONT_LOAD_DEFAULT: Negation of wxLOCALE_LOAD_DEFAULT.

Returns

true on success or false if the given locale couldn't be set.

bool wxLocale::Init (const wxString & *name*, const wxString & *shortName* = wxEmptyString, const wxString & *locale* = wxEmptyString, bool *bLoadDefault* = `true`)

Deprecated This form is deprecated, use the other one unless you know what you are doing.

Parameters

name	The name of the locale. Only used in diagnostic messages.
shortName	The standard 2 letter locale abbreviation; it is used as the directory prefix when looking for the message catalog files.
locale	The parameter for the call to setlocale(). Note that it is platform-specific.
bLoadDefault	May be set to false to prevent loading of the message catalog for the given locale containing the translations of standard wxWidgets messages. This parameter would be rarely used in normal circumstances.

static bool wxLocale::IsAvailable (int *lang*) `[static]`

Check whether the operating system and/or C run time environment supports this locale.

For example in Windows 2000 and Windows XP, support for many locales is not installed by default. Returns true if the locale is supported.

The argument *lang* is the wxLanguage identifier. To obtain this for a given a two letter ISO language code, use FindLanguageInfo() to obtain its wxLanguageInfo structure. See AddLanguage() for the wxLanguageInfo description.

Since

2.7.1.

bool wxLocale::IsLoaded (const wxString & *domain*) const

Calls wxTranslations::IsLoaded().

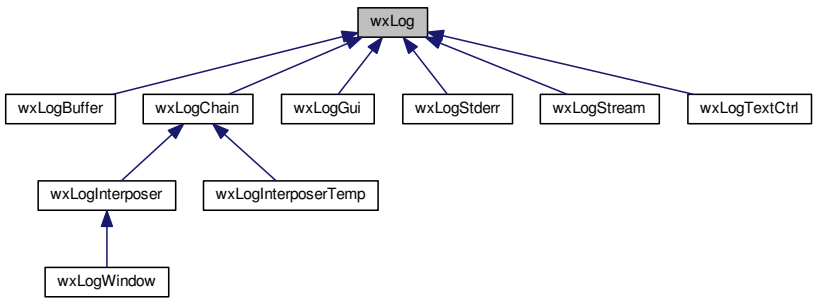
bool wxLocale::IsOk ( ) const

Returns true if the locale could be set successfully.

21.430 wxLog Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLog:



21.430.1 Detailed Description

wxLog class defines the interface for the *log targets* used by wxWidgets logging functions as explained in the Logging Overview.

The only situations when you need to directly use this class is when you want to derive your own log target because the existing ones don't satisfy your needs.

Otherwise, it is completely hidden behind the wxLogXXX() functions and you may not even know about its existence.

Note

For console-mode applications, the default target is wxLogStderr, so that all *wxLogXXX()* functions print on `stderr` when `wxUSE_GUI` = 0.

Library: wxBase

Category: Logging

See also

Logging Overview, wxLogXXX() functions

Public Member Functions

- wxLogFormatter * SetFormatter (wxLogFormatter *formatter)

  Sets the specified formatter as the active one.
- virtual void Flush ()

  Show all pending output and clear the buffer.
- void LogRecord (wxLogLevel level, const wxString &msg, const wxLogRecordInfo &info)

  Log the given record.

Static Public Member Functions

Trace mask functions

- static void AddTraceMask (const wxString &mask)

  Add the mask to the list of allowed masks for wxLogTrace().
- static void ClearTraceMasks ()

  Removes all trace masks previously set with AddTraceMask().
- static const wxArrayString & GetTraceMasks ()

  Returns the currently allowed list of string trace masks.
- static bool IsAllowedTraceMask (const wxString &mask)

  Returns true if the mask is one of allowed masks for wxLogTrace().
- static void RemoveTraceMask (const wxString &mask)

  Remove the mask from the list of allowed masks for wxLogTrace().

Log target functions

- static void DontCreateOnDemand ()

  Instructs wxLog to not create new log targets on the fly if there is none currently (see GetActiveTarget()).
- static wxLog * GetActiveTarget ()

  Returns the pointer to the active log target (may be NULL).
- static wxLog * SetActiveTarget (wxLog *logtarget)

  Sets the specified log target as the active one.
- static wxLog * SetThreadActiveTarget (wxLog *logger)

Sets a thread-specific log target.

- static void FlushActive ()

  Flushes the current log target if any, does nothing if there is none.
- static void Resume ()

  Resumes logging previously suspended by a call to Suspend().
- static void Suspend ()

  Suspends the logging until Resume() is called.

Log level functions

- static wxLogLevel GetLogLevel ()

  Returns the current log level limit.
- static bool IsLevelEnabled (wxLogLevel level, wxString component)

  Returns true if logging at this level is enabled for the current thread.
- static void SetComponentLevel (const wxString &component, wxLogLevel level)

  Sets the log level for the given component.
- static void SetLogLevel (wxLogLevel logLevel)

  Specifies that log messages with level greater (numerically) than logLevel should be ignored and not sent to the active log target.

Enable/disable features functions

- static bool EnableLogging (bool enable=true)

  Globally enable or disable logging.
- static bool IsEnabled ()

  Returns true if logging is enabled at all now.
- static bool GetRepetitionCounting ()

  Returns whether the repetition counting mode is enabled.
- static void SetRepetitionCounting (bool repetCounting=true)

  Enables logging mode in which a log message is logged once, and in case exactly the same message successively repeats one or more times, only the number of repetitions is logged.
- static const wxString & GetTimestamp ()

  Returns the current timestamp format string.
- static void SetTimestamp (const wxString &format)

  Sets the timestamp format prepended by the default log targets to all messages.
- static void DisableTimestamp ()

  Disables time stamping of the log messages.
- static bool GetVerbose ()

  Returns whether the verbose mode is currently active.
- static void SetVerbose (bool verbose=true)

  Activates or deactivates verbose mode in which the verbose messages are logged as the normal ones instead of being silently dropped.

Protected Member Functions

Logging callbacks.

The functions which should be overridden by custom log targets.

When defining a new log target, you have a choice between overriding DoLogRecord(), which provides maximal flexibility, DoLogTextAtLevel() which can be used if you don't intend to change the default log messages formatting but want to handle log messages of different levels differently or, in the simplest case, DoLogText().

- virtual void DoLogRecord (wxLogLevel level, const wxString &msg, const wxLogRecordInfo &info)

  Called to log a new record.
- virtual void DoLogTextAtLevel (wxLogLevel level, const wxString &msg)

  Called to log the specified string at given level.
- virtual void DoLogText (const wxString &msg)

  Called to log the specified string.

21.430.2 Member Function Documentation

static void wxLog::AddTraceMask (const wxString & *mask*) `[static]`

Add the *mask* to the list of allowed masks for wxLogTrace().

See also

RemoveTraceMask(), GetTraceMasks()

static void wxLog::ClearTraceMasks ( ) `[static]`

Removes all trace masks previously set with AddTraceMask().

See also

RemoveTraceMask()

static void wxLog::DisableTimestamp ( ) `[static]`

Disables time stamping of the log messages.

Notice that the current time stamp is only used by the default log formatter and custom formatters may ignore calls to this function.

Since

2.9.0

virtual void wxLog::DoLogRecord (wxLogLevel *level*, const wxString & *msg*, const wxLogRecordInfo & *info*) `[protected]`, `[virtual]`

Called to log a new record.

Any log message created by wxLogXXX() functions is passed to this method of the active log target. The default implementation prepends the timestamp and, for some log levels (e.g. error and warning), the corresponding prefix to *msg* and passes it to DoLogTextAtLevel().

You may override this method to implement custom formatting of the log messages or to implement custom filtering of log messages (e.g. you could discard all log messages coming from the given source file).

virtual void wxLog::DoLogText (const wxString & *msg*) `[protected]`, `[virtual]`

Called to log the specified string.

A simple implementation might just send the string to `stdout` or `stderr` or save it in a file (of course, the already existing wxLogStderr can be used for this).

The base class version of this function asserts so it must be overridden if you don't override DoLogRecord() or DoLogTextAtLevel().

virtual void wxLog::DoLogTextAtLevel (wxLogLevel *level*, const wxString & *msg*) `[protected]`, `[virtual]`

Called to log the specified string at given level.

The base class versions logs debug and trace messages on the system default debug output channel and passes all the other messages to DoLogText().

static void wxLog::DontCreateOnDemand ( ) [static]

Instructs wxLog to not create new log targets on the fly if there is none currently (see GetActiveTarget()).

(Almost) for internal use only: it is supposed to be called by the application shutdown code (where you don't want the log target to be automatically created anymore).

Note that this function also calls ClearTraceMasks().

static bool wxLog::EnableLogging (bool *enable* = true) [static]

Globally enable or disable logging.

Calling this function with false argument disables all log messages for the current thread.

See also

wxLogNull, IsEnabled()

Returns

The old state, i.e. true if logging was previously enabled and false if it was disabled.

virtual void wxLog::Flush ( ) [virtual]

Show all pending output and clear the buffer.

Some of wxLog implementations, most notably the standard wxLogGui class, buffer the messages (for example, to avoid showing the user a zillion of modal message boxes one after another – which would be really annoying). This function shows them all and clears the buffer contents. If the buffer is already empty, nothing happens.

If you override this method in a derived class, call the base class version first, before doing anything else.

Reimplemented in wxLogGui, and wxLogBuffer.

static void wxLog::FlushActive ( ) [static]

Flushes the current log target if any, does nothing if there is none.

When this method is called from the main thread context, it also flushes any previously buffered messages logged by the other threads. When it is called from the other threads it simply calls Flush() on the currently active log target, so it mostly makes sense to do this if a thread has its own logger set with SetThreadActiveTarget().

static wxLog* wxLog::GetActiveTarget ( ) [static]

Returns the pointer to the active log target (may be NULL).

Notice that if SetActiveTarget() hadn't been previously explicitly called, this function will by default try to create a log target by calling wxAppTraits::CreateLogTarget() which may be overridden in a user-defined traits class to change the default behaviour. You may also call DontCreateOnDemand() to disable this behaviour.

When this function is called from threads other than main one, auto-creation doesn't happen. But if the thread has a thread-specific log target previously set by SetThreadActiveTarget(), it is returned instead of the global one. Otherwise, the global log target is returned.

static wxLogLevel wxLog::GetLogLevel ( ) [static]

Returns the current log level limit.

All messages at levels strictly greater than the value returned by this function are not logged at all.

See also

SetLogLevel(), IsLevelEnabled()

static bool wxLog::GetRepetitionCounting ( ) [static]

Returns whether the repetition counting mode is enabled.

static const wxString& wxLog::GetTimestamp ( ) [static]

Returns the current timestamp format string.

Notice that the current time stamp is only used by the default log formatter and custom formatters may ignore this format.

static const wxArrayString& wxLog::GetTraceMasks ( ) [static]

Returns the currently allowed list of string trace masks.

See also

AddTraceMask().

static bool wxLog::GetVerbose ( ) [static]

Returns whether the verbose mode is currently active.

static bool wxLog::IsAllowedTraceMask (const wxString & *mask*) [static]

Returns true if the *mask* is one of allowed masks for wxLogTrace().

See also: AddTraceMask(), RemoveTraceMask()

static bool wxLog::IsEnabled ( ) [static]

Returns true if logging is enabled at all now.

See also

IsLevelEnabled(), EnableLogging()

static bool wxLog::IsLevelEnabled (wxLogLevel *level,* wxString *component*) [static]

Returns true if logging at this level is enabled for the current thread.

This function only returns true if logging is globally enabled and if *level* is less than or equal to the maximal log level enabled for the given *component*.

See also

IsEnabled(), SetLogLevel(), GetLogLevel(), SetComponentLevel()

Since

2.9.1

void wxLog::LogRecord (wxLogLevel *level,* const wxString & *msg,* const wxLogRecordInfo & *info*)

Log the given record.

This function should only be called from the DoLog() implementations in the derived classes if they need to call DoLogRecord() on another log object (they can, of course, just use wxLog::DoLogRecord() call syntax to call it on the object itself). It should not be used for logging new messages which can be only sent to the currently active logger using OnLog() which also checks if the logging (for this level) is enabled while this method just directly calls DoLog().

Example of use of this class from wxLogChain:

```
void wxLogChain::DoLogRecord(wxLogLevel level,
                             const wxString& msg,
                             const wxLogRecordInfo& info)
{
    // let the previous logger show it
    if ( m_logOld && IsPassingMessages() )
        m_logOld->LogRecord(level, msg, info);

    // and also send it to the new one
    if ( m_logNew && m_logNew != this )
        m_logNew->LogRecord(level, msg, info);
}
```

Since

2.9.1

static void wxLog::RemoveTraceMask (const wxString & *mask*) `[static]`

Remove the *mask* from the list of allowed masks for wxLogTrace().

See also

AddTraceMask()

static void wxLog::Resume ( ) `[static]`

Resumes logging previously suspended by a call to Suspend().

All messages logged in the meanwhile will be flushed soon.

static wxLog* wxLog::SetActiveTarget (wxLog * *logtarget*) `[static]`

Sets the specified log target as the active one.

Returns the pointer to the previous active log target (may be NULL). To suppress logging use a new instance of wxLogNull not NULL. If the active log target is set to NULL a new default log target will be created when logging occurs.

See also

SetThreadActiveTarget()

static void wxLog::SetComponentLevel (const wxString & *component,* wxLogLevel *level*) `[static]`

Sets the log level for the given component.

For example, to disable all but error messages from wxWidgets network classes you may use

```
wxLog::SetComponentLevel("wx/net", wxLOG_Error);
```

SetLogLevel() may be used to set the global log level.

Parameters

component	Non-empty component name, possibly using slashes (`/`) to separate it into several parts.
level	Maximal level of log messages from this component which will be handled instead of being simply discarded.

Since

2.9.1

wxLogFormatter* wxLog::SetFormatter (wxLogFormatter * *formatter*)

Sets the specified formatter as the active one.

Parameters

formatter	The new formatter. If NULL, reset to the default formatter.

Returns the pointer to the previous formatter. You must delete it if you don't plan to attach it again to a wxLog object later.

Since

2.9.4

static void wxLog::SetLogLevel (wxLogLevel *logLevel*) [static]

Specifies that log messages with level greater (numerically) than *logLevel* should be ignored and not sent to the active log target.

See also

SetComponentLevel()

static void wxLog::SetRepetitionCounting (bool *repetCounting* = `true`) [static]

Enables logging mode in which a log message is logged once, and in case exactly the same message successively repeats one or more times, only the number of repetitions is logged.

static wxLog* wxLog::SetThreadActiveTarget (wxLog * *logger*) [static]

Sets a thread-specific log target.

The log target passed to this function will be used for all messages logged by the current thread using the usual wxLog functions. This shouldn't be called from the main thread which never uses a thread- specific log target but can be used for the other threads to handle thread logging completely separately; instead of buffering thread log messages in the main thread logger.

Notice that unlike for SetActiveTarget(), wxWidgets does not destroy the thread-specific log targets when the thread terminates so doing this is your responsibility.

This method is only available if `wxUSE_THREADS` is 1, i.e. wxWidgets was compiled with threads support.

Parameters

logger	The new thread-specific log target, possibly NULL.

Returns

The previous thread-specific log target, initially NULL.

Since

2.9.1

static void wxLog::SetTimestamp (const wxString & *format*) `[static]`

Sets the timestamp format prepended by the default log targets to all messages.

The string may contain any normal characters as well as % prefixed format specifiers, see *strftime()* manual for details. Passing an empty string to this function disables message time stamping.

Notice that the current time stamp is only used by the default log formatter and custom formatters may ignore this format. You can also define a custom wxLogFormatter to customize the time stamp handling beyond changing its format.

static void wxLog::SetVerbose (bool *verbose* = `true`) `[static]`

Activates or deactivates verbose mode in which the verbose messages are logged as the normal ones instead of being silently dropped.

The verbose messages are the trace messages which are not disabled in the release mode and are generated by wxLogVerbose().

See also

Logging Overview

static void wxLog::Suspend ( ) `[static]`

Suspends the logging until Resume() is called.

Note that the latter must be called the same number of times as the former to undo it, i.e. if you call Suspend() twice you must call Resume() twice as well.

Note that suspending the logging means that the log sink won't be flushed periodically, it doesn't have any effect if the current log target does the logging immediately without waiting for Flush() to be called (the standard GUI log target only shows the log dialog when it is flushed, so Suspend() works as expected with it).

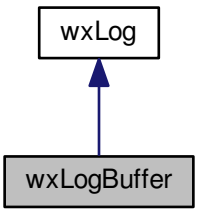
See also

Resume(), wxLogNull

21.431 wxLogBuffer Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogBuffer:



21.431.1 Detailed Description

wxLogBuffer is a very simple implementation of log sink which simply collects all the logged messages in a string (except the debug messages which are output in the usual way immediately as we're presumably not interested in collecting them for later).

The messages from different log function calls are separated by the new lines.

All the messages collected so far can be shown to the user (and the current buffer cleared) by calling the overloaded wxLogBuffer::Flush method.

Library: wxBase

Category: Logging

Public Member Functions

- wxLogBuffer ()

  The default ctor does nothing.

- virtual void Flush ()

  Shows all the messages collected so far to the user (using a message box in the GUI applications or by printing them out to the console in text mode) and clears the internal buffer.

- const wxString & GetBuffer () const

  Returns the current buffer contains.

Additional Inherited Members

21.431.2 Constructor & Destructor Documentation

wxLogBuffer::wxLogBuffer ( )

The default ctor does nothing.

21.431.3 Member Function Documentation

virtual void wxLogBuffer::Flush ( ) [virtual]

Shows all the messages collected so far to the user (using a message box in the GUI applications or by printing them out to the console in text mode) and clears the internal buffer.

Reimplemented from wxLog.

const wxString& wxLogBuffer::GetBuffer ( ) const

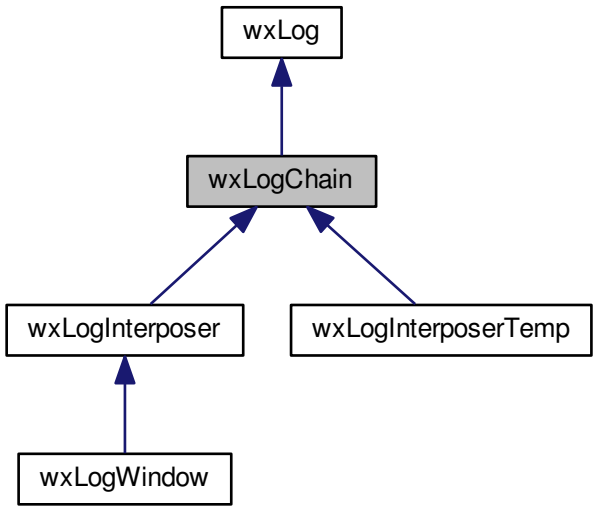
Returns the current buffer contains.

Messages from different log function calls are separated with the new lines in the buffer. The buffer can be cleared by Flush() which will also show the current contents to the user.

21.432 wxLogChain Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogChain:



21.432.1 Detailed Description

This simple class allows you to chain log sinks, that is to install a new sink but keep passing log messages to the old one instead of replacing it completely as wxLog::SetActiveTarget does.

It is especially useful when you want to divert the logs somewhere (for example to a file or a log window) but also keep showing the error messages using the standard dialogs as wxLogGui does by default.

Example of usage:

```
wxLogChain *logChain = new wxLogChain(new wxLogStderr);
```

```
// all the log messages are sent to stderr and also processed as usually
...

// don't delete logChain directly as this would leave a dangling
// pointer as active log target, use SetActiveTarget() instead
delete wxLog::SetActiveTarget(...something else or NULL...);
```

Library: wxBase

Category: Logging

Public Member Functions

- wxLogChain (wxLog *logger)

  Sets the specified `logger` (which may be NULL) as the default log target but the log messages are also passed to the previous log target if any.
- virtual ~wxLogChain ()

  Destroys the previous log target.
- void DetachOldLog ()

  Detaches the old log target so it won't be destroyed when the wxLogChain object is destroyed.
- wxLog * GetOldLog () const

  Returns the pointer to the previously active log target (which may be NULL).
- bool IsPassingMessages () const

  Returns true if the messages are passed to the previously active log target (default) or false if PassMessages() had been called.
- void PassMessages (bool passMessages)

  By default, the log messages are passed to the previously active log target.
- void SetLog (wxLog *logger)

  Sets another log target to use (may be NULL).

Additional Inherited Members

21.432.2 Constructor & Destructor Documentation

wxLogChain::wxLogChain (wxLog * *logger*)

Sets the specified `logger` (which may be NULL) as the default log target but the log messages are also passed to the previous log target if any.

virtual wxLogChain::~wxLogChain ( ) [virtual]

Destroys the previous log target.

21.432.3 Member Function Documentation

void wxLogChain::DetachOldLog ( )

Detaches the old log target so it won't be destroyed when the wxLogChain object is destroyed.

wxLog* wxLogChain::GetOldLog ( ) const

Returns the pointer to the previously active log target (which may be NULL).

bool wxLogChain::IsPassingMessages ( ) const

Returns true if the messages are passed to the previously active log target (default) or false if PassMessages() had been called.

void wxLogChain::PassMessages (bool *passMessages*)

By default, the log messages are passed to the previously active log target.

Calling this function with false parameter disables this behaviour (presumably temporarily, as you shouldn't use wxLogChain at all otherwise) and it can be reenabled by calling it again with *passMessages* set to true.

void wxLogChain::SetLog (wxLog * *logger*)

Sets another log target to use (may be NULL).

The log target specified in the wxLogChain(wxLog*) constructor or in a previous call to this function is deleted. This doesn't change the old log target value (the one the messages are forwarded to) which still remains the same as was active when wxLogChain object was created.

21.433 wxLogFormatter Class Reference

`#include <wx/log.h>`

21.433.1 Detailed Description

wxLogFormatter class is used to format the log messages.

It implements the default formatting and can be derived from to create custom formatters.

The default implementation formats the message into a string containing the time stamp, level-dependent prefix and the message itself.

To change it, you can derive from it and override its Format() method. For example, to include the thread id in the log messages you can use

```
class LogFormatterWithThread : public wxLogFormatter
{
    virtual wxString Format(wxLogLevel level,
                            const wxString& msg,
                            const wxLogRecordInfo& info) const
    {
        return wxString::Format("[%d] %s(%d) : %s",
            info.threadId, info.filename, info.line, msg);
    }
};
```

And then associate it with wxLog instance using its SetFormatter(). Then, if you call:

```
wxLogMessage(_("*** Application started ***"));
```

the log output could be something like:

```
    [7872] d:\testApp\src\testApp.cpp(85) : *** Application started ***
```

Library: wxBase

Category: Logging

See also

Logging Overview

Since

2.9.4

Public Member Functions

- wxLogFormatter ()

  The default ctor does nothing.
- virtual wxString Format (wxLogLevel level, const wxString &msg, const wxLogRecordInfo &info) const

  This function creates the full log message string.

Protected Member Functions

- virtual wxString FormatTime (time_t time) const

  This function formats the time stamp part of the log message.

21.433.2 Constructor & Destructor Documentation

wxLogFormatter::wxLogFormatter ( )

The default ctor does nothing.

21.433.3 Member Function Documentation

virtual wxString wxLogFormatter::Format (wxLogLevel *level,* const wxString & *msg,* const wxLogRecordInfo & *info*) const `[virtual]`

This function creates the full log message string.

Override it to customize the output string format.

Parameters

level	The level of this log record, e.g. wxLOG_Error.
msg	The log message itself.
info	All the other information (such as time, component, location...) associated with this log record.

Returns

The formated message.

Note

Time stamping is disabled for Visual C++ users in debug builds by default because otherwise it would be impossible to directly go to the line from which the log message was generated by simply clicking in the debugger window on the corresponding error message. If you wish to enable it, override FormatTime().

virtual wxString wxLogFormatter::FormatTime (time_t *time*) const `[protected], [virtual]`

This function formats the time stamp part of the log message.

Override this function if you need to customize just the time stamp.

Parameters

time	Time to format.

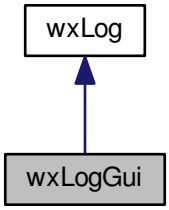
Returns

The formated time string, may be empty.

21.434 wxLogGui Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogGui:



21.434.1 Detailed Description

This is the default log target for the GUI wxWidgets applications.

Please see Logging Customization for explanation of how to change the default log target.

An object of this class is used by default to show the log messages created by using wxLogMessage(), wxLogError() and other logging functions. It doesn't display the messages logged by them immediately however but accumulates all messages logged during an event handler execution and then shows them all at once when its Flush() method is called during the idle time processing. This has the important advantage of showing only a single dialog to the user even if several messages were logged because of a single error as it often happens (e.g. a low level function could log a message because it failed to open a file resulting in its caller logging another message due to the failure of higher level operation requiring the use of this file). If you need to force the display of all previously logged messages immediately you can use wxLog::FlushActive() to force the dialog display.

Also notice that if an error message is logged when several informative messages had been already logged before, the informative messages are discarded on the assumption that they are not useful – and may be confusing and hence harmful – any more after the error. The warning and error messages are never discarded however and any informational messages logged after the first error one are also kept (as they may contain information about the error recovery). You may override DoLog() method to change this behaviour.

At any rate, it is possible that that several messages were accumulated before this class Flush() method is called. If this is the case, Flush() uses a custom dialog which shows the last message directly and allows the user to view the previously logged ones by expanding the "Details" wxCollapsiblePane inside it. This custom dialog also provides the buttons for copying the log messages to the clipboard and saving them to a file.

However if only a single message is present when Flush() is called, just a wxMessageBox() is used to show it. This has the advantage of being closer to the native behaviour but it doesn't give the user any possibility to copy or save the message (except for the recent Windows versions where `Ctrl-C` may be pressed in the message box to copy its contents to the clipboard) so you may want to override DoShowSingleLogMessage() to customize wxLogGui – the dialogs sample shows how to do this.

Library: wxCore

Category: Logging

Public Member Functions

- wxLogGui ()

  Default constructor.
- virtual void Flush ()

  Presents the accumulated log messages, if any, to the user.

Protected Member Functions

- wxString GetTitle () const

  Returns the appropriate title for the dialog.
- int GetSeverityIcon () const

  Returns wxICON_ERROR, wxICON_WARNING or wxICON_INFORMATION depending on the current maximal severity.
- void Clear ()

  Forgets all the currently stored messages.

Protected Attributes

- wxArrayString m_aMessages

  All currently accumulated messages.
- wxArrayInt m_aSeverity

  The severities of each logged message.
- wxArrayLong m_aTimes

  The time stamps of each logged message.
- bool m_bErrors

  True if there any error messages.
- bool m_bWarnings

  True if there any warning messages.
- bool m_bHasMessages

  True if there any messages to be shown to the user.

Private Member Functions

- virtual void DoShowSingleLogMessage (const wxString &message, const wxString &title, int style)

  Method called by Flush() to show a single log message.
- virtual void DoShowMultipleLogMessages (const wxArrayString &messages, const wxArrayInt &severities, const wxArrayLong &times, const wxString &title, int style)

  Method called by Flush() to show multiple log messages.

Additional Inherited Members

21.434.2 Constructor & Destructor Documentation

wxLogGui::wxLogGui ( )

Default constructor.

21.434.3 Member Function Documentation

void wxLogGui::Clear ( ) `[protected]`

Forgets all the currently stored messages.

If you override Flush() (and don't call the base class version), you must call this method to avoid messages being logged over and over again.

virtual void wxLogGui::DoShowMultipleLogMessages (const wxArrayString & *messages*, const wxArrayInt & *severities*, const wxArrayLong & *times*, const wxString & *title*, int *style*) `[private], [virtual]`

Method called by Flush() to show multiple log messages.

This function can be overridden to show the messages in a different way. By default a special log dialog showing the most recent message and allowing the user to expand it to view the previously logged ones is used.

Parameters

messages	Array of messages to show; it contains more than one element.
severities	Array of message severities containing `wxLOG_XXX` values.
times	Array of time_t values indicating when each message was logged.
title	The suggested title for the dialog showing the message, see GetTitle().
style	One of `wxICON_XXX` constants, see GetSeverityIcon().

virtual void wxLogGui::DoShowSingleLogMessage (const wxString & *message*, const wxString & *title*, int *style*) `[private], [virtual]`

Method called by Flush() to show a single log message.

This function can be overridden to show the message in a different way. By default a simple wxMessageBox() call is used.

Parameters

message	The message to show (it can contain multiple lines).
title	The suggested title for the dialog showing the message, see GetTitle().
style	One of `wxICON_XXX` constants, see GetSeverityIcon().

virtual void wxLogGui::Flush ( ) `[virtual]`

Presents the accumulated log messages, if any, to the user.

This method is called during the idle time and should show any messages accumulated in wxLogGui::m_aMessages field to the user.

Reimplemented from wxLog.

int wxLogGui::GetSeverityIcon ( ) const [protected]

Returns wxICON_ERROR, wxICON_WARNING or wxICON_INFORMATION depending on the current maximal severity.

This value is suitable to be used in the style parameter of wxMessageBox() function.

wxString wxLogGui::GetTitle ( ) const [protected]

Returns the appropriate title for the dialog.

The title is constructed from wxApp::GetAppDisplayName() and the severity string (e.g. "error" or "warning") appropriate for the current wxLogGui::m_bErrors and wxLogGui::m_bWarnings values.

21.434.4 Member Data Documentation

wxArrayString wxLogGui::m_aMessages [protected]

All currently accumulated messages.

This array may be empty if no messages were logged.

See also

m_aSeverity, m_aTimes

wxArrayInt wxLogGui::m_aSeverity [protected]

The severities of each logged message.

This array is synchronized with wxLogGui::m_aMessages, i.e. the n-th element of this array corresponds to the severity of the n-th message. The possible severity values are wxLOG_XXX constants, e.g. wxLOG_Error, wxLOG_Warning, wxLOG_Message etc.

wxArrayLong wxLogGui::m_aTimes [protected]

The time stamps of each logged message.

The elements of this array are time_t values corresponding to the time when the message was logged.

bool wxLogGui::m_bErrors [protected]

True if there any error messages.

bool wxLogGui::m_bHasMessages [protected]

True if there any messages to be shown to the user.

This variable is used instead of simply checking whether wxLogGui::m_aMessages array is empty to allow blocking further calls to Flush() while a log dialog is already being shown, even if the messages array hasn't been emptied yet.

bool wxLogGui::m_bWarnings [protected]

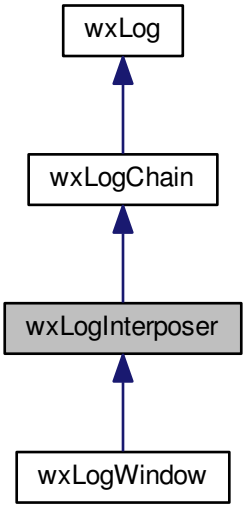
True if there any warning messages.

If both wxLogGui::m_bErrors and this member are false, there are only informational messages to be shown.

21.435 wxLogInterposer Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogInterposer:



21.435.1 Detailed Description

A special version of wxLogChain which uses itself as the new log target.

It forwards log messages to the previously installed one in addition to processing them itself.

Unlike wxLogChain which is usually used directly as is, this class must be derived from to implement wxLog::DoLog and/or wxLog::DoLogString methods.

wxLogInterposer destroys the previous log target in its destructor. If you don't want this to happen, use wxLog↩ InterposerTemp instead.

Library: wxBase

Category: Logging

Public Member Functions

- wxLogInterposer ()

  The default constructor installs this object as the current active log target.

Additional Inherited Members

21.435.2 Constructor & Destructor Documentation

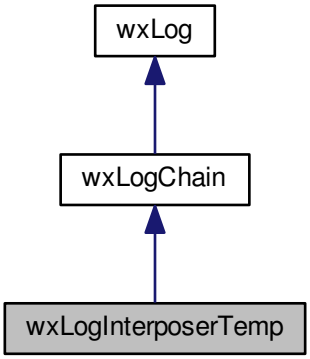
wxLogInterposer::wxLogInterposer ( )

The default constructor installs this object as the current active log target.

21.436 wxLogInterposerTemp Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogInterposerTemp:



21.436.1 Detailed Description

A special version of wxLogChain which uses itself as the new log target.

It forwards log messages to the previously installed one in addition to processing them itself. Unlike wxLog↩ Interposer, it doesn't delete the old target which means it can be used to temporarily redirect log output.

As per wxLogInterposer, this class must be derived from to implement wxLog::DoLog and/or wxLog::DoLogString methods.

Library: wxBase

Category: Logging

Public Member Functions

- wxLogInterposerTemp ()

  The default constructor installs this object as the current active log target.

Additional Inherited Members

21.436.2 Constructor & Destructor Documentation

wxLogInterposerTemp::wxLogInterposerTemp ( )

The default constructor installs this object as the current active log target.

21.437 wxLogNull Class Reference

`#include <wx/log.h>`

21.437.1 Detailed Description

This class allows you to temporarily suspend logging.

All calls to the log functions during the life time of an object of this class are just ignored.

In particular, it can be used to suppress the log messages given by wxWidgets itself but it should be noted that it is rarely the best way to cope with this problem as **all** log messages are suppressed, even if they indicate a completely different error than the one the programmer wanted to suppress.

For instance, the example of the overview:

```
wxFile file;

// wxFile.Open() normally complains if file can't be opened, we don't want it
{
  wxLogNull logNo;
  if ( !file.Open("bar") )
    ... process error ourselves ...
} // ~wxLogNull called, old log sink restored

wxLogMessage("..."); // ok
```

would be better written as:

```
wxFile file;

// don't try to open file if it doesn't exist, we are prepared to deal with
// this ourselves - but all other errors are not expected
if ( wxFile::Exists("bar") )
{
    // gives an error message if the file couldn't be opened
    file.Open("bar");
}
else
{
    ...
}
```

Library: wxBase

Category: Logging

Public Member Functions

- wxLogNull ()

  Suspends logging.

- ~wxLogNull ()

  Resumes logging.

21.437.2 Constructor & Destructor Documentation

wxLogNull::wxLogNull ( )

Suspends logging.

wxLogNull::~wxLogNull ( )

Resumes logging.

21.438 wxLogRecordInfo Class Reference

`#include <wx/log.h>`

21.438.1 Detailed Description

Information about a log record (unit of the log output).

Public Attributes

- const char * filename

  The name of the file where this log message was generated.
- int line

  The line number at which this log message was generated.
- const char * func

  The name of the function where the log record was generated.
- time_t timestamp

  Time when the log message was generated.
- wxThreadIdType threadId

  Id of the thread in which the message was generated.

21.438.2 Member Data Documentation

const char* wxLogRecordInfo::filename

The name of the file where this log message was generated.

const char* wxLogRecordInfo::func

The name of the function where the log record was generated.

This field may be NULL if the compiler doesn't support **`__FUNCTION__`** (but most modern compilers do).

int wxLogRecordInfo::line

The line number at which this log message was generated.

wxThreadIdType wxLogRecordInfo::threadId

Id of the thread in which the message was generated.

This field is only available if wxWidgets was built with threads support (`wxUSE_THREADS == 1`).

See also

wxThread::GetCurrentId()

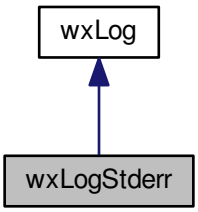
time_t wxLogRecordInfo::timestamp

Time when the log message was generated.

21.439 wxLogStderr Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogStderr:



21.439.1 Detailed Description

This class can be used to redirect the log messages to a C file stream (not to be confused with C++ streams).

It is the default log target for the non-GUI wxWidgets applications which send all the output to `stderr`.

Library: wxBase

Category: Logging

See also

wxLogStream

Public Member Functions

- wxLogStderr (FILE *fp=NULL)

  Constructs a log target which sends all the log messages to the given `FILE`.

Additional Inherited Members

21.439.2 Constructor & Destructor Documentation

wxLogStderr::wxLogStderr (FILE * *fp* = NULL)

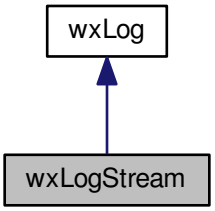
Constructs a log target which sends all the log messages to the given FILE.

If it is NULL, the messages are sent to stderr.

21.440 wxLogStream Class Reference

#include <wx/log.h>

Inheritance diagram for wxLogStream:



21.440.1 Detailed Description

This class can be used to redirect the log messages to a C++ stream.

Please note that this class is only available if wxWidgets was compiled with the standard iostream library support (wxUSE_STD_IOSTREAM must be on).

Library: wxBase

Category: Logging

See also

wxLogStderr, wxStreamToTextRedirector

Public Member Functions

- wxLogStream (std::ostream *ostr=NULL)

  Constructs a log target which sends all the log messages to the given output stream.

Additional Inherited Members

21.440.2 Constructor & Destructor Documentation

wxLogStream::wxLogStream (std::ostream * *ostr* = NULL)

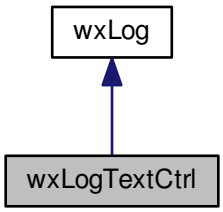
Constructs a log target which sends all the log messages to the given output stream.

If it is NULL, the messages are sent to `cerr`.

21.441 wxLogTextCtrl Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogTextCtrl:



21.441.1 Detailed Description

Using these target all the log messages can be redirected to a text control.

The text control must have been created with `wxTE_MULTILINE` style by the caller previously.

Library: wxCore

Category: Logging

See also

wxTextCtrl, wxStreamToTextRedirector

Public Member Functions

- wxLogTextCtrl (wxTextCtrl *pTextCtrl)

  Constructs a log target which sends all the log messages to the given text control.

Additional Inherited Members

21.441.2 Constructor & Destructor Documentation

wxLogTextCtrl::wxLogTextCtrl (wxTextCtrl * *pTextCtrl*)

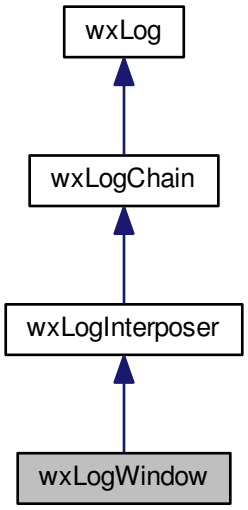
Constructs a log target which sends all the log messages to the given text control.

The *textctrl* parameter cannot be NULL.

21.442 wxLogWindow Class Reference

`#include <wx/log.h>`

Inheritance diagram for wxLogWindow:



21.442.1 Detailed Description

This class represents a background log window: to be precise, it collects all log messages in the log frame which it manages but also passes them on to the log target which was active at the moment of its creation.

This allows you, for example, to show all the log messages in a frame but still continue to process them normally by showing the standard log dialog.

Library: wxCore

Category: Logging

See also

wxLogTextCtrl

Public Member Functions

- wxLogWindow (wxWindow *pParent, const wxString &szTitle, bool show=true, bool passToOld=true)

Creates the log frame window and starts collecting the messages in it.

- wxFrame * GetFrame () const

  Returns the associated log frame window.

- virtual bool OnFrameClose (wxFrame *frame)

  Called if the user closes the window interactively, will not be called if it is destroyed for another reason (such as when program exits).

- virtual void OnFrameDelete (wxFrame *frame)

  Called right before the log frame is going to be deleted: will always be called unlike OnFrameClose().

- void Show (bool show=true)

  Shows or hides the frame.

Additional Inherited Members

21.442.2 Constructor & Destructor Documentation

wxLogWindow::wxLogWindow (wxWindow * *pParent*, const wxString & *szTitle*, bool *show* = `true`, bool *passToOld* = `true`)

Creates the log frame window and starts collecting the messages in it.

Parameters

pParent	The parent window for the log frame, may be NULL
szTitle	The title for the log frame
show	true to show the frame initially (default), otherwise Show() must be called later.
passToOld	true to process the log messages normally in addition to logging them in the log frame (default), false to only log them in the log frame. Note that if no targets were set using wxLog::SetActiveTarget() then wxLogWindow simply becomes the active one and messages won't be passed to other targets.

21.442.3 Member Function Documentation

wxFrame* wxLogWindow::GetFrame ( ) const

Returns the associated log frame window.

This may be used to position or resize it but use Show() to show or hide it.

virtual bool wxLogWindow::OnFrameClose (wxFrame * *frame*) `[virtual]`

Called if the user closes the window interactively, will not be called if it is destroyed for another reason (such as when program exits).

Return true from here to allow the frame to close, false to prevent this from happening.

See also

OnFrameDelete()

virtual void wxLogWindow::OnFrameDelete (wxFrame * *frame*) `[virtual]`

Called right before the log frame is going to be deleted: will always be called unlike OnFrameClose().

void wxLogWindow::Show (bool *show* = `true`)

Shows or hides the frame.

21.443 wxLongLong Class Reference

`#include <wx/longlong.h>`

21.443.1 Detailed Description

This class represents a signed 64 bit long number.

It is implemented using the native 64 bit type where available (machines with 64 bit longs or compilers which have (an analog of) *long* long type) and uses the emulation code in the other cases which ensures that it is the most efficient solution for working with 64 bit integers independently of the architecture.

wxLongLong defines all usual arithmetic operations such as addition, subtraction, bitwise shifts and logical operations as well as multiplication and division (not yet for the machines without native *long* long). It also has operators for implicit construction from and conversion to the native *long* long type if it exists and *long*.

You would usually use this type in exactly the same manner as any other (built-in) arithmetic type. Note that wxLongLong is a signed type, if you want unsigned values use wxULongLong which has exactly the same API as wxLongLong except when explicitly mentioned otherwise.

If a native (i.e. supported directly by the compiler) 64 bit integer type was found to exist, *wxLongLong_t* macro will be defined to correspond to it. Also, in this case only, two additional macros will be defined:

- wxLongLongFmtSpec() for printing 64 bit integers using the standard `printf()` function (but see also wxLongLong::ToString for a more portable solution);
- wxLL() for defining 64 bit integer compile-time constants.

Library: wxBase

Category: Data Structures

Public Member Functions

- wxLongLong ()

  Default constructor initializes the object to 0.
- wxLongLong (wxLongLong_t ll)

  Constructor from native long long (only for compilers supporting it).
- wxLongLong (long hi, unsigned long lo)

  Constructor from 2 longs: the high and low part are combined into one wxLongLong.
- wxLongLong Assign (double d)

  This allows to convert a double value to wxLongLong type.
- long GetHi () const

  Returns the high 32 bits of 64 bit integer.
- unsigned long GetLo () const

  Returns the low 32 bits of 64 bit integer.
- wxLongLong_t GetValue () const

  Convert to native long long (only for compilers supporting it).
- double ToDouble () const

Returns the value as `double`.

- long ToLong () const

  Truncate wxLongLong to long.
- wxString ToString () const

  Returns the string representation of a wxLongLong.
- wxLongLong operator+ (const wxLongLong &ll) const

  Adds 2 wxLongLongs together and returns the result.
- wxLongLong & operator+ (const wxLongLong &ll)

  Add another wxLongLong to this one.
- wxLongLong operator- (const wxLongLong &ll) const

  Subtracts 2 wxLongLongs and returns the result.
- wxLongLong & operator- (const wxLongLong &ll)

  Subtracts another wxLongLong from this one.
- wxLongLong operator- () const

  Returns the value of this wxLongLong with opposite sign.
- wxLongLong & operator= (const wxULongLong &ll)

  Assignment operator from unsigned long long.
- wxLongLong & operator= (wxLongLong_t ll)

  Assignment operator from native long long (only for compilers supporting it).
- wxLongLong & operator= (wxULongLong_t ll)

  Assignment operator from native unsigned long long (only for compilers supporting it).
- wxLongLong & operator= (long l)

  Assignment operator from long.
- wxLongLong & operator= (unsigned long l)

  Assignment operator from unsigned long.

- wxLongLong Abs () const

  Returns an absolute value of wxLongLong - either making a copy (const version) or modifying it in place (the second one).
- wxLongLong & Abs ()

  Returns an absolute value of wxLongLong - either making a copy (const version) or modifying it in place (the second one).

- wxLongLong operator++ ()

  Pre/post increment operator.
- wxLongLong operator++ (int)

  Pre/post increment operator.

- wxLongLong operator-- ()

  Pre/post decrement operator.
- wxLongLong operator-- (int)

  Pre/post decrement operator.

21.443.2 Constructor & Destructor Documentation

wxLongLong::wxLongLong ( )

Default constructor initializes the object to 0.

wxLongLong::wxLongLong (wxLongLong_t *ll*)

Constructor from native long long (only for compilers supporting it).

wxLongLong::wxLongLong (long *hi,* unsigned long *lo*)

Constructor from 2 longs: the high and low part are combined into one wxLongLong.

21.443.3 Member Function Documentation

wxLongLong wxLongLong::Abs ( ) const

Returns an absolute value of wxLongLong - either making a copy (const version) or modifying it in place (the second one).

Not in wxULongLong.

wxLongLong& wxLongLong::Abs ( )

Returns an absolute value of wxLongLong - either making a copy (const version) or modifying it in place (the second one).

Not in wxULongLong.

wxLongLong wxLongLong::Assign (double *d*)

This allows to convert a double value to wxLongLong type.

Such conversion is not always possible in which case the result will be silently truncated in a platform-dependent way. Not in wxULongLong.

long wxLongLong::GetHi ( ) const

Returns the high 32 bits of 64 bit integer.

unsigned long wxLongLong::GetLo ( ) const

Returns the low 32 bits of 64 bit integer.

wxLongLong_t wxLongLong::GetValue ( ) const

Convert to native long long (only for compilers supporting it).

wxLongLong wxLongLong::operator+ (const wxLongLong & *ll*) const

Adds 2 wxLongLongs together and returns the result.

wxLongLong& wxLongLong::operator+ (const wxLongLong & *ll*)

Add another wxLongLong to this one.

wxLongLong wxLongLong::operator++ ( )

Pre/post increment operator.

wxLongLong wxLongLong::operator++ (int)

Pre/post increment operator.

wxLongLong wxLongLong::operator- (const wxLongLong & //) const

Subtracts 2 wxLongLongs and returns the result.

wxLongLong& wxLongLong::operator- (const wxLongLong & //)

Subtracts another wxLongLong from this one.

wxLongLong wxLongLong::operator- ( ) const

Returns the value of this wxLongLong with opposite sign.

Not in wxULongLong.

wxLongLong wxLongLong::operator-- ( )

Pre/post decrement operator.

wxLongLong wxLongLong::operator-- (int)

Pre/post decrement operator.

wxLongLong& wxLongLong::operator= (const wxULongLong & //)

Assignment operator from unsigned long long.

The sign bit will be copied too.

Since

2.7.0

wxLongLong& wxLongLong::operator= (wxLongLong_t //)

Assignment operator from native long long (only for compilers supporting it).

wxLongLong& wxLongLong::operator= (wxULongLong_t //)

Assignment operator from native unsigned long long (only for compilers supporting it).

Since

2.7.0

wxLongLong& wxLongLong::operator= (long l)

Assignment operator from long.

Since

2.7.0

wxLongLong& wxLongLong::operator= (unsigned long l)

Assignment operator from unsigned long.

Since

2.7.0

double wxLongLong::ToDouble ( ) const

Returns the value as `double`.

long wxLongLong::ToLong ( ) const

Truncate wxLongLong to long.

If the conversion loses data (i.e. the wxLongLong value is outside the range of built-in long type), an assert will be triggered in debug mode.

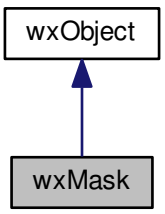
wxString wxLongLong::ToString ( ) const

Returns the string representation of a wxLongLong.

21.444 wxMask Class Reference

`#include <wx/bitmap.h>`

Inheritance diagram for wxMask:



21.444.1 Detailed Description

This class encapsulates a monochrome mask bitmap, where the masked area is black and the unmasked area is white.

When associated with a bitmap and drawn in a device context, the unmasked area of the bitmap will be drawn, and the masked area will not be drawn.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxBitmap, wxDC::Blit, wxMemoryDC

Public Member Functions

- wxMask ()

  Default constructor.
- wxMask (const wxBitmap &bitmap, int index)

  Constructs a mask from a bitmap and a palette index that indicates the background.
- wxMask (const wxBitmap &bitmap)

  Constructs a mask from a monochrome bitmap.
- wxMask (const wxBitmap &bitmap, const wxColour &colour)

  Constructs a mask from a bitmap and a colour that indicates the background.
- virtual ~wxMask ()

  Destroys the wxMask object and the underlying bitmap data.
- bool Create (const wxBitmap &bitmap, int index)

  Constructs a mask from a bitmap and a palette index that indicates the background.
- bool Create (const wxBitmap &bitmap)

  Constructs a mask from a monochrome bitmap.
- bool Create (const wxBitmap &bitmap, const wxColour &colour)

  Constructs a mask from a bitmap and a colour that indicates the background.
- wxBitmap GetBitmap () const

  Returns the mask as a monochrome bitmap.

Additional Inherited Members

21.444.2 Constructor & Destructor Documentation

wxMask::wxMask ( )

Default constructor.

wxMask::wxMask (const wxBitmap & *bitmap,* int *index*)

Constructs a mask from a bitmap and a palette index that indicates the background.

Not implemented for GTK.

Parameters

bitmap	A valid bitmap.
index	Index into a palette, specifying the transparency colour.

wxMask::wxMask (const wxBitmap & *bitmap*)

Constructs a mask from a monochrome bitmap.

wxMask::wxMask (const wxBitmap & *bitmap,* const wxColour & *colour*)

Constructs a mask from a bitmap and a colour that indicates the background.

virtual wxMask::~wxMask ( ) [virtual]

Destroys the wxMask object and the underlying bitmap data.

21.444.3 Member Function Documentation

bool wxMask::Create (const wxBitmap & *bitmap,* int *index*)

Constructs a mask from a bitmap and a palette index that indicates the background.

Not implemented for GTK.

Parameters

bitmap	A valid bitmap.
index	Index into a palette, specifying the transparency colour.

bool wxMask::Create (const wxBitmap & *bitmap*)

Constructs a mask from a monochrome bitmap.

bool wxMask::Create (const wxBitmap & *bitmap,* const wxColour & *colour*)

Constructs a mask from a bitmap and a colour that indicates the background.

wxBitmap wxMask::GetBitmap ( ) const

Returns the mask as a monochrome bitmap.

Currently this method is implemented in wxMSW, wxGTK and wxOSX.

Since

2.9.5

21.445 wxMatrix2D Class Reference

`#include <wx/affinematrix2dbase.h>`

21.445.1 Detailed Description

A simple container for 2x2 matrix.

This simple structure is used with wxAffineMatrix2D.

Library: wxCore

Category: Miscellaneous

Since

2.9.2

Public Member Functions

- wxMatrix2D (wxDouble v11=1, wxDouble v12=0, wxDouble v21=0, wxDouble v22=1)

  Default constructor.

Public Attributes

- wxDouble m_11

  The matrix elements in the usual mathematical notation.
- wxDouble m_12
- wxDouble m_21
- wxDouble m_22

21.445.2 Constructor & Destructor Documentation

wxMatrix2D::wxMatrix2D (wxDouble *v11* = `1`, wxDouble *v12* = `0`, wxDouble *v21* = `0`, wxDouble *v22* = `1`)

Default constructor.

Initializes the matrix elements to the identity.

21.445.3 Member Data Documentation

wxDouble wxMatrix2D::m_11

The matrix elements in the usual mathematical notation.

wxDouble wxMatrix2D::m_12

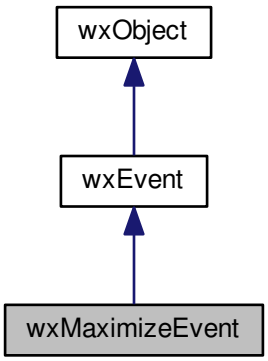
wxDouble wxMatrix2D::m_21

wxDouble wxMatrix2D::m_22

21.446 wxMaximizeEvent Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxMaximizeEvent:



21.446.1 Detailed Description

An event being sent when a top level window is maximized.

Notice that it is not sent when the window is restored to its original size after it had been maximized, only a normal wxSizeEvent is generated in this case.

Currently this event is only generated in wxMSW, wxGTK and wxOSX/Cocoa ports so portable programs should only rely on receiving `wxEVT_SIZE` and not necessarily this event when the window is maximized.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMaximizeEvent& event)

Event macros:

- EVT_MAXIMIZE(func): Process a `wxEVT_MAXIMIZE` event.

Library: wxCore

Category: Events

See also

Events and Event Handling, wxTopLevelWindow::Maximize, wxTopLevelWindow::IsMaximized

Public Member Functions

- wxMaximizeEvent (int id=0)

  Constructor.

Additional Inherited Members

21.446.2 Constructor & Destructor Documentation

wxMaximizeEvent::wxMaximizeEvent (int *id* = `0`)

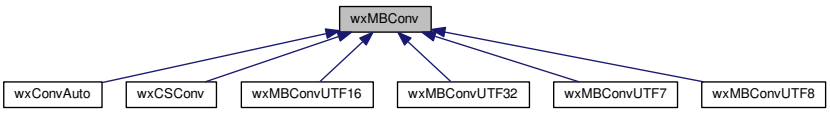
Constructor.

Only used by wxWidgets internally.

21.447 wxMBConv Class Reference

`#include <wx/strconv.h>`

Inheritance diagram for wxMBConv:



21.447.1 Detailed Description

This class is the base class of a hierarchy of classes capable of converting text strings between multibyte (SBCS or DBCS) encodings and Unicode.

This is an abstract base class which defines the operations implemented by all different conversion classes. The derived classes don't add any new operations of their own (except, possibly, some non-default constructors) and so you should simply use this class ToWChar() and FromWChar() (or cMB2WC() and cWC2MB()) methods with the objects of the derived class.

In the documentation for this and related classes please notice that length of the string refers to the number of characters in the string not counting the terminating `NUL`, if any. While the size of the string is the total number of bytes in the string, including any trailing `NUL`. Thus, length of wide character string `L"foo"` is 3 while its size can be either 8 or 16 depending on whether `wchar_t` is 2 bytes (as under Windows) or 4 (Unix).

Library: wxBase

Category: Text Conversion

See also

wxCSConv, wxEncodingConverter, wxMBConv Overview

Public Member Functions

- wxMBConv ()

  Trivial default constructor.
- virtual wxMBConv * Clone () const =0

  This pure virtual function is overridden in each of the derived classes to return a new copy of the object it is called on.
- virtual size_t GetMBNulLen () const

This function returns 1 for most of the multibyte encodings in which the string is terminated by a single NUL, 2 for UTF-16 and 4 for UTF-32 for which the string is terminated with 2 and 4 NUL characters respectively.

- virtual size_t ToWChar (wchar_t *dst, size_t dstLen, const char *src, size_t srcLen=wxNO_LEN) const

  Convert multibyte string to a wide character one.
- virtual size_t FromWChar (char *dst, size_t dstLen, const wchar_t *src, size_t srcLen=wxNO_LEN) const

  Converts wide character string to multibyte.
- const wxWCharBuffer cMB2WC (const char *in, size_t inLen, size_t *outLen) const

  Converts from multibyte encoding to Unicode by calling ToWChar() and allocating a temporary wxWCharBuffer to hold the result.
- const wxWCharBuffer cMB2WC (const wxCharBuffer &buf) const

  Converts a char buffer to wide char one.
- const wxCharBuffer cWC2MB (const wchar_t *in, size_t inLen, size_t *outLen) const

  Converts from Unicode to multibyte encoding by calling FromWChar() and allocating a temporary wxCharBuffer to hold the result.
- const wxCharBuffer cWC2MB (const wxWCharBuffer &buf) const

  Converts a wide char buffer to char one.
- virtual size_t MB2WC (wchar_t *out, const char *in, size_t outLen) const
- virtual size_t WC2MB (char *buf, const wchar_t *psz, size_t n) const

- const char * cMB2WX (const char *psz) const

  Converts from multibyte encoding to the current wxChar type (which depends on whether wxUSE_UNICODE is set to 1).
- const wxWCharBuffer cMB2WX (const char *psz) const

  Converts from multibyte encoding to the current wxChar type (which depends on whether wxUSE_UNICODE is set to 1).

- const wchar_t * cWC2WX (const wchar_t *psz) const

  Converts from Unicode to the current wxChar type.
- const wxCharBuffer cWC2WX (const wchar_t *psz) const

  Converts from Unicode to the current wxChar type.

- const char * cWX2MB (const wxChar *psz) const

  Converts from the current wxChar type to multibyte encoding.
- const wxCharBuffer cWX2MB (const wxChar *psz) const

  Converts from the current wxChar type to multibyte encoding.

- const wchar_t * cWX2WC (const wxChar *psz) const

  Converts from the current wxChar type to Unicode.
- const wxWCharBuffer cWX2WC (const wxChar *psz) const

  Converts from the current wxChar type to Unicode.

Static Public Member Functions

- static size_t GetMaxMBNulLen ()

  Returns the maximal value which can be returned by GetMBNulLen() for any conversion object.

21.447.2 Constructor & Destructor Documentation

wxMBConv::wxMBConv ( )

Trivial default constructor.

21.447.3 Member Function Documentation

virtual wxMBConv* wxMBConv::Clone ( ) const `[pure virtual]`

This pure virtual function is overridden in each of the derived classes to return a new copy of the object it is called on.

It is used for copying the conversion objects while preserving their dynamic type.

const wxWCharBuffer wxMBConv::cMB2WC (const char * *in*, size_t *inLen*, size_t * *outLen*) const

Converts from multibyte encoding to Unicode by calling ToWChar() and allocating a temporary wxWCharBuffer to hold the result.

This function is a convenient wrapper around ToWChar() as it takes care of allocating the buffer of the necessary size itself. Its parameters have the same meaning as for ToWChar(), in particular *inLen* can be specified explicitly in which case exactly that many characters are converted and *outLen* receives (if non-NULL) exactly the corresponding number of wide characters, whether the last one of them is `NUL` or not. However if `inLen` is `wxNO_LEN`, then `outLen` doesn't count the trailing `NUL` even if it is always present in this case.

Finally notice that if the conversion fails, the returned buffer is invalid and *outLen* is set to 0 (and not `wxCONV_FAILED` for compatibility concerns).

const wxWCharBuffer wxMBConv::cMB2WC (const wxCharBuffer & *buf*) const

Converts a char buffer to wide char one.

This is the most convenient and safest conversion function as you don't have to deal with the buffer lengths directly. Use it if the input buffer is known not to be empty or if you are sure that the conversion is going to succeed – otherwise, use the overload above to be able to distinguish between empty input and conversion failure.

Returns

The buffer containing the converted text, empty if the input was empty or if the conversion failed.

Since

2.9.1

const char* wxMBConv::cMB2WX (const char * *psz*) const

Converts from multibyte encoding to the current wxChar type (which depends on whether wxUSE_UNICODE is set to 1).

If wxChar is char, it returns the parameter unaltered. If wxChar is wchar_t, it returns the result in a wxWCharBuffer. The macro wxMB2WXbuf is defined as the correct return type (without const).

const wxWCharBuffer wxMBConv::cMB2WX (const char * *psz*) const

Converts from multibyte encoding to the current wxChar type (which depends on whether wxUSE_UNICODE is set to 1).

If wxChar is char, it returns the parameter unaltered. If wxChar is wchar_t, it returns the result in a wxWCharBuffer. The macro wxMB2WXbuf is defined as the correct return type (without const).

const wxCharBuffer wxMBConv::cWC2MB (const wchar_t * *in*, size_t *inLen*, size_t * *outLen*) const

Converts from Unicode to multibyte encoding by calling FromWChar() and allocating a temporary wxCharBuffer to hold the result.

This function is a convenient wrapper around FromWChar() as it takes care of allocating the buffer of necessary size itself.

Its parameters have the same meaning as the corresponding parameters of FromWChar(), please see the description of cMB2WC() for more details.

const wxCharBuffer wxMBConv::cWC2MB (const wxWCharBuffer & *buf*) const

Converts a wide char buffer to char one.

This is the most convenient and safest conversion function as you don't have to deal with the buffer lengths directly. Use it if the input buffer is known not to be empty or if you are sure that the conversion is going to succeed – otherwise, use the overload above to be able to distinguish between empty input and conversion failure.

Returns

The buffer containing the converted text, empty if the input was empty or if the conversion failed.

Since

2.9.1

const wchar_t* wxMBConv::cWC2WX (const wchar_t * *psz*) const

Converts from Unicode to the current wxChar type.

If wxChar is wchar_t, it returns the parameter unaltered. If wxChar is char, it returns the result in a wxCharBuffer. The macro wxWC2WXbuf is defined as the correct return type (without const).

const wxCharBuffer wxMBConv::cWC2WX (const wchar_t * *psz*) const

Converts from Unicode to the current wxChar type.

If wxChar is wchar_t, it returns the parameter unaltered. If wxChar is char, it returns the result in a wxCharBuffer. The macro wxWC2WXbuf is defined as the correct return type (without const).

const char* wxMBConv::cWX2MB (const wxChar * *psz*) const

Converts from the current wxChar type to multibyte encoding.

If wxChar is char, it returns the parameter unaltered. If wxChar is wchar_t, it returns the result in a wxCharBuffer. The macro wxWX2MBbuf is defined as the correct return type (without const).

const wxCharBuffer wxMBConv::cWX2MB (const wxChar * *psz*) const

Converts from the current wxChar type to multibyte encoding.

If wxChar is char, it returns the parameter unaltered. If wxChar is wchar_t, it returns the result in a wxCharBuffer. The macro wxWX2MBbuf is defined as the correct return type (without const).

const wchar_t* wxMBConv::cWX2WC (const wxChar * *psz*) const

Converts from the current wxChar type to Unicode.

If wxChar is wchar_t, it returns the parameter unaltered. If wxChar is char, it returns the result in a wxWCharBuffer. The macro wxWX2WCbuf is defined as the correct return type (without const).

const wxWCharBuffer wxMBConv::cWX2WC (const wxChar * *psz*) const

Converts from the current wxChar type to Unicode.

If wxChar is wchar_t, it returns the parameter unaltered. If wxChar is char, it returns the result in a wxWCharBuffer. The macro wxWX2WCbuf is defined as the correct return type (without const).

virtual size_t wxMBConv::FromWChar (char * *dst*, size_t *dstLen*, const wchar_t * *src*, size_t *srcLen* = `wxNO_LEN`) const `[virtual]`

Converts wide character string to multibyte.

This function has the same semantics as ToWChar() except that it converts a wide string to multibyte one. As with ToWChar(), it may be more convenient to use cWC2MB() when working with `NUL` terminated strings.

Parameters

dst	Pointer to output buffer of the size of at least *dstLen* or NULL.
dstLen	Maximal number of characters to be written to the output buffer if *dst* is non-NULL, unused otherwise.
src	Point to the source string, must not be NULL.
srcLen	The number of characters of the source string to convert or `wxNO_LEN` (default parameter) to convert everything up to and including the terminating `NUL` character.

Returns

The number of character written (or which would have been written if it were non-NULL) to *dst* or `wxCONV_FAILED` on error.

static size_t wxMBConv::GetMaxMBNulLen ( ) `[static]`

Returns the maximal value which can be returned by GetMBNulLen() for any conversion object.

Currently this value is 4.

This method can be used to allocate the buffer with enough space for the trailing `NUL` characters for any encoding.

virtual size_t wxMBConv::GetMBNulLen ( ) const `[virtual]`

This function returns 1 for most of the multibyte encodings in which the string is terminated by a single `NUL`, 2 for UTF-16 and 4 for UTF-32 for which the string is terminated with 2 and 4 `NUL` characters respectively.

The other cases are not currently supported and `wxCONV_FAILED` (defined as -1) is returned for them.

virtual size_t wxMBConv::MB2WC (wchar_t * *out*, const char * *in*, size_t *outLen*) const `[virtual]`

Deprecated This function is deprecated, please use ToWChar() instead.

Converts from a string *in* multibyte encoding to Unicode putting up to *outLen* characters into the buffer *out*.

If *out* is NULL, only the length of the string which would result from the conversion is calculated and returned. Note that this is the length and not size, i.e. the returned value does not include the trailing NUL. But when the function is called with a non-NULL *out* buffer, the *outLen* parameter should be one more to allow to properly NUL-terminate the string.

So to properly use this function you need to write:

```
size_t lenConv = conv.MB2WC(NULL, in, 0);
if ( lenConv == wxCONV_FAILED )
    ... handle error ...
// allocate 1 more character for the trailing NUL and also pass
// the size of the buffer to the function now
wchar_t *out = new wchar_t[lenConv + 1];
if ( conv.MB2WC(out, in, lenConv + 1) == wxCONV_FAILED )
    ... handle error ...
```

For this and other reasons, ToWChar() is strongly recommended as a replacement.

Parameters

out	The output buffer, may be NULL if the caller is only interested in the length of the resulting string
in	The NUL-terminated input string, cannot be NULL
outLen	The length of the output buffer but including NUL, ignored if out is NULL

Returns

The length of the converted string excluding the trailing NUL.

virtual size_t wxMBConv::ToWChar (wchar_t * *dst*, size_t *dstLen*, const char * *src*, size_t *srcLen* = wxNO_LEN) const [virtual]

Convert multibyte string to a wide character one.

This is the most general function for converting a multibyte string to a wide string, cMB2WC() may be often more convenient, however this function is the most efficient one as it allows to avoid any unnecessary copying.

The main case is when *dst* is not NULL and *srcLen* is not wxNO_LEN (which is defined as (size_t)-1): then the function converts exactly *srcLen* bytes starting at *src* into wide string which it output to *dst*. If the length of the resulting wide string is greater than *dstLen*, an error is returned. Note that if *srcLen* bytes don't include NUL characters, the resulting wide string is not NUL-terminated neither.

If *srcLen* is wxNO_LEN, the function supposes that the string is properly (i.e. as necessary for the encoding handled by this conversion) NUL-terminated and converts the entire string, including any trailing NUL bytes. In this case the wide string is also NUL-terminated.

Finally, if *dst* is NULL, the function returns the length of the needed buffer.

Example of use of this function:

```
size_t dstLen = conv.ToWChar(NULL, 0, src);
if ( dstLen == wxCONV_FAILED )
    ... handle error ...
wchar_t *dst = new wchar_t[dstLen];
if ( conv.ToWChar(dst, dstLen, src) == wxCONV_FAILED )
    ... handle error ...
```

Notice that when passing the explicit source length the output will *not* be NUL terminated if you pass strlen(str) as parameter. Either leave *srcLen* as default wxNO_LEN or add one to strlen result if you want the output to be NUL terminated.

Parameters

dst	Pointer to output buffer of the size of at least *dstLen* or NULL.
dstLen	Maximal number of characters to be written to the output buffer if *dst* is non-NULL, unused otherwise.
src	Point to the source string, must not be NULL.
srcLen	The number of characters of the source string to convert or `wxNO_LEN` (default parameter) to convert everything up to and including the terminating `NUL` character(s).

Returns

The number of character written (or which would have been written if it were non-NULL) to *dst* or `wxCONV_FAILED` on error.

virtual size_t wxMBConv::WC2MB (char * *buf*, const wchar_t * *psz*, size_t *n*) const `[virtual]`

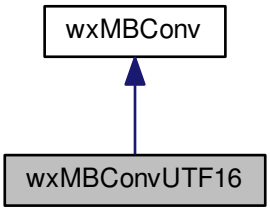
Deprecated This function is deprecated, please use FromWChar() instead.

Converts from Unicode to multibyte encoding. The semantics of this function (including the return value meaning) is the same as for wxMBConv::MB2WC. Notice that when the function is called with a non-NULL buffer, the *n* parameter should be the size of the buffer and so it should take into account the trailing `NUL`, which might take two or four bytes for some encodings (UTF-16 and UTF-32) and not one, i.e. GetMBNulLen().

21.448 wxMBConvUTF16 Class Reference

`#include <wx/strconv.h>`

Inheritance diagram for wxMBConvUTF16:



21.448.1 Detailed Description

This class is used to convert between multibyte encodings and UTF-16 Unicode encoding (also known as UCS-2).

Unlike UTF-8 encoding, UTF-16 uses words and not bytes and hence depends on the byte ordering: big or little endian. Hence this class is provided in two versions: wxMBConvUTF16LE and wxMBConvUTF16BE and wxMBConvUTF16 itself is just a typedef for one of them (native for the given platform, e.g. LE under Windows and BE under Mac).

Library: wxBase

Category: Text Conversion

See also

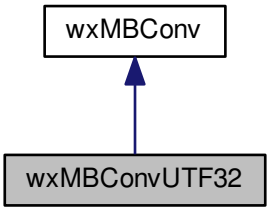
wxMBConvUTF8, wxMBConvUTF32, wxMBConv Overview

Additional Inherited Members

21.449 wxMBConvUTF32 Class Reference

`#include <wx/strconv.h>`

Inheritance diagram for wxMBConvUTF32:



21.449.1 Detailed Description

This class is used to convert between multibyte encodings and UTF-32 Unicode encoding (also known as UCS-4).

Unlike UTF-8 encoding, UTF-32 uses (double) words and not bytes and hence depends on the byte ordering: big or little endian. Hence this class is provided in two versions: wxMBConvUTF32LE and wxMBConvUTF32BE and wxMBConvUTF32 itself is just a typedef for one of them (native for the given platform, e.g. LE under Windows and BE under Mac).

Library: wxBase

Category: Text Conversion

See also

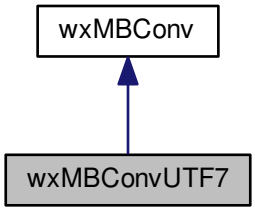
wxMBConvUTF8, wxMBConvUTF16, wxMBConv Overview

Additional Inherited Members

21.450 wxMBConvUTF7 Class Reference

`#include <wx/strconv.h>`

Inheritance diagram for wxMBConvUTF7:



21.450.1 Detailed Description

This class converts between the UTF-7 encoding and Unicode.

It has one predefined instance, **wxConvUTF7**.

Notice that, unlike all the other conversion objects, this converter is stateful, i.e. it remembers its state from the last call to its ToWChar() or FromWChar() and assumes it is called on the continuation of the same string when the same method is called again. This assumption is only made if an explicit length is specified as parameter to these functions as if an entire `NUL` terminated string is processed the state doesn't need to be remembered.

This also means that, unlike the other predefined conversion objects, **wxConvUTF7** is *not* thread-safe.

Library: wxBase

Category: Text Conversion

See also

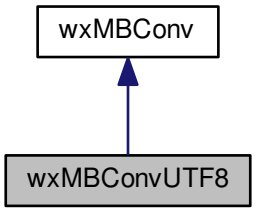
wxMBConvUTF8, wxMBConv Overview

Additional Inherited Members

21.451 wxMBConvUTF8 Class Reference

```
#include <wx/strconv.h>
```

Inheritance diagram for wxMBConvUTF8:



21.451.1 Detailed Description

This class converts between the UTF-8 encoding and Unicode.

It has one predefined instance, **wxConvUTF8**.

Library: wxBase

Category: Text Conversion

See also

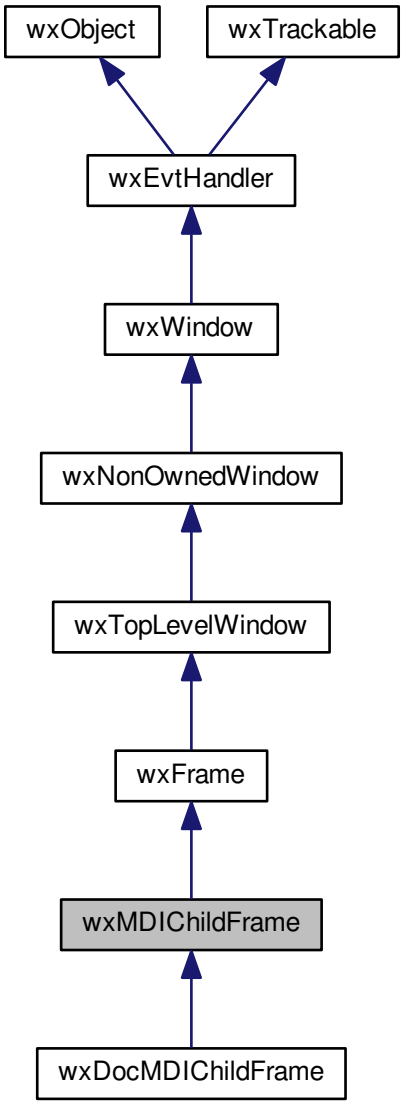
wxMBConvUTF7, wxMBConv Overview

Additional Inherited Members

21.452 wxMDIChildFrame Class Reference

`#include <wx/mdi.h>`

Inheritance diagram for wxMDIChildFrame:



21.452.1 Detailed Description

An MDI child frame is a frame that can only exist inside a wxMDIClientWindow, which is itself a child of wxMDI↩ ParentFrame.

Styles

This class supports the following styles:

All of the standard wxFrame styles can be used but most of them are ignored by TDI-based MDI implementations.

Remarks

Although internally an MDI child frame is a child of the MDI client window, in wxWidgets you create it as a child of wxMDIParentFrame. In fact, you can usually forget that the client window exists. MDI child frames are clipped to the area of the MDI client window, and may be iconized on the client window. You can associate a menubar with a child frame as usual, although an MDI child doesn't display its menubar under its own title bar. The MDI parent frame's menubar will be changed to reflect the currently active child frame. If there are currently no children, the parent frame's own menubar will be displayed.

Library: wxCore

Category: Managed Windows

See also

wxMDIClientWindow, wxMDIParentFrame, wxFrame

Public Member Functions

- wxMDIChildFrame ()

  Default constructor.
- wxMDIChildFrame (wxMDIParentFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor, creating the window.
- virtual ~wxMDIChildFrame ()

  Destructor.
- virtual void Activate ()

  Activates this MDI child frame.
- bool Create (wxMDIParentFrame *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Used in two-step frame construction.
- wxMDIParentFrame * GetMDIParent () const

  Returns the MDI parent frame containing this child.
- virtual bool IsAlwaysMaximized () const

  Returns true for MDI children in TDI implementations.
- virtual void Maximize (bool maximize=true)

  Maximizes this MDI child frame.
- virtual void Restore ()

  Restores this MDI child frame (unmaximizes).

Additional Inherited Members

21.452.2 Constructor & Destructor Documentation

wxMDIChildFrame::wxMDIChildFrame ( )

Default constructor.

wxMDIChildFrame::wxMDIChildFrame (wxMDIParentFrame * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor, creating the window.

Parameters

parent	The window parent. This should not be NULL.
id	The window identifier. It may take a value of -1 to indicate a default value.
title	The caption to be displayed on the frame's title bar.
pos	The window position. The value wxDefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.
size	The window size. The value wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.
style	The window style. See wxMDIChildFrame.
name	The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

See also

Create()

virtual wxMDIChildFrame::~wxMDIChildFrame ( ) [virtual]

Destructor.

Destroys all child windows and menu bar if present.

21.452.3 Member Function Documentation

virtual void wxMDIChildFrame::Activate ( ) [virtual]

Activates this MDI child frame.

See also

Maximize(), Restore()

bool wxMDIChildFrame::Create (wxMDIParentFrame * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = wxFrameNameStr)

Used in two-step frame construction.

See wxMDIChildFrame() for further details.

wxMDIParentFrame* wxMDIChildFrame::GetMDIParent ( ) const

Returns the MDI parent frame containing this child.

Notice that this may return a different object than GetParent() as the child frames may be created as children of the client window internally.

virtual bool wxMDIChildFrame::IsAlwaysMaximized ( ) const [virtual]

Returns true for MDI children in TDI implementations.

TDI-based implementations represent MDI children as pages in a wxNotebook and so they are always maximized and can't be restored or iconized.

See also

wxMDIParentFrame::IsTDI().

Reimplemented from wxTopLevelWindow.

virtual void wxMDIChildFrame::Maximize (bool *maximize* = `true`) `[virtual]`

Maximizes this MDI child frame.

This function doesn't do anything if IsAlwaysMaximized() returns true.

See also

Activate(), Restore()

Reimplemented from wxTopLevelWindow.

virtual void wxMDIChildFrame::Restore ( ) `[virtual]`

Restores this MDI child frame (unmaximizes).

This function doesn't do anything if IsAlwaysMaximized() returns true.

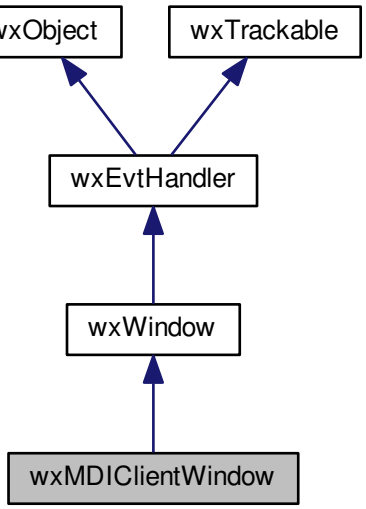
See also

Activate(), Maximize()

21.453 wxMDIClientWindow Class Reference

`#include <wx/mdi.h>`

Inheritance diagram for wxMDIClientWindow:



21.453.1 Detailed Description

An MDI client window is a child of wxMDIParentFrame, and manages zero or more wxMDIChildFrame objects.

Remarks

The client window is the area where MDI child windows exist. It doesn't have to cover the whole parent frame; other windows such as toolbars and a help window might coexist with it. There can be scrollbars on a client window, which are controlled by the parent window style.

The wxMDIClientWindow class is usually adequate without further derivation, and it is created automatically when the MDI parent frame is created. If the application needs to derive a new class, the function wxMDIParentFrame::OnCreateClient() must be overridden in order to give an opportunity to use a different class of client window.

Under wxMSW, the client window will automatically have a sunken border style when the active child is not maximized, and no border style when a child is maximized.

Library: wxCore

Category: Managed Windows

See also

wxMDIChildFrame, wxMDIParentFrame, wxFrame

Public Member Functions

- wxMDIClientWindow ()

  Default constructor.
- virtual bool CreateClient (wxMDIParentFrame *parent, long style=0)

  Called by wxMDIParentFrame immediately after creating the client window.

Additional Inherited Members

21.453.2 Constructor & Destructor Documentation

wxMDIClientWindow::wxMDIClientWindow ( )

Default constructor.

Objects of this class are only created by wxMDIParentFrame which uses the default constructor and calls CreateClient() immediately afterwards.

21.453.3 Member Function Documentation

virtual bool wxMDIClientWindow::CreateClient (wxMDIParentFrame * *parent*, long *style* = 0) `[virtual]`

Called by wxMDIParentFrame immediately after creating the client window.

This function may be overridden in the derived class but the base class version must usually be called first to really create the window.

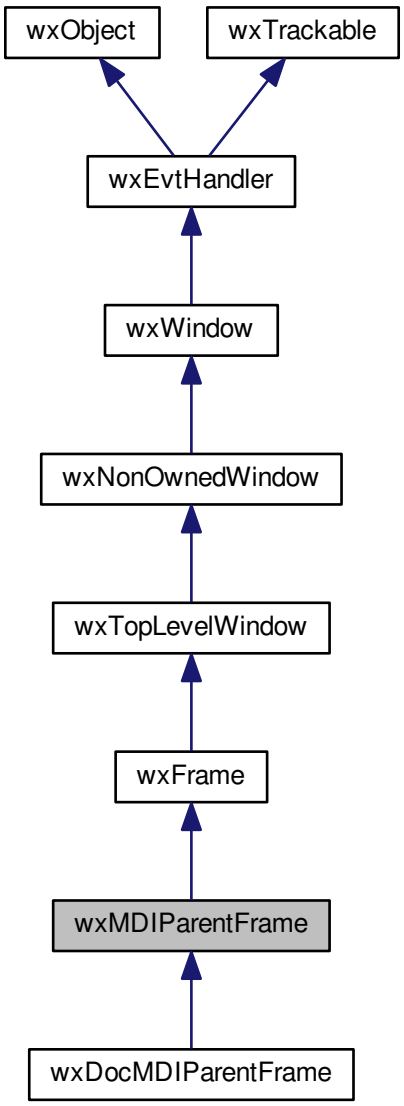
Parameters

parent	The window parent.
style	The window style. Only wxHSCROLL and wxVSCROLL bits are meaningful here.

21.454 wxMDIParentFrame Class Reference

`#include <wx/mdi.h>`

Inheritance diagram for wxMDIParentFrame:



21.454.1 Detailed Description

An MDI (Multiple Document Interface) parent frame is a window which can contain MDI child frames in its client area which emulates the full desktop.

MDI is a user-interface model in which all the window reside inside the single parent window as opposed to being separate from each other. It remains popular despite dire warnings from Microsoft itself (which popularized this model in the first model) that MDI is obsolete.

An MDI parent frame always has a wxMDIClientWindow associated with it, which is the parent for MDI child frames. In the simplest case, the client window takes up the entire parent frame area but it is also possible to resize it to be smaller in order to have other windows in the frame, a typical example is using a sidebar along one of the window edges.

The appearance of MDI applications differs between different ports. The classic MDI model, with child windows which can be independently moved, resized etc, is only available under MSW, which provides native support for it. In Mac ports, multiple top level windows are used for the MDI children too and the MDI parent frame itself is invisible, to accommodate the native look and feel requirements. In all the other ports, a tab-based MDI implementation (sometimes called TDI) is used and so at most one MDI child is visible at any moment (child frames are always maximized).

Remarks

Although it is possible to have multiple MDI parent frames, a typical MDI application has a single MDI parent frame window inside which multiple MDI child frames, i.e. objects of class wxMDIChildFrame, can be created.

Styles

This class supports the following styles:

There are no special styles for this class, all wxFrame styles apply to it in the usual way. The only exception is that wxHSCROLL and wxVSCROLL styles apply not to the frame itself but to the client window, so that using them enables horizontal and vertical scrollbars for this window and not the frame.

Library: wxCore

Category: Managed Windows

See also

wxMDIChildFrame, wxMDIClientWindow, wxFrame, wxDialog

Public Member Functions

- wxMDIParentFrame ()

  Default constructor.
- wxMDIParentFrame (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE|wxVSCROLL|wxHSCROLL, const wxString &name=wxFrameNameStr)

  Constructor, creating the window.
- virtual ~wxMDIParentFrame ()

  Destructor.
- virtual void ActivateNext ()

  Activates the MDI child following the currently active one.

- virtual void ActivatePrevious ()

  Activates the MDI child preceding the currently active one.
- virtual void ArrangeIcons ()

  Arranges any iconized (minimized) MDI child windows.
- virtual void Cascade ()

  Arranges the MDI child windows in a cascade.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE|wxVSCROLL|wxHSCROLL, const wxString &name=wxFrameNameStr)

  Used in two-step frame construction.
- virtual wxMDIChildFrame * GetActiveChild () const

  Returns a pointer to the active MDI child, if there is one.
- wxMDIClientWindowBase * GetClientWindow () const

  Returns a pointer to the client window.
- wxMenu * GetWindowMenu () const

  Returns the current MDI Window menu.
- virtual wxMDIClientWindow * OnCreateClient ()

  Override this to return a different kind of client window.
- virtual void SetWindowMenu (wxMenu *menu)

  Replace the current MDI Window menu.
- virtual void Tile (wxOrientation orient=wxHORIZONTAL)

  Tiles the MDI child windows either horizontally or vertically depending on whether orient is `wxHORIZONTAL` or `wxVERTICAL`.

Static Public Member Functions

- static bool IsTDI ()

  Returns whether the MDI implementation is tab-based.

Additional Inherited Members

21.454.2 Constructor & Destructor Documentation

wxMDIParentFrame::wxMDIParentFrame ( )

Default constructor.

Use Create() for the objects created using this constructor.

wxMDIParentFrame::wxMDIParentFrame (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE|wxVSCROLL|wxHSCROLL, const wxString & *name* = `wxFrameNameStr`)

Constructor, creating the window.

Notice that if you override virtual OnCreateClient() method you shouldn't be using this constructor but the default constructor and Create() as otherwise your overridden method is never going to be called because of the usual C++ virtual call resolution rules.

Parameters

parent	The window parent. Usually is NULL.
id	The window identifier. It may take a value of `wxID_ANY` to indicate a default value.
title	The caption to be displayed on the frame's title bar.
pos	The window position. The value wxDefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.
size	The window size. The value wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.
style	The window style. Default value includes wxHSCROLL and wxVSCROLL styles.
name	The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

Under wxMSW, the client window will automatically have a sunken border style when the active child is not maximized, and no border style when a child is maximized.

See also

Create(), OnCreateClient()

virtual wxMDIParentFrame::~wxMDIParentFrame ( ) [virtual]

Destructor.

Destroys all child windows and menu bar if present.

21.454.3 Member Function Documentation

virtual void wxMDIParentFrame::ActivateNext ( ) [virtual]

Activates the MDI child following the currently active one.

The MDI children are maintained in an ordered list and this function switches to the next element in this list, wrapping around the end of it if the currently active child is the last one.

See also

ActivatePrevious()

virtual void wxMDIParentFrame::ActivatePrevious ( ) [virtual]

Activates the MDI child preceding the currently active one.

See also

ActivateNext()

virtual void wxMDIParentFrame::ArrangeIcons ( ) [virtual]

Arranges any iconized (minimized) MDI child windows.

This method is only implemented in MSW MDI implementation and does nothing under the other platforms.

See also

Cascade(), Tile()

virtual void wxMDIParentFrame::Cascade ( ) [virtual]

Arranges the MDI child windows in a cascade.

This method is only implemented in MSW MDI implementation and does nothing under the other platforms.

See also

Tile(), ArrangeIcons()

bool wxMDIParentFrame::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE|wxVSCROLL|wxHSCROLL, const wxString & *name* = wxFrameNameStr)

Used in two-step frame construction.

See wxMDIParentFrame() for further details.

virtual wxMDIChildFrame* wxMDIParentFrame::GetActiveChild ( ) const [virtual]

Returns a pointer to the active MDI child, if there is one.

If there are any children at all this function returns a non-NULL pointer.

wxMDIClientWindowBase* wxMDIParentFrame::GetClientWindow ( ) const

Returns a pointer to the client window.

See also

OnCreateClient()

wxMenu* wxMDIParentFrame::GetWindowMenu ( ) const

Returns the current MDI Window menu.

Unless wxFRAME_NO_WINDOW_MENU style was used, a default menu listing all the currently active children and providing the usual operations (tile, cascade, ...) on them is created automatically by the library and this function can be used to retrieve it. Notice that the default menu can be replaced by calling SetWindowMenu().

This function is currently not available under OS X.

Returns

The current Window menu or NULL.

static bool wxMDIParentFrame::IsTDI ( ) [static]

Returns whether the MDI implementation is tab-based.

Currently only the MSW port uses the real MDI. In Mac ports the usual SDI is used, as common under this platforms, and all the other ports use TDI implementation.

TDI-based MDI applications have different appearance and functionality (e.g. child frames can't be minimized and only one of them is visible at any given time) so the application may need to adapt its interface somewhat depending on the return value of this function.

virtual wxMDIClientWindow* wxMDIParentFrame::OnCreateClient ( ) [virtual]

Override this to return a different kind of client window.

If you override this function, you must create your parent frame in two stages, or your function will never be called, due to the way C++ treats virtual functions called from constructors. For example:

```
frame = new MyParentFrame;
frame->Create(parent, myParentFrameId, "My Parent Frame");
```

Remarks

You might wish to derive from wxMDIClientWindow in order to implement different erase behaviour, for example, such as painting a bitmap on the background.

Note that it is probably impossible to have a client window that scrolls as well as painting a bitmap or pattern, since in **OnScroll**, the scrollbar positions always return zero.

See also

GetClientWindow(), wxMDIClientWindow

virtual void wxMDIParentFrame::SetWindowMenu (wxMenu * *menu*) [virtual]

Replace the current MDI Window menu.

Ownership of the menu object passes to the frame when you call this function, i.e. the menu will be deleted by it when it's no longer needed (usually when the frame itself is deleted or when SetWindowMenu() is called again).

To remove the window completely, you can use the wxFRAME_NO_WINDOW_MENU window style but this function also allows to do it by passing NULL pointer as *menu*.

The menu may include the items with the following standard identifiers (but may use arbitrary text and help strings and bitmaps for them):

- wxID_MDI_WINDOW_CASCADE
- wxID_MDI_WINDOW_TILE_HORZ
- wxID_MDI_WINDOW_TILE_VERT
- wxID_MDI_WINDOW_ARRANGE_ICONS
- wxID_MDI_WINDOW_PREV
- wxID_MDI_WINDOW_NEXT All of which are handled by wxMDIParentFrame itself. If any other commands are used in the menu, the derived frame should handle them.

This function is currently not available under OS X.

Parameters

menu	The menu to be used instead of the standard MDI Window menu or NULL.

virtual void wxMDIParentFrame::Tile (wxOrientation *orient* = wxHORIZONTAL) `[virtual]`

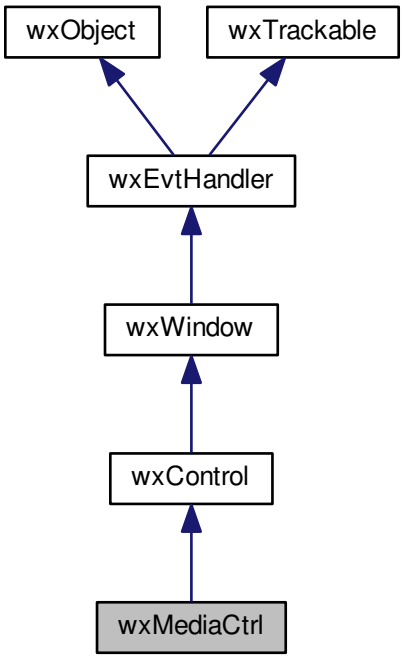
Tiles the MDI child windows either horizontally or vertically depending on whether *orient* is `wxHORIZONTAL` or `wxVERTICAL`.

This method is only implemented in MSW MDI implementation and does nothing under the other platforms.

21.455 wxMediaCtrl Class Reference

`#include <wx/mediactrl.h>`

Inheritance diagram for wxMediaCtrl:



21.455.1 Detailed Description

wxMediaCtrl is a class for displaying types of media, such as videos, audio files, natively through native codecs.

wxMediaCtrl uses native backends to render media, for example on Windows there is a ActiveMovie/DirectShow backend, and on Macintosh there is a QuickTime backend.

21.455.2 Rendering media

Depending upon the backend, wxMediaCtrl can render and display pretty much any kind of media that the native system can - such as an image, mpeg video, or mp3 (without license restrictions - since it relies on native system calls that may not technically have mp3 decoding available, for example, it falls outside the realm of licensing restrictions).

For general operation, all you need to do is call Load() to load the file you want to render, catch the EVT_MEDIA↩_LOADED event, and then call Play() to show the video/audio of the media in that event.

More complex operations are generally more heavily dependent on the capabilities of the backend. For example, QuickTime cannot set the playback rate of certain streaming media - while DirectShow is slightly more flexible in that regard.

21.455.3 Operation

When wxMediaCtrl plays a file, it plays until the stop position is reached (currently the end of the file/stream). Right before it hits the end of the stream, it fires off a EVT_MEDIA_STOP event to its parent window, at which point the event handler can choose to veto the event, preventing the stream from actually stopping.

Example:

```
//connect to the media event
this->Connect(wxMY_ID, wxEVT_MEDIA_STOP, (wxObjectEventFunction)
(wxEventFunction)(wxMediaEventFunction) &MyFrame::OnMediaStop);

//...
void MyFrame::OnMediaStop(const wxMediaEvent& evt)
{
    if(bUserWantsToSeek)
    {
        m_mediactrl->SetPosition(
            m_mediactrl->GetDuration() << 1
                                );
        evt.Veto();
    }
}
```

When wxMediaCtrl stops, either by the EVT_MEDIA_STOP not being vetoed, or by manually calling Stop(), where it actually stops is not at the beginning, rather, but at the beginning of the stream. That is, when it stops and play is called, playback is guaranteed to start at the beginning of the media. This is because some streams are not seekable, and when stop is called on them they return to the beginning, thus wxMediaCtrl tries to keep consistent for all types of media.

Note that when changing the state of the media through Play() and other methods, the media may not actually be in the wxMEDIASTATE_PLAYING, for example. If you are relying on the media being in certain state catch the event relevant to the state. See wxMediaEvent for the kinds of events that you can catch.

21.455.4 Video size

By default, wxMediaCtrl will scale the size of the video to the requested amount passed to either its constructor or Create(). After calling wxMediaCtrl::Load or performing an equivalent operation, you can subsequently obtain the "real" size of the video (if there is any) by calling wxMediaCtrl::GetBestSize(). Note that the actual result on the display will be slightly different when wxMediaCtrl::ShowPlayerControls is activated and the actual video size will be less than specified due to the extra controls provided by the native toolkit. In addition, the backend may modify wxMediaCtrl::GetBestSize() to include the size of the extra controls - so if you want the real size of the video just disable wxMediaCtrl::ShowPlayerControls().

The idea with setting wxMediaCtrl::GetBestSize() to the size of the video is that GetBestSize() is a wxWindow-derived function that is called when sizers on a window recalculate. What this means is that if you use sizers by default the video will show in its original size without any extra assistance needed from the user.

21.455.5 Player controls

Normally, when you use wxMediaCtrl it is just a window for the video to play in. However, some toolkits have their own media player interface. For example, QuickTime generally has a bar below the video with a slider. A special feature available to wxMediaCtrl, you can use the toolkits interface instead of making your own by using the ShowPlayerControls() function. There are several options for the flags parameter, with the two general flags being `wxMEDIACTRLPLAYERCONTROLS_NONE` which turns off the native interface, and `wxMEDIACTRLPLAYER↩CONTROLS_DEFAULT` which lets wxMediaCtrl decide what native controls on the interface. Be sure to review the caveats outlined in Video size before doing so.

21.455.6 Choosing a backend

Generally, you should almost certainly leave this part up to wxMediaCtrl - but if you need a certain backend for a particular reason, such as QuickTime for playing .mov files, all you need to do to choose a specific backend is to pass the name of the backend class to wxMediaCtrl::Create().

The following are valid backend identifiers:

- **wxMEDIABACKEND_DIRECTSHOW:** Use ActiveMovie/DirectShow. Uses the native ActiveMovie (I.E. DirectShow) control. Default backend on Windows and supported by nearly all Windows versions, even some Windows CE versions. May display a windows media player logo while inactive.
- **wxMEDIABACKEND_QUICKTIME:** Use QuickTime. Mac Only. WARNING: May not working correctly embedded in a wxNotebook.
- **wxMEDIABACKEND_GSTREAMER**, Use GStreamer. Unix Only. Requires GStreamer 0.8 along with at the very least the xvimagesink, xoverlay, and gst-play modules of gstreamer to function. You need the correct modules to play the relevant files, for example the mad module to play mp3s, etc.
- **wxMEDIABACKEND_WMP10**, Uses Windows Media Player 10 (Windows only) - works on mobile machines with Windows Media Player 10 and desktop machines with either Windows Media Player 9 or 10.

Note that other backends such as wxMEDIABACKEND_MCI can now be found at wxCode (`http://wxcode.↩sourceforge.net/`).

21.455.7 Creating a backend

Creating a backend for wxMediaCtrl is a rather simple process. Simply derive from wxMediaBackendCommonBase and implement the methods you want. The methods in wxMediaBackend correspond to those in wxMediaCtrl except for wxMediaCtrl::CreateControl which does the actual creation of the control, in cases where a custom control is not needed you may simply call wxControl::Create().

You need to make sure to use the `DECLARE_CLASS` and `IMPLEMENT_CLASS` macros.

The only real tricky part is that you need to make sure the file in compiled in, which if there are just backends in there will not happen and you may need to use a force link hack (see `http://www.wxwidgets.↩org/wiki/index.php/RTTI`).

This is a rather simple example of how to create a backend in the wxActiveXContainer documentation.

Library: wxMedia

Category: Multimedia

See also

wxMediaEvent

Public Member Functions

- wxMediaCtrl ()

  Default constructor - you MUST call Create() before calling any other methods of wxMediaCtrl.
- wxMediaCtrl (wxWindow *parent, wxWindowID id, const wxString &fileName=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &szBackend=wxEmptyString, const wxValidator &validator=wxDefaultValidator, const wxString &name="mediaCtrl")

  Constructor that calls Create().
- bool Create (wxWindow *parent, wxWindowID id, const wxString &fileName=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &szBackend=wxEmptyString, const wxValidator &validator=wxDefaultValidator, const wxString &name="mediaCtrl")

  Creates this control.
- wxSize GetBestSize () const

  Obtains the best size relative to the original/natural size of the video, if there is any.
- double GetPlaybackRate ()

  Obtains the playback rate, or speed of the media.
- wxMediaState GetState ()

  Obtains the state the playback of the media is in.
- double GetVolume ()

  Gets the volume of the media from a 0.0 to 1.0 range.
- wxFileOffset Length ()

  Obtains the length - the total amount of time the movie has in milliseconds.
- bool Load (const wxString &fileName)

  Loads the file that fileName refers to.
- bool Load (const wxURI &uri)

  Loads the location that uri refers to.
- bool Load (const wxURI &uri, const wxURI &proxy)

  Loads the location that `uri` refers to with the proxy `proxy`.
- bool LoadURI (const wxString &fileName)

  Same as Load(const wxURI& uri).
- bool LoadURIWithProxy (const wxString &fileName, const wxString &proxy)

  Same as Load(const wxURI& uri, const wxURI& proxy).
- bool Pause ()

  Pauses playback of the movie.
- bool Play ()

  Resumes playback of the movie.
- wxFileOffset Seek (wxFileOffset where, wxSeekMode mode=wxFromStart)

  Seeks to a position within the movie.
- bool SetPlaybackRate (double dRate)

  Sets the playback rate, or speed of the media, to that referred by dRate.
- bool SetVolume (double dVolume)

  Sets the volume of the media from a 0.0 to 1.0 range to that referred by `dVolume`.
- bool ShowPlayerControls (wxMediaCtrlPlayerControls flags=wxMEDIACTRLPLAYERCONTROLS_DEFAULT)

  A special feature to wxMediaCtrl.
- bool Stop ()

  Stops the media.
- wxFileOffset Tell ()

  Obtains the current position in time within the movie in milliseconds.

Additional Inherited Members

21.455.8 Constructor & Destructor Documentation

wxMediaCtrl::wxMediaCtrl ( )

Default constructor - you MUST call Create() before calling any other methods of wxMediaCtrl.

wxMediaCtrl::wxMediaCtrl (wxWindow * *parent*, wxWindowID *id*, const wxString & *fileName* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *szBackend* = wxEmptyString, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "mediaCtrl")

Constructor that calls Create().

You may prefer to call Create() directly to check to see if wxMediaCtrl is available on the system.

Parameters

parent	parent of this control. Must not be NULL.
id	id to use for events
fileName	If not empty, the path of a file to open.
pos	Position to put control at.
size	Size to put the control at and to stretch movie to.
style	Optional styles.
szBackend	Name of backend you want to use, leave blank to make wxMediaCtrl figure it out.
validator	validator to use.
name	Window name.

21.455.9 Member Function Documentation

bool wxMediaCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *fileName* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *szBackend* = wxEmptyString, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "mediaCtrl")

Creates this control.

Returns false if it can't load the movie located at *fileName* or it cannot load one of its native backends.

If you specify a file to open via *fileName* and you don't specify a backend to use, wxMediaCtrl tries each of its backends until one that can render the path referred to by *fileName* can be found.

Parameters

parent	parent of this control. Must not be NULL.
id	id to use for events
fileName	If not empty, the path of a file to open.
pos	Position to put control at.
size	Size to put the control at and to stretch movie to.
style	Optional styles.
szBackend	Name of backend you want to use, leave blank to make wxMediaCtrl figure it out.
validator	validator to use.
name	Window name.

wxSize wxMediaCtrl::GetBestSize ( ) const

Obtains the best size relative to the original/natural size of the video, if there is any.

See Video size for more information.

double wxMediaCtrl::GetPlaybackRate ( )

Obtains the playback rate, or speed of the media.

`1.0` represents normal speed, while `2.0` represents twice the normal speed of the media, for example. Not supported on the GStreamer (Unix) backend.

Returns

zero on failure.

wxMediaState wxMediaCtrl::GetState ( )

Obtains the state the playback of the media is in.

wxMEDIASTATE_STOPPED	The movie has stopped.
wxMEDIASTATE_PAUSED	The movie is paused.
wxMEDIASTATE_PLAYING	The movie is currently playing.

double wxMediaCtrl::GetVolume ( )

Gets the volume of the media from a 0.0 to 1.0 range.

Note

Due to rounding and other errors the value returned may not be the exact value sent to SetVolume().

wxFileOffset wxMediaCtrl::Length ( )

Obtains the length - the total amount of time the movie has in milliseconds.

bool wxMediaCtrl::Load (const wxString & *fileName*)

Loads the file that fileName refers to.

Returns false if loading fails.

bool wxMediaCtrl::Load (const wxURI & *uri*)

Loads the location that uri refers to.

Note that this is very implementation-dependent, although HTTP URI/URLs are generally supported, for example. Returns false if loading fails.

bool wxMediaCtrl::Load (const wxURI & *uri,* const wxURI & *proxy*)

Loads the location that `uri` refers to with the proxy `proxy`.

Not implemented on most backends so it should be called with caution. Returns false if loading fails.

bool wxMediaCtrl::LoadURI (const wxString & *fileName*)

Same as Load(const wxURI& uri).

Kept for wxPython compatibility.

bool wxMediaCtrl::LoadURIWithProxy (const wxString & *fileName,* const wxString & *proxy*)

Same as Load(const wxURI& uri, const wxURI& proxy).

Kept for wxPython compatibility.

bool wxMediaCtrl::Pause ( )

Pauses playback of the movie.

bool wxMediaCtrl::Play ( )

Resumes playback of the movie.

wxFileOffset wxMediaCtrl::Seek (wxFileOffset *where,* wxSeekMode *mode* = wxFromStart)

Seeks to a position within the movie.

Todo Document the wxSeekMode parameter *mode*, and perhaps also the wxFileOffset and wxSeekMode themselves.

bool wxMediaCtrl::SetPlaybackRate (double *dRate*)

Sets the playback rate, or speed of the media, to that referred by *dRate*.

`1.0` represents normal speed, while `2.0` represents twice the normal speed of the media, for example. Not supported on the GStreamer (Unix) backend. Returns true if successful.

bool wxMediaCtrl::SetVolume (double *dVolume*)

Sets the volume of the media from a 0.0 to 1.0 range to that referred by `dVolume`.

`1.0` represents full volume, while `0.5` represents half (50 percent) volume, for example.

Note

The volume may not be exact due to conversion and rounding errors, although setting the volume to full or none is always exact. Returns true if successful.

bool wxMediaCtrl::ShowPlayerControls (wxMediaCtrlPlayerControls *flags* = wxMEDIACTRLPLAYERCONTROLS_↩ DEFAULT)

A special feature to wxMediaCtrl.

Applications using native toolkits such as QuickTime usually have a scrollbar, play button, and more provided to them by the toolkit. By default wxMediaCtrl does not do this. However, on the directshow and quicktime backends you can show or hide the native controls provided by the underlying toolkit at will using ShowPlayerControls().

Simply calling the function with default parameters tells wxMediaCtrl to use the default controls provided by the toolkit. The function takes a wxMediaCtrlPlayerControls enumeration, please see available show modes there.

For more info see Player controls.

Currently only implemented on the QuickTime and DirectShow backends. The function returns true on success.

bool wxMediaCtrl::Stop ( )

Stops the media.

See Operation for an overview of how stopping works.

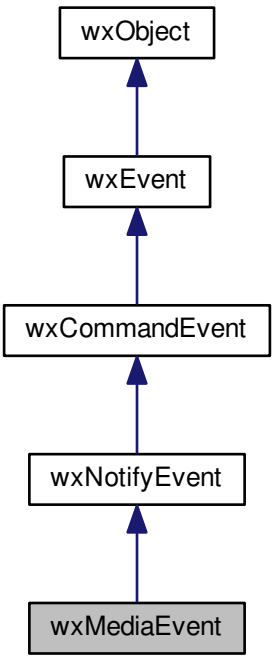
wxFileOffset wxMediaCtrl::Tell ( )

Obtains the current position in time within the movie in milliseconds.

21.456 wxMediaEvent Class Reference

`#include <wx/mediactrl.h>`

Inheritance diagram for wxMediaEvent:



21.456.1 Detailed Description

Event wxMediaCtrl uses.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMediaEvent& event)

Event macros:

- EVT_MEDIA_LOADED(id, func): Sent when a media has loaded enough data that it can start playing. Processes a `wxEVT_MEDIA_LOADED` event type.
- EVT_MEDIA_STOP(id, func): Sent when a media has switched to the `wxMEDIASTATE_STOPPED` state. You may be able to Veto this event to prevent it from stopping, causing it to continue playing - even if it has reached that end of the media (note that this may not have the desired effect - if you want to loop the media, for example, catch the `EVT_MEDIA_FINISHED` and play there instead). Processes a `wxEVT_MEDIA_STOP` event type.
- EVT_MEDIA_FINISHED(id, func): Sent when a media has finished playing in a wxMediaCtrl. Processes a `wxEVT_MEDIA_FINISHED` event type.
- EVT_MEDIA_STATECHANGED(id, func): Sent when a media has switched its state (from any media state). Processes a `wxEVT_MEDIA_STATECHANGED` event type.
- EVT_MEDIA_PLAY(id, func): Sent when a media has switched to the `wxMEDIASTATE_PLAYING` state. Processes a `wxEVT_MEDIA_PLAY` event type.
- EVT_MEDIA_PAUSE(id, func): Sent when a media has switched to the `wxMEDIASTATE_PAUSED` state. Processes a `wxEVT_MEDIA_PAUSE` event type.

Library: wxMedia

Category: Events

Public Member Functions

- wxMediaEvent (wxEventType commandType=wxEVT_NULL, int winid=0)
  Default ctor.

Additional Inherited Members

21.456.2 Constructor & Destructor Documentation

wxMediaEvent::wxMediaEvent (wxEventType *commandType* = wxEVT_NULL, int *winid* = `0`)

Default ctor.

21.457 wxMemoryBuffer Class Reference

```
#include <wx/buffer.h>
```

21.457.1 Detailed Description

A **wxMemoryBuffer** is a useful data structure for storing arbitrary sized blocks of memory.

wxMemoryBuffer guarantees deletion of the memory block when the object is destroyed.

Library: wxBase

Category: Data Structures

Public Member Functions

- wxMemoryBuffer (const wxMemoryBuffer &src)

  Copy constructor, refcounting is used for performance, but wxMemoryBuffer is not a copy-on-write structure so changes made to one buffer effect all copies made from it.
- wxMemoryBuffer (size_t size=1024)

  Create a new buffer.
- void AppendByte (char data)

  Append a single byte to the buffer.
- void AppendData (const void *data, size_t len)

  Single call to append a data block to the buffer.
- void Clear ()

  Clear the buffer contents.
- void * GetAppendBuf (size_t sizeNeeded)

  Ensure that the buffer is big enough and return a pointer to the start of the empty space in the buffer.
- size_t GetBufSize () const

  Returns the size of the buffer.
- void * GetData () const

  Return a pointer to the data in the buffer.
- size_t GetDataLen () const

  Returns the length of the valid data in the buffer.
- void * GetWriteBuf (size_t sizeNeeded)

  Ensure the buffer is big enough and return a pointer to the buffer which can be used to directly write into the buffer up to sizeNeeded bytes.
- bool IsEmpty () const

  Returns true if the buffer contains no data.
- void SetBufSize (size_t size)

  Ensures the buffer has at least size bytes available.
- void SetDataLen (size_t size)

  Sets the length of the data stored in the buffer.
- void UngetAppendBuf (size_t sizeUsed)

  Update the length after completing a direct append, which you must have used GetAppendBuf() to initialise.
- void UngetWriteBuf (size_t sizeUsed)

  Update the buffer after completing a direct write, which you must have used GetWriteBuf() to initialise.

21.457.2 Constructor & Destructor Documentation

wxMemoryBuffer::wxMemoryBuffer (const wxMemoryBuffer & *src*)

Copy constructor, refcounting is used for performance, but wxMemoryBuffer is not a copy-on-write structure so changes made to one buffer effect all copies made from it.

See also

Reference Counting

wxMemoryBuffer::wxMemoryBuffer (size_t *size* = 1024)

Create a new buffer.

Parameters

size	size of the new buffer, 1KiB by default.

21.457.3 Member Function Documentation

void wxMemoryBuffer::AppendByte (char *data*)

Append a single byte to the buffer.

Parameters

data	New byte to append to the buffer.

void wxMemoryBuffer::AppendData (const void * *data,* size_t *len*)

Single call to append a data block to the buffer.

Parameters

data	Pointer to block to append to the buffer.
len	Length of data to append.

void wxMemoryBuffer::Clear ( )

Clear the buffer contents.

The buffer won't contain any data after this method is called.

See also

IsEmpty()

Since

2.9.4

void* wxMemoryBuffer::GetAppendBuf (size_t *sizeNeeded*)

Ensure that the buffer is big enough and return a pointer to the start of the empty space in the buffer.

This pointer can be used to directly write data into the buffer, this new data will be appended to the existing data.

Parameters

sizeNeeded	Amount of extra space required in the buffer for the append operation

size_t wxMemoryBuffer::GetBufSize ( ) const

Returns the size of the buffer.

void* wxMemoryBuffer::GetData ( ) const

Return a pointer to the data in the buffer.

size_t wxMemoryBuffer::GetDataLen ( ) const

Returns the length of the valid data in the buffer.

void* wxMemoryBuffer::GetWriteBuf (size_t *sizeNeeded*)

Ensure the buffer is big enough and return a pointer to the buffer which can be used to directly write into the buffer up to *sizeNeeded* bytes.

bool wxMemoryBuffer::IsEmpty ( ) const

Returns true if the buffer contains no data.

See also

Clear()

Since

2.9.4

void wxMemoryBuffer::SetBufSize (size_t *size*)

Ensures the buffer has at least *size* bytes available.

void wxMemoryBuffer::SetDataLen (size_t *size*)

Sets the length of the data stored in the buffer.

Mainly useful for truncating existing data.

Parameters

size	New length of the valid data in the buffer. This is distinct from the allocated size

void wxMemoryBuffer::UngetAppendBuf (size_t *sizeUsed*)

Update the length after completing a direct append, which you must have used GetAppendBuf() to initialise.

Parameters

sizeUsed	This is the amount of new data that has been appended.

void wxMemoryBuffer::UngetWriteBuf (size_t *sizeUsed*)

Update the buffer after completing a direct write, which you must have used GetWriteBuf() to initialise.

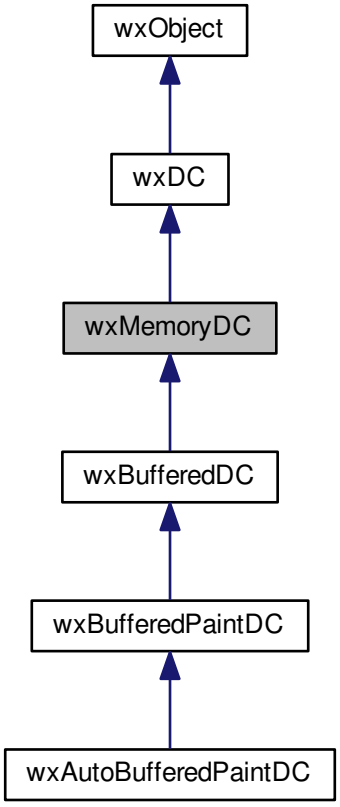
Parameters

sizeUsed	The amount of data written in to buffer by the direct write

21.458 wxMemoryDC Class Reference

`#include <wx/dcmemory.h>`

Inheritance diagram for wxMemoryDC:



21.458.1 Detailed Description

A memory device context provides a means to draw graphics onto a bitmap.

When drawing in to a mono-bitmap, using `wxWHITE`, `wxWHITE_PEN` and `wxWHITE_BRUSH` will draw the background colour (i.e. 0) whereas all other colours will draw the foreground colour (i.e. 1).

A bitmap must be selected into the new memory DC before it may be used for anything. Typical usage is as follows:

```
// Create a memory DC
wxMemoryDC temp_dc;
temp_dc.SelectObject(test_bitmap);

// We can now draw into the memory DC...
// Copy from this DC to another DC.
old_dc.Blit(250, 50, BITMAP_WIDTH, BITMAP_HEIGHT, temp_dc, 0, 0);
```

Note that the memory DC must be deleted (or the bitmap selected out of it) before a bitmap can be reselected into another memory DC.

And, before performing any other operations on the bitmap data, the bitmap must be selected out of the memory DC:

```
temp_dc.SelectObject(wxNullBitmap);
```

This happens automatically when wxMemoryDC object goes out of scope.

Library: wxCore

Category: Device Contexts

See also

wxBitmap, wxDC

Public Member Functions

- wxMemoryDC ()

  Constructs a new memory device context.
- wxMemoryDC (wxDC ∗dc)

  Constructs a new memory device context having the same characteristics as the given existing device context.
- wxMemoryDC (wxBitmap &bitmap)

  Constructs a new memory device context and calls SelectObject() with the given bitmap.
- void SelectObject (wxBitmap &bitmap)

  Works exactly like SelectObjectAsSource() but this is the function you should use when you select a bitmap because you want to modify it, e.g.
- void SelectObjectAsSource (const wxBitmap &bitmap)

  Selects the given bitmap into the device context, to use as the memory bitmap.

Additional Inherited Members

21.458.2 Constructor & Destructor Documentation

wxMemoryDC::wxMemoryDC ( )

Constructs a new memory device context.

Use the wxDC::IsOk() member to test whether the constructor was successful in creating a usable device context. Don't forget to select a bitmap into the DC before drawing on it.

wxMemoryDC::wxMemoryDC (wxDC ∗ *dc*)

Constructs a new memory device context having the same characteristics as the given existing device context.

This constructor creates a memory device context *compatible* with *dc* in wxMSW, the argument is ignored in the other ports. If *dc* is NULL, a device context compatible with the screen is created, just as with the default constructor.

wxMemoryDC::wxMemoryDC (wxBitmap & *bitmap*)

Constructs a new memory device context and calls SelectObject() with the given bitmap.

Use the wxDC::IsOk() member to test whether the constructor was successful in creating a usable device context.

21.458.3 Member Function Documentation

void wxMemoryDC::SelectObject (wxBitmap & *bitmap*)

Works exactly like SelectObjectAsSource() but this is the function you should use when you select a bitmap because you want to modify it, e.g.

drawing on this DC.

Using SelectObjectAsSource() when modifying the bitmap may incur some problems related to wxBitmap being a reference counted object (see Reference Counting).

Before using the updated bitmap data, make sure to select it out of context first either by selecting wxNullBitmap into the device context or destroying the device context entirely.

If the bitmap is already selected in this device context, nothing is done. If it is selected in another context, the function asserts and drawing on the bitmap won't work correctly.

See also

wxDC::DrawBitmap()

void wxMemoryDC::SelectObjectAsSource (const wxBitmap & *bitmap*)

Selects the given bitmap into the device context, to use as the memory bitmap.

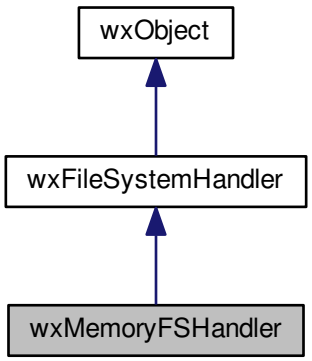
Selecting the bitmap into a memory DC allows you to draw into the DC (and therefore the bitmap) and also to use wxDC::Blit() to copy the bitmap to a window. For this purpose, you may find wxDC::DrawIcon() easier to use instead.

If the argument is wxNullBitmap (or some other uninitialised wxBitmap) the current bitmap is selected out of the device context, and the original bitmap restored, allowing the current bitmap to be destroyed safely.

21.459 wxMemoryFSHandler Class Reference

```
#include <wx/fs_mem.h>
```

Inheritance diagram for wxMemoryFSHandler:



21.459.1 Detailed Description

This wxFileSystem handler can store arbitrary data in memory stream and make them accessible via an URL.

It is particularly suitable for storing bitmaps from resources or included XPM files so that they can be used with wxHTML or wxWebView.

Filenames are prefixed with "memory:", e.g. "memory:myfile.html".

Example:

```
#ifndef __WXMSW__
#include "logo.xpm"
#endif

void MyFrame::OnAbout(wxCommandEvent&)
{
    wxFileSystem::AddHandler(new wxMemoryFSHandler);
    wxMemoryFSHandler::AddFile("logo.png", wxBITMAP(logo),
      wxBITMAP_TYPE_PNG);
    wxMemoryFSHandler::AddFile("about.htm",
                           "<html><body>About: "
                           "<img src=\"memory:logo.png\"></body></html>");

    wxDialog dlg(this, -1, wxString(_("About")));
    wxBoxSizer *topsizer;
    topsizer = new wxBoxSizer(wxVERTICAL);
#ifdef USE_WEBVIEW
    wxWebView* browser = wxWebView::New(&dlg, wxID_ANY,
      wxWebViewDefaultURLStr,
                            wxDefaultPosition, wxSize(380, 160));
    browser->RegisterHandler(wxSharedPtr<wxWebViewHandler>(new
      wxWebViewFSHandler("memory")));
    browser->LoadURL("memory:about.htm");
#else // Use wxHtml
    wxHtmlWindow *browser;
    browser = new wxHtmlWindow(&dlg, -1, wxDefaultPosition,
                               wxSize(380, 160), wxHW_SCROLLBAR_NEVER);
    browser->SetBorders(0);
    browser->LoadPage("memory:about.htm");
    browser->SetSize(browser->GetInternalRepresentation()->
      GetWidth(),
                 browser->GetInternalRepresentation()->
      GetHeight());
#endif
    topsizer->Add(browser, 1, wxALL, 10);
    topsizer->Add(new wxStaticLine(&dlg, -1), 0, wxEXPAND |
      wxLEFT | wxRIGHT, 10);
    topsizer->Add(new wxButton(&dlg, wxID_OK, "Ok"),
                 0, wxALL | wxALIGN_RIGHT, 15);
    dlg.SetAutoLayout(true);
    dlg.SetSizer(topsizer);
    topsizer->Fit(&dlg);
    dlg.Centre();
    dlg.ShowModal();

    wxMemoryFSHandler::RemoveFile("logo.png");
    wxMemoryFSHandler::RemoveFile("about.htm");
}
```

Library: wxBase

Category: Virtual File System

See also

wxMemoryFSHandler::AddFileWithMimeType

Public Member Functions

- wxMemoryFSHandler ()

  Constructor.

Static Public Member Functions

- static void RemoveFile (const wxString &filename)

  Removes a file from memory FS and frees the occupied memory.

- static void AddFile (const wxString &filename, wxImage &image, wxBitmapType type)

  Adds a file to the list of the files stored in memory.

- static void AddFile (const wxString &filename, const wxBitmap &bitmap, wxBitmapType type)

  Adds a file to the list of the files stored in memory.

- static void AddFileWithMimeType (const wxString &filename, const wxString &textdata, const wxString &mimetype)

  Like AddFile(), but lets you explicitly specify added file's MIME type.

- static void AddFileWithMimeType (const wxString &filename, const void *binarydata, size_t size, const wxString &mimetype)

  Like AddFile(), but lets you explicitly specify added file's MIME type.

Additional Inherited Members

21.459.2 Constructor & Destructor Documentation

wxMemoryFSHandler::wxMemoryFSHandler ( )

Constructor.

21.459.3 Member Function Documentation

static void wxMemoryFSHandler::AddFile (const wxString & *filename,* wxImage & *image,* wxBitmapType *type*) `[static]`

Adds a file to the list of the files stored in memory.

Stored data (bitmap, text or raw data) will be copied into private memory stream and available under name "`memory:`" + *filename*.

Note

you must use a *type* value (aka image format) that wxWidgets can save (e.g. JPG, PNG, see wxImage documentation)!

See also

AddFileWithMimeType()

static void wxMemoryFSHandler::AddFile (const wxString & *filename,* const wxBitmap & *bitmap,* wxBitmapType *type*) `[static]`

Adds a file to the list of the files stored in memory.

Stored data (bitmap, text or raw data) will be copied into private memory stream and available under name "`memory:`" + *filename*.

Note

you must use a *type* value (aka image format) that wxWidgets can save (e.g. JPG, PNG, see wxImage documentation)!

See also

AddFileWithMimeType()

static void wxMemoryFSHandler::AddFileWithMimeType (const wxString & *filename,* const wxString & *textdata,* const wxString & *mimetype*) [static]

Like AddFile(), but lets you explicitly specify added file's MIME type.

This version should be used whenever you know the MIME type, because it makes accessing the files faster.

Since

2.8.5

See also

AddFile()

static void wxMemoryFSHandler::AddFileWithMimeType (const wxString & *filename,* const void * *binarydata,* size_t *size,* const wxString & *mimetype*) [static]

Like AddFile(), but lets you explicitly specify added file's MIME type.

This version should be used whenever you know the MIME type, because it makes accessing the files faster.

Since

2.8.5

See also

AddFile()

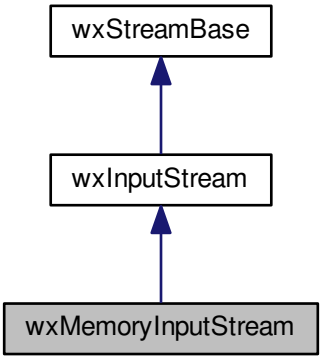
static void wxMemoryFSHandler::RemoveFile (const wxString & *filename*) [static]

Removes a file from memory FS and frees the occupied memory.

21.460 wxMemoryInputStream Class Reference

```
#include <wx/mstream.h>
```

Inheritance diagram for wxMemoryInputStream:



21.460.1 Detailed Description

This class allows to use all methods taking a wxInputStream reference to read in-memory data.

Example:

```
// we've got a block of memory containing a BMP image and we want
// to use wxImage to load it:
wxMemoryInputStream stream(my_memory_block, size);

wxImage theBitmap;
if (!theBitmap.LoadFile(stream, wxBITMAP_TYPE_BMP))
    return;

// we can now safely delete the original memory buffer as the data
// has been decoded by wxImage:
delete [] my_memory_block;
```

Library: wxBase

Category: Streams

See also

wxStreamBuffer, wxMemoryOutputStream

Public Member Functions

- wxMemoryInputStream (const void *data, size_t len)

  Initializes a new read-only memory stream which will use the specified buffer data of length len.
- wxMemoryInputStream (const wxMemoryOutputStream &stream)

  Creates a new read-only memory stream, initializing it with the data from the given output stream stream.
- wxMemoryInputStream (wxInputStream &stream, wxFileOffset len=wxInvalidOffset)

  Creates a new read-only memory stream, initializing it with the data from the given input stream stream.
- virtual ~wxMemoryInputStream ()

  Destructor.

- wxStreamBuffer * GetInputStreamBuffer () const

  Returns the pointer to the stream object used as an internal buffer for that stream.

Additional Inherited Members

21.460.2 Constructor & Destructor Documentation

wxMemoryInputStream::wxMemoryInputStream (const void * *data*, size_t *len*)

Initializes a new read-only memory stream which will use the specified buffer data of length len.

The stream does not take ownership of the buffer, i.e. the buffer will not be deleted in its destructor.

wxMemoryInputStream::wxMemoryInputStream (const wxMemoryOutputStream & *stream*)

Creates a new read-only memory stream, initializing it with the data from the given output stream *stream*.

wxMemoryInputStream::wxMemoryInputStream (wxInputStream & *stream*, wxFileOffset *len* = wxInvalidOffset)

Creates a new read-only memory stream, initializing it with the data from the given input stream *stream*.

The *len* argument specifies the amount of data to read from the *stream*. Setting it to wxInvalidOffset means that the *stream* is to be read entirely (i.e. till the EOF is reached).

virtual wxMemoryInputStream::~wxMemoryInputStream ( ) [virtual]

Destructor.

Does NOT free the buffer provided in the ctor.

21.460.3 Member Function Documentation

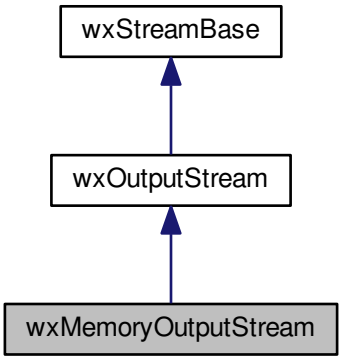
wxStreamBuffer* wxMemoryInputStream::GetInputStreamBuffer ( ) const

Returns the pointer to the stream object used as an internal buffer for that stream.

21.461 wxMemoryOutputStream Class Reference

`#include <wx/mstream.h>`

Inheritance diagram for wxMemoryOutputStream:



21.461.1 Detailed Description

This class allows to use all methods taking a wxOutputStream reference to write to in-memory data.

Example:

```
wxMemoryOutputStream stream;
if (!my_wxImage.SaveFile(stream))
    return;

// now we can access the saved image bytes:
wxStreamBuffer* theBuffer = stream.GetOutputStreamBuffer();
unsigned char byte;
if (theBuffer->Read(byte, 1) != 1)
    return;

// ... do something with 'byte'...

// remember that ~wxMemoryOutputStream will destroy the internal
// buffer since we didn't provide our own when constructing it
```

Library: wxBase

Category: Streams

See also

wxStreamBuffer

Public Member Functions

- wxMemoryOutputStream (void *data=NULL, size_t length=0)

  If data is NULL, then it will initialize a new empty buffer which will grow if required.
- virtual ~wxMemoryOutputStream ()

  Destructor.
- size_t CopyTo (void *buffer, size_t len) const

  Allows you to transfer data from the internal buffer of wxMemoryOutputStream to an external buffer.

- wxStreamBuffer * GetOutputStreamBuffer () const

  Returns the pointer to the stream object used as an internal buffer for this stream.

Additional Inherited Members

21.461.2 Constructor & Destructor Documentation

wxMemoryOutputStream::wxMemoryOutputStream (void * *data* = NULL, size_t *length* = 0)

If *data* is NULL, then it will initialize a new empty buffer which will grow if required.

Warning

If the buffer is created by wxMemoryOutputStream, it will be destroyed at the destruction of the stream.

virtual wxMemoryOutputStream::~wxMemoryOutputStream ( ) [virtual]

Destructor.

If the buffer wasn't provided by the user, it will be deleted here.

21.461.3 Member Function Documentation

size_t wxMemoryOutputStream::CopyTo (void * *buffer*, size_t *len*) const

Allows you to transfer data from the internal buffer of wxMemoryOutputStream to an external buffer.

len specifies the size of the buffer.

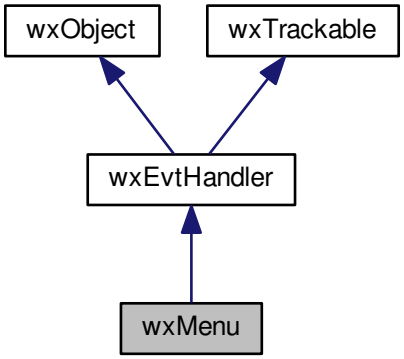
wxStreamBuffer* wxMemoryOutputStream::GetOutputStreamBuffer ( ) const

Returns the pointer to the stream object used as an internal buffer for this stream.

21.462 wxMenu Class Reference

```
#include <wx/menu.h>
```

Inheritance diagram for wxMenu:



21.462.1 Detailed Description

A menu is a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu).

Menus may be used to construct either menu bars or popup menus.

A menu item has an integer ID associated with it which can be used to identify the selection, or to change the menu item in some way. A menu item with a special identifier *wxID_SEPARATOR* is a separator item and doesn't have an associated command but just makes a separator line appear in the menu.

Note

> Please note that *wxID_ABOUT* and *wxID_EXIT* are predefined by wxWidgets and have a special meaning since entries using these IDs will be taken out of the normal menus under OS X and will be inserted into the system menu (following the appropriate OS X interface guideline).

Menu items may be either *normal* items, *check* items or *radio* items. Normal items don't have any special properties while the check items have a boolean flag associated to them and they show a checkmark in the menu when the flag is set. wxWidgets automatically toggles the flag value when the item is clicked and its value may be retrieved using either wxMenu::IsChecked method of wxMenu or wxMenuBar itself or by using wxEvent::IsChecked when you get the menu notification for the item in question.

The radio items are similar to the check items except that all the other items in the same radio group are unchecked when a radio item is checked. The radio group is formed by a contiguous range of radio items, i.e. it starts at the first item of this kind and ends with the first item of a different kind (or the end of the menu). Notice that because the radio groups are defined in terms of the item positions inserting or removing the items in the menu containing the radio items risks to not work correctly.

21.462.2 Allocation strategy

All menus must be created on the **heap** because all menus attached to a menubar or to another menu will be deleted by their parent when it is deleted. The only exception to this rule are the popup menus (i.e. menus used with wxWindow::PopupMenu()) as wxWidgets does not destroy them to allow reusing the same menu more than once. But the exception applies only to the menus themselves and not to any submenus of popup menus which are still destroyed by wxWidgets as usual and so must be heap-allocated.

As the frame menubar is deleted by the frame itself, it means that normally all menus used are deleted automatically.

21.462.3 Event handling

If the menu is part of a menubar, then wxMenuBar event processing is used.

With a popup menu (see wxWindow::PopupMenu), there is a variety of ways to handle a menu selection event (`wxEVT_MENU`):

- Provide `EVT_MENU` handlers in the window which pops up the menu, or in an ancestor of that window (the simplest method);
- Derive a new class from wxMenu and define event table entries using the `EVT_MENU` macro;
- Set a new event handler for wxMenu, through wxEvtHandler::SetNextHandler, specifying an object whose class has `EVT_MENU` entries;

Note that instead of static `EVT_MENU` macros you can also use dynamic connection; see Dynamic Event Handling.

Library: wxCore

Category: Menus

See also

wxMenuBar, wxWindow::PopupMenu, Events and Event Handling, wxFileHistory (most recently used files menu)

Public Member Functions

- wxMenu ()

  Constructs a wxMenu object.
- wxMenu (long style)

  Constructs a wxMenu object.
- wxMenu (const wxString &title, long style=0)

  Constructs a wxMenu object with a title.
- virtual ~wxMenu ()

  Destructor, destroying the menu.
- wxMenuItem * Append (int id, const wxString &item=wxEmptyString, const wxString &helpString=wxEmptyString, wxItemKind kind=wxITEM_NORMAL)

  Adds a menu item.
- wxMenuItem * Append (int id, const wxString &item, wxMenu *subMenu, const wxString &helpString=wxEmptyString)

  Adds a submenu.
- wxMenuItem * Append (wxMenuItem *menuItem)

  Adds a menu item object.
- wxMenuItem * AppendCheckItem (int id, const wxString &item, const wxString &help=wxEmptyString)

  Adds a checkable item to the end of the menu.
- wxMenuItem * AppendRadioItem (int id, const wxString &item, const wxString &help=wxEmptyString)

  Adds a radio item to the end of the menu.
- wxMenuItem * AppendSeparator ()

  Adds a separator to the end of the menu.
- wxMenuItem * AppendSubMenu (wxMenu *submenu, const wxString &text, const wxString &help=wxEmptyString)

  Adds the given submenu to this menu.

- virtual void Break ()

  Inserts a break in a menu, causing the next appended item to appear in a new column.
- void Check (int id, bool check)

  Checks or unchecks the menu item.
- bool Delete (int id)

  Deletes the menu item from the menu.
- bool Delete (wxMenuItem *item)

  Deletes the menu item from the menu.
- bool Destroy (int id)

  Deletes the menu item from the menu.
- bool Destroy (wxMenuItem *item)

  Deletes the menu item from the menu.
- void Enable (int id, bool enable)

  Enables or disables (greys out) a menu item.
- wxMenuItem * FindChildItem (int id, size_t *pos=NULL) const

  Finds the menu item object associated with the given menu item identifier and, optionally, the position of the item in the menu.
- virtual int FindItem (const wxString &itemString) const

  Finds the menu id for a menu item string.
- wxMenuItem * FindItem (int id, wxMenu **menu=NULL) const

  Finds the menu item object associated with the given menu item identifier and, optionally, the (sub)menu it belongs to.
- wxMenuItem * FindItemByPosition (size_t position) const

  Returns the wxMenuItem given a position in the menu.
- virtual wxString GetHelpString (int id) const

  Returns the help string associated with a menu item.
- wxString GetLabel (int id) const

  Returns a menu item label.
- wxString GetLabelText (int id) const

  Returns a menu item label, without any of the original mnemonics and accelerators.
- size_t GetMenuItemCount () const

  Returns the number of items in the menu.
- const wxString & GetTitle () const

  Returns the title of the menu.
- wxMenuItem * Insert (size_t pos, wxMenuItem *menuItem)

  Inserts the given item before the position pos.
- wxMenuItem * Insert (size_t pos, int id, const wxString &item=wxEmptyString, const wxString &help↩String=wxEmptyString, wxItemKind kind=wxITEM_NORMAL)

  Inserts the given item before the position pos.
- wxMenuItem * Insert (size_t pos, int id, const wxString &text, wxMenu *submenu, const wxString &help=wx↩EmptyString)

  Inserts the given submenu before the position pos.
- wxMenuItem * InsertCheckItem (size_t pos, int id, const wxString &item, const wxString &helpString=wx↩EmptyString)

  Inserts a checkable item at the given position.
- wxMenuItem * InsertRadioItem (size_t pos, int id, const wxString &item, const wxString &helpString=wx↩EmptyString)

  Inserts a radio item at the given position.
- wxMenuItem * InsertSeparator (size_t pos)

  Inserts a separator at the given position.
- bool IsChecked (int id) const

Determines whether a menu item is checked.

- bool IsEnabled (int id) const

  Determines whether a menu item is enabled.

- wxMenuItem * Prepend (wxMenuItem *item)

  Inserts the given item at position 0, i.e. before all the other existing items.

- wxMenuItem * Prepend (int id, const wxString &item=wxEmptyString, const wxString &helpString=wxEmptyString, wxItemKind kind=wxITEM_NORMAL)

  Inserts the given item at position 0, i.e. before all the other existing items.

- wxMenuItem * Prepend (int id, const wxString &text, wxMenu *submenu, const wxString &help=wxEmptyString)

  Inserts the given submenu at position 0.

- wxMenuItem * PrependCheckItem (int id, const wxString &item, const wxString &helpString=wxEmptyString)

  Inserts a checkable item at position 0.

- wxMenuItem * PrependRadioItem (int id, const wxString &item, const wxString &helpString=wxEmptyString)

  Inserts a radio item at position 0.

- wxMenuItem * PrependSeparator ()

  Inserts a separator at position 0.

- wxMenuItem * Remove (int id)

  Removes the menu item from the menu but doesn't delete the associated C++ object.

- wxMenuItem * Remove (wxMenuItem *item)

  Removes the menu item from the menu but doesn't delete the associated C++ object.

- virtual void SetHelpString (int id, const wxString &helpString)

  Sets an item's help string.

- void SetLabel (int id, const wxString &label)

  Sets the label of a menu item.

- virtual void SetTitle (const wxString &title)

  Sets the title of the menu.

- void UpdateUI (wxEvtHandler *source=NULL)

  Sends events to source (or owning window if NULL) to update the menu UI.

- void SetInvokingWindow (wxWindow *win)
- wxWindow * GetInvokingWindow () const
- wxWindow * GetWindow () const
- long GetStyle () const
- void SetParent (wxMenu *parent)
- wxMenu * GetParent () const
- virtual void Attach (wxMenuBar *menubar)
- virtual void Detach ()
- bool IsAttached () const

- wxMenuItemList & GetMenuItems ()

  Returns the list of items in the menu.

- const wxMenuItemList & GetMenuItems () const

  Returns the list of items in the menu.

Additional Inherited Members

21.462.4 Constructor & Destructor Documentation

wxMenu::wxMenu ( )

Constructs a wxMenu object.

wxMenu::wxMenu (long *style*)

Constructs a wxMenu object.

Parameters

style	If set to wxMENU_TEAROFF, the menu will be detachable (wxGTK and wxQT only).

wxMenu::wxMenu (const wxString & *title*, long *style* = 0)

Constructs a wxMenu object with a title.

Parameters

title	Title at the top of the menu (not always supported).
style	If set to wxMENU_TEAROFF, the menu will be detachable (wxGTK and wxQT only).

virtual wxMenu::~wxMenu ( ) [virtual]

Destructor, destroying the menu.

Note

Under Motif, a popup menu must have a valid parent (the window it was last popped up on) when being destroyed. Therefore, make sure you delete or re-use the popup menu *before* destroying the parent window. Re-use in this context means popping up the menu on a different window from last time, which causes an implicit destruction and recreation of internal data structures.

21.462.5 Member Function Documentation

wxMenuItem* wxMenu::Append (int *id*, const wxString & *item* = wxEmptyString, const wxString & *helpString* = wxEmptyString, wxItemKind *kind* = wxITEM_NORMAL)

Adds a menu item.

Parameters

id	The menu command identifier.
item	The string to appear on the menu item. See wxMenuItem::SetItemLabel() for more details.
helpString	An optional help string associated with the item. By default, the handler for the wxEVT_MENU_HIGHLIGHT event displays this string in the status line.
kind	May be wxITEM_SEPARATOR, wxITEM_NORMAL, wxITEM_CHECK or wxITEM_RADIO.

Example:

```
m_pFileMenu->Append(ID_NEW_FILE, "&New file\tCTRL+N", "Creates a new XYZ document");
```

or even better for stock menu items (see wxMenuItem::wxMenuItem):

```
m_pFileMenu->Append(wxID_NEW, "", "Creates a new XYZ document");
```

Remarks

This command can be used after the menu has been shown, as well as on initial creation of a menu or menubar.

See also

AppendSeparator(), AppendCheckItem(), AppendRadioItem(), AppendSubMenu(), Insert(), SetLabel(), GetHelpString(), SetHelpString(), wxMenuItem

wxMenuItem* wxMenu::Append (int *id,* const wxString & *item,* wxMenu * *subMenu,* const wxString & *helpString* = wxEmptyString)

Adds a submenu.

Deprecated This function is deprecated, use AppendSubMenu() instead.

Parameters

id	The menu command identifier.
item	The string to appear on the menu item.
subMenu	Pull-right submenu.
helpString	An optional help string associated with the item. By default, the handler for the `wxEVT_MENU_HIGHLIGHT` event displays this string in the status line.

See also

AppendSeparator(), AppendCheckItem(), AppendRadioItem(), AppendSubMenu(), Insert(), SetLabel(), GetHelpString(), SetHelpString(), wxMenuItem

wxMenuItem* wxMenu::Append (wxMenuItem * *menuItem*)

Adds a menu item object.

This is the most generic variant of Append() method because it may be used for both items (including separators) and submenus and because you can also specify various extra properties of a menu item this way, such as bitmaps and fonts.

Parameters

menuItem	A menuitem object. It will be owned by the wxMenu object after this function is called, so do not delete it yourself.

Remarks

See the remarks for the other Append() overloads.

See also

AppendSeparator(), AppendCheckItem(), AppendRadioItem(), AppendSubMenu(), Insert(), SetLabel(), GetHelpString(), SetHelpString(), wxMenuItem

wxMenuItem* wxMenu::AppendCheckItem (int *id,* const wxString & *item,* const wxString & *help* = wxEmptyString)

Adds a checkable item to the end of the menu.

See also

Append(), InsertCheckItem()

wxMenuItem* wxMenu::AppendRadioItem (int *id,* const wxString & *item,* const wxString & *help* = wxEmptyString)

Adds a radio item to the end of the menu.

All consequent radio items form a group and when an item in the group is checked, all the others are automatically unchecked.

Note

Radio items are not supported under wxMotif.

See also

Append(), InsertRadioItem()

wxMenuItem* wxMenu::AppendSeparator ( )

Adds a separator to the end of the menu.

See also

Append(), InsertSeparator()

wxMenuItem* wxMenu::AppendSubMenu (wxMenu * *submenu,* const wxString & *text,* const wxString & *help =* wxEmptyString)

Adds the given *submenu* to this menu.

text is the text shown in the menu for it and *help* is the help string shown in the status bar when the submenu item is selected.

See also

Insert(), Prepend()

virtual void wxMenu::Attach (wxMenuBar * *menubar*) [virtual]

virtual void wxMenu::Break ( ) [virtual]

Inserts a break in a menu, causing the next appended item to appear in a new column.

void wxMenu::Check (int *id,* bool *check*)

Checks or unchecks the menu item.

Parameters

id	The menu item identifier.
check	If true, the item will be checked, otherwise it will be unchecked.

See also

IsChecked()

bool wxMenu::Delete (int *id*)

Deletes the menu item from the menu.

If the item is a submenu, it will **not** be deleted. Use Destroy() if you want to delete a submenu.

Parameters

id	Id of the menu item to be deleted.

See also

FindItem(), Destroy(), Remove()

bool wxMenu::Delete (wxMenuItem * *item*)

Deletes the menu item from the menu.

If the item is a submenu, it will **not** be deleted. Use Destroy() if you want to delete a submenu.

Parameters

item	Menu item to be deleted.

See also

FindItem(), Destroy(), Remove()

bool wxMenu::Destroy (int *id*)

Deletes the menu item from the menu.

If the item is a submenu, it will be deleted. Use Remove() if you want to keep the submenu (for example, to reuse it later).

Parameters

id	Id of the menu item to be deleted.

See also

FindItem(), Delete(), Remove()

bool wxMenu::Destroy (wxMenuItem * *item*)

Deletes the menu item from the menu.

If the item is a submenu, it will be deleted. Use Remove() if you want to keep the submenu (for example, to reuse it later).

Parameters

item	Menu item to be deleted.

See also

FindItem(), Delete(), Remove()

virtual void wxMenu::Detach ( ) [virtual]

void wxMenu::Enable (int *id,* bool *enable*)

Enables or disables (greys out) a menu item.

Parameters

id	The menu item identifier.
enable	true to enable the menu item, false to disable it.

See also

IsEnabled()

wxMenuItem* wxMenu::FindChildItem (int *id*, size_t * *pos* = NULL) const

Finds the menu item object associated with the given menu item identifier and, optionally, the position of the item in the menu.

Unlike FindItem(), this function doesn't recurse but only looks at the direct children of this menu.

Parameters

id	The identifier of the menu item to find.
pos	If the pointer is not NULL, it is filled with the item's position if it was found or (size_t)wxNOT_FOUND otherwise.

Returns

Menu item object or NULL if not found.

virtual int wxMenu::FindItem (const wxString & *itemString*) const [virtual]

Finds the menu id for a menu item string.

Parameters

itemString	Menu item string to find.

Returns

Menu item identifier, or wxNOT_FOUND if none is found.

Remarks

Any special menu codes are stripped out of source and target strings before matching.

wxMenuItem* wxMenu::FindItem (int *id*, wxMenu ** *menu* = NULL) const

Finds the menu item object associated with the given menu item identifier and, optionally, the (sub)menu it belongs to.

Parameters

id	Menu item identifier.
menu	If the pointer is not NULL, it will be filled with the item's parent menu (if the item was found)

Returns

Menu item object or NULL if none is found.

wxMenuItem* wxMenu::FindItemByPosition (size_t *position*) const

Returns the wxMenuItem given a position in the menu.

virtual wxString wxMenu::GetHelpString (int *id*) const `[virtual]`

Returns the help string associated with a menu item.

Parameters

id	The menu item identifier.

Returns

The help string, or the empty string if there is no help string or the item was not found.

See also

SetHelpString(), Append()

wxWindow* wxMenu::GetInvokingWindow ( ) const

wxString wxMenu::GetLabel (int *id*) const

Returns a menu item label.

Parameters

id	The menu item identifier.

Returns

The item label, or the empty string if the item was not found.

See also

GetLabelText(), SetLabel()

wxString wxMenu::GetLabelText (int *id*) const

Returns a menu item label, without any of the original mnemonics and accelerators.

Parameters

id	The menu item identifier.

Returns

The item label, or the empty string if the item was not found.

See also

GetLabel(), SetLabel()

size_t wxMenu::GetMenuItemCount ( ) const

Returns the number of items in the menu.

wxMenuItemList& wxMenu::GetMenuItems ( )

Returns the list of items in the menu.

wxMenuItemList is a pseudo-template list class containing wxMenuItem pointers, see wxList.

const wxMenuItemList& wxMenu::GetMenuItems ( ) const

Returns the list of items in the menu.

wxMenuItemList is a pseudo-template list class containing wxMenuItem pointers, see wxList.

wxMenu* wxMenu::GetParent ( ) const

long wxMenu::GetStyle ( ) const

const wxString& wxMenu::GetTitle ( ) const

Returns the title of the menu.

See also

SetTitle()

wxWindow* wxMenu::GetWindow ( ) const

wxMenuItem* wxMenu::Insert (size_t *pos*, wxMenuItem * *menuItem*)

Inserts the given *item* before the position *pos*.

Inserting the item at position GetMenuItemCount() is the same as appending it.

See also

Append(), Prepend()

wxMenuItem* wxMenu::Insert (size_t *pos*, int *id*, const wxString & *item* = wxEmptyString, const wxString & *helpString* = wxEmptyString, wxItemKind *kind* = wxITEM_NORMAL)

Inserts the given *item* before the position *pos*.

Inserting the item at position GetMenuItemCount() is the same as appending it.

See also

Append(), Prepend()

wxMenuItem* wxMenu::Insert (size_t *pos*, int *id*, const wxString & *text*, wxMenu * *submenu*, const wxString & *help* = wxEmptyString)

Inserts the given *submenu* before the position *pos*.

text is the text shown in the menu for it and *help* is the help string shown in the status bar when the submenu item is selected.

See also

AppendSubMenu(), Prepend()

wxMenuItem* wxMenu::InsertCheckItem (size_t *pos*, int *id*, const wxString & *item*, const wxString & *helpString* = wxEmptyString)

Inserts a checkable item at the given position.

See also

Insert(), AppendCheckItem()

wxMenuItem* wxMenu::InsertRadioItem (size_t *pos*, int *id*, const wxString & *item*, const wxString & *helpString* = wxEmptyString)

Inserts a radio item at the given position.

See also

Insert(), AppendRadioItem()

wxMenuItem* wxMenu::InsertSeparator (size_t *pos*)

Inserts a separator at the given position.

See also

Insert(), AppendSeparator()

bool wxMenu::IsAttached ( ) const

bool wxMenu::IsChecked (int *id*) const

Determines whether a menu item is checked.

Parameters

id	The menu item identifier.

Returns

true if the menu item is checked, false otherwise.

See also

Check()

bool wxMenu::IsEnabled (int *id*) const

Determines whether a menu item is enabled.

Parameters

id	The menu item identifier.

Returns

true if the menu item is enabled, false otherwise.

See also

Enable()

wxMenuItem* wxMenu::Prepend (wxMenuItem * *item*)

Inserts the given *item* at position 0, i.e. before all the other existing items.

See also

Append(), Insert()

wxMenuItem* wxMenu::Prepend (int *id*, const wxString & *item* = wxEmptyString, const wxString & *helpString* = wxEmptyString, wxItemKind *kind* = wxITEM_NORMAL)

Inserts the given *item* at position 0, i.e. before all the other existing items.

See also

Append(), Insert()

wxMenuItem* wxMenu::Prepend (int *id*, const wxString & *text*, wxMenu * *submenu*, const wxString & *help* = wxEmptyString)

Inserts the given *submenu* at position 0.

See also

AppendSubMenu(), Insert()

wxMenuItem* wxMenu::PrependCheckItem (int *id*, const wxString & *item*, const wxString & *helpString* = wxEmptyString)

Inserts a checkable item at position 0.

See also

Prepend(), AppendCheckItem()

wxMenuItem* wxMenu::PrependRadioItem (int *id*, const wxString & *item*, const wxString & *helpString* = wxEmptyString)

Inserts a radio item at position 0.

See also

Prepend(), AppendRadioItem()

wxMenuItem* wxMenu::PrependSeparator ( )

Inserts a separator at position 0.

See also

Prepend(), AppendSeparator()

wxMenuItem* wxMenu::Remove (int *id*)

Removes the menu item from the menu but doesn't delete the associated C++ object.

This allows you to reuse the same item later by adding it back to the menu (especially useful with submenus).

Parameters

id	The identifier of the menu item to remove.

Returns

A pointer to the item which was detached from the menu.

wxMenuItem* wxMenu::Remove (wxMenuItem * *item*)

Removes the menu item from the menu but doesn't delete the associated C++ object.

This allows you to reuse the same item later by adding it back to the menu (especially useful with submenus).

Parameters

item	The menu item to remove.

Returns

A pointer to the item which was detached from the menu.

virtual void wxMenu::SetHelpString (int *id*, const wxString & *helpString*) [virtual]

Sets an item's help string.

Parameters

id	The menu item identifier.
helpString	The help string to set.

See also

GetHelpString()

void wxMenu::SetInvokingWindow (wxWindow * *win*)

void wxMenu::SetLabel (int *id*, const wxString & *label*)

Sets the label of a menu item.

Parameters

id	The menu item identifier.
label	The menu item label to set.

See also

Append(), GetLabel()

void wxMenu::SetParent (wxMenu * *parent*)

virtual void wxMenu::SetTitle (const wxString & *title*) [virtual]

Sets the title of the menu.

Parameters

title	The title to set.

Remarks

Notice that you can only call this method directly for the popup menus, to change the title of a menu that is part of a menu bar you need to use wxMenuBar::SetLabelTop().

See also

GetTitle()

void wxMenu::UpdateUI (wxEvtHandler * *source* = NULL)

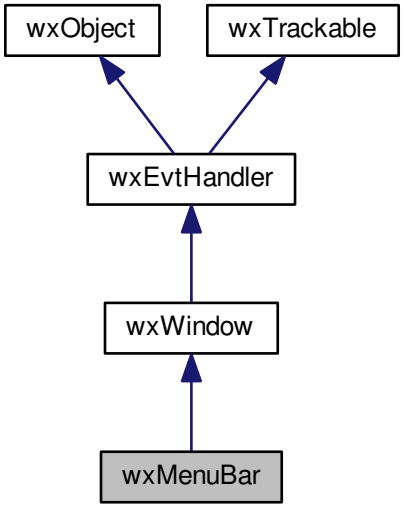
Sends events to *source* (or owning window if NULL) to update the menu UI.

This is called just before the menu is popped up with wxWindow::PopupMenu, but the application may call it at other times if required.

21.463 wxMenuBar Class Reference

```
#include <wx/menu.h>
```

Inheritance diagram for wxMenuBar:



21.463.1 Detailed Description

A menu bar is a series of menus accessible from the top of a frame.

Remarks

To respond to a menu selection, provide a handler for EVT_MENU, in the frame that contains the menu bar.

If you have a toolbar which uses the same identifiers as your EVT_MENU entries, events from the toolbar will also be processed by your EVT_MENU event handlers.

Tip: under Windows, if you discover that menu shortcuts (for example, Alt-F to show the file menu) are not working, check any EVT_CHAR events you are handling in child windows. If you are not calling event.Skip() for events that you don't process in these event handlers, menu shortcuts may cease to work.

Library: wxCore

Category: Menus

See also

wxMenu, Events and Event Handling

Public Member Functions

- wxMenuBar (long style=0)

  Construct an empty menu bar.
- wxMenuBar (size_t n, wxMenu *menus[], const wxString titles[], long style=0)

  Construct a menu bar from arrays of menus and titles.

- virtual ~wxMenuBar ()

  Destructor, destroying the menu bar and removing it from the parent frame (if any).
- virtual bool Append (wxMenu *menu, const wxString &title)

  Adds the item to the end of the menu bar.
- void Check (int id, bool check)

  Checks or unchecks a menu item.
- void Enable (int id, bool enable)

  Enables or disables (greys out) a menu item.
- bool IsEnabledTop (size_t pos) const

  Returns true if the menu with the given index is enabled.
- virtual void EnableTop (size_t pos, bool enable)

  Enables or disables a whole menu.
- virtual wxMenuItem * FindItem (int id, wxMenu **menu=NULL) const

  Finds the menu item object associated with the given menu item identifier.
- int FindMenu (const wxString &title) const

  Returns the index of the menu with the given title or `wxNOT_FOUND` if no such menu exists in this menubar.
- virtual int FindMenuItem (const wxString &menuString, const wxString &itemString) const

  Finds the menu item id for a menu name/menu item string pair.
- wxString GetHelpString (int id) const

  Gets the help string associated with the menu item identifier.
- wxString GetLabel (int id) const

  Gets the label associated with a menu item.
- wxString GetLabelTop (size_t pos) const

  Returns the label of a top-level menu.
- wxMenu * GetMenu (size_t menuIndex) const

  Returns the menu at menuIndex (zero-based).
- size_t GetMenuCount () const

  Returns the number of menus in this menubar.
- virtual wxString GetMenuLabel (size_t pos) const

  Returns the label of a top-level menu.
- virtual wxString GetMenuLabelText (size_t pos) const

  Returns the label of a top-level menu.
- virtual bool Insert (size_t pos, wxMenu *menu, const wxString &title)

  Inserts the menu at the given position into the menu bar.
- bool IsChecked (int id) const

  Determines whether an item is checked.
- bool IsEnabled (int id) const

  Determines whether an item is enabled.
- virtual void Refresh (bool eraseBackground=true, const wxRect *rect=NULL)

  Redraw the menu bar.
- virtual wxMenu * Remove (size_t pos)

  Removes the menu from the menu bar and returns the menu object - the caller is responsible for deleting it.
- virtual wxMenu * Replace (size_t pos, wxMenu *menu, const wxString &title)

  Replaces the menu at the given position with another one.
- void SetHelpString (int id, const wxString &helpString)

  Sets the help string associated with a menu item.
- void SetLabel (int id, const wxString &label)

  Sets the label of a menu item.
- void SetLabelTop (size_t pos, const wxString &label)

  Sets the label of a top-level menu.
- virtual void SetMenuLabel (size_t pos, const wxString &label)

Sets the label of a top-level menu.

- wxMenu * OSXGetAppleMenu () const

  Returns the Apple menu.
- wxFrame * GetFrame () const
- bool IsAttached () const
- virtual void Attach (wxFrame *frame)
- virtual void Detach ()

Static Public Member Functions

- static void MacSetCommonMenuBar (wxMenuBar *menubar)

  Enables you to set the global menubar on Mac, that is, the menubar displayed when the app is running without any frames open.
- static wxMenuBar * MacGetCommonMenuBar ()

  Enables you to get the global menubar on Mac, that is, the menubar displayed when the app is running without any frames open.

Additional Inherited Members

21.463.2 Constructor & Destructor Documentation

wxMenuBar::wxMenuBar (long *style* = 0)

Construct an empty menu bar.

Parameters

style	If wxMB_DOCKABLE the menu bar can be detached (wxGTK only).

wxMenuBar::wxMenuBar (size_t *n*, wxMenu * *menus[]*, const wxString *titles[]*, long *style* = 0)

Construct a menu bar from arrays of menus and titles.

Parameters

n	The number of menus.
menus	An array of menus. Do not use this array again - it now belongs to the menu bar.
titles	An array of title strings. Deallocate this array after creating the menu bar.
style	If wxMB_DOCKABLE the menu bar can be detached (wxGTK only).

wxPerl Note: Not supported by wxPerl.

virtual wxMenuBar::~wxMenuBar ( ) [virtual]

Destructor, destroying the menu bar and removing it from the parent frame (if any).

21.463.3 Member Function Documentation

virtual bool wxMenuBar::Append (wxMenu * *menu*, const wxString & *title*) [virtual]

Adds the item to the end of the menu bar.

Parameters

menu	The menu to add. Do not deallocate this menu after calling Append().
title	The title of the menu, must be non-empty.

Returns

true on success, false if an error occurred.

See also

Insert()

virtual void wxMenuBar::Attach (wxFrame * *frame*) [virtual]

void wxMenuBar::Check (int *id*, bool *check*)

Checks or unchecks a menu item.

Parameters

id	The menu item identifier.
check	If true, checks the menu item, otherwise the item is unchecked.

Remarks

Only use this when the menu bar has been associated with a frame; otherwise, use the wxMenu equivalent call.

virtual void wxMenuBar::Detach ( ) [virtual]

void wxMenuBar::Enable (int *id*, bool *enable*)

Enables or disables (greys out) a menu item.

Parameters

id	The menu item identifier.
enable	true to enable the item, false to disable it.

Remarks

Only use this when the menu bar has been associated with a frame; otherwise, use the wxMenu equivalent call.

virtual void wxMenuBar::EnableTop (size_t *pos*, bool *enable*) [virtual]

Enables or disables a whole menu.

Parameters

pos	The position of the menu, starting from zero.
enable	true to enable the menu, false to disable it.

Remarks

Only use this when the menu bar has been associated with a frame.

virtual wxMenuItem* wxMenuBar::FindItem (int *id,* wxMenu ** *menu =* NULL) const [virtual]

Finds the menu item object associated with the given menu item identifier.

Parameters

id	Menu item identifier.
menu	If not NULL, menu will get set to the associated menu.

Returns

The found menu item object, or NULL if one was not found.

wxPerl Note: In wxPerl this method takes just the *id* parameter; in scalar context it returns the associated `Wx::MenuItem`, in list context it returns a 2-element list (item, submenu).

int wxMenuBar::FindMenu (const wxString & *title*) const

Returns the index of the menu with the given *title* or `wxNOT_FOUND` if no such menu exists in this menubar.

The *title* parameter may specify either the menu title (with accelerator characters, i.e. `"&File"`) or just the menu label (`"File"`) indifferently.

virtual int wxMenuBar::FindMenuItem (const wxString & *menuString,* const wxString & *itemString*) const [virtual]

Finds the menu item id for a menu name/menu item string pair.

Parameters

menuString	Menu title to find.
itemString	Item to find.

Returns

The menu item identifier, or wxNOT_FOUND if none was found.

Remarks

Any special menu codes are stripped out of source and target strings before matching.

wxFrame* wxMenuBar::GetFrame ( ) const

wxString wxMenuBar::GetHelpString (int *id*) const

Gets the help string associated with the menu item identifier.

Parameters

id	The menu item identifier.

Returns

The help string, or the empty string if there was no help string or the menu item was not found.

See also

SetHelpString()

wxString wxMenuBar::GetLabel (int *id*) const

Gets the label associated with a menu item.

Parameters

id	The menu item identifier.

Returns

The menu item label, or the empty string if the item was not found.

Remarks

Use only after the menubar has been associated with a frame.

wxString wxMenuBar::GetLabelTop (size_t *pos*) const

Returns the label of a top-level menu.

Note that the returned string does not include the accelerator characters which could have been specified in the menu title string during its construction.

Parameters

pos	Position of the menu on the menu bar, starting from zero.

Returns

The menu label, or the empty string if the menu was not found.

Remarks

Use only after the menubar has been associated with a frame.

Deprecated This function is deprecated in favour of GetMenuLabel() and GetMenuLabelText().

See also

SetLabelTop()

wxMenu* wxMenuBar::GetMenu (size_t *menuIndex*) const

Returns the menu at *menuIndex* (zero-based).

size_t wxMenuBar::GetMenuCount ( ) const

Returns the number of menus in this menubar.

virtual wxString wxMenuBar::GetMenuLabel (size_t *pos*) const [virtual]

Returns the label of a top-level menu.

Note that the returned string includes the accelerator characters that have been specified in the menu title string during its construction.

Parameters

pos	Position of the menu on the menu bar, starting from zero.

Returns

The menu label, or the empty string if the menu was not found.

Remarks

Use only after the menubar has been associated with a frame.

See also

GetMenuLabelText(), SetMenuLabel()

virtual wxString wxMenuBar::GetMenuLabelText (size_t *pos*) const [virtual]

Returns the label of a top-level menu.

Note that the returned string does not include any accelerator characters that may have been specified in the menu title string during its construction.

Parameters

pos	Position of the menu on the menu bar, starting from zero.

Returns

The menu label, or the empty string if the menu was not found.

Remarks

Use only after the menubar has been associated with a frame.

See also

GetMenuLabel(), SetMenuLabel()

virtual bool wxMenuBar::Insert (size_t *pos*, wxMenu * *menu*, const wxString & *title*) [virtual]

Inserts the menu at the given position into the menu bar.

Inserting menu at position 0 will insert it in the very beginning of it, inserting at position GetMenuCount() is the same as calling Append().

Parameters

pos	The position of the new menu in the menu bar
menu	The menu to add. wxMenuBar owns the menu and will free it.
title	The title of the menu.

Returns

true on success, false if an error occurred.

See also

Append()

bool wxMenuBar::IsAttached ( ) const

bool wxMenuBar::IsChecked (int *id*) const

Determines whether an item is checked.

Parameters

id	The menu item identifier.

Returns

true if the item was found and is checked, false otherwise.

bool wxMenuBar::IsEnabled (int *id*) const

Determines whether an item is enabled.

Parameters

id	The menu item identifier.

Returns

true if the item was found and is enabled, false otherwise.

bool wxMenuBar::IsEnabledTop (size_t *pos*) const

Returns true if the menu with the given index is enabled.

Parameters

pos	The menu position (0-based)

Since

2.9.4

static wxMenuBar* wxMenuBar::MacGetCommonMenuBar ( ) [static]

Enables you to get the global menubar on Mac, that is, the menubar displayed when the app is running without any frames open.

Returns

The global menubar.

Remarks

Only exists on Mac, other platforms do not have this method.

Availability: only available for the wxOSX port.

static void wxMenuBar::MacSetCommonMenuBar (wxMenuBar * *menubar*) [static]

Enables you to set the global menubar on Mac, that is, the menubar displayed when the app is running without any frames open.

Parameters

menubar	The menubar to set.

Remarks

Only exists on Mac, other platforms do not have this method.

Availability: only available for the wxOSX port.

wxMenu* wxMenuBar::OSXGetAppleMenu ( ) const

Returns the Apple menu.

This is the leftmost menu with application's name as its title. You shouldn't remove any items from it, but it is safe to insert extra menu items or submenus into it.

Availability: only available for the wxOSX port.

Since

3.0.1

virtual void wxMenuBar::Refresh (bool *eraseBackground* = `true`, const wxRect * *rect* = `NULL`) `[virtual]`

Redraw the menu bar.

Reimplemented from wxWindow.

virtual wxMenu* wxMenuBar::Remove (size_t *pos*) `[virtual]`

Removes the menu from the menu bar and returns the menu object - the caller is responsible for deleting it.

This function may be used together with Insert() to change the menubar dynamically.

See also

Replace()

virtual wxMenu* wxMenuBar::Replace (size_t *pos*, wxMenu * *menu*, const wxString & *title*) `[virtual]`

Replaces the menu at the given position with another one.

Parameters

pos	The position of the new menu in the menu bar
menu	The menu to add.
title	The title of the menu.

Returns

The menu which was previously at position pos. The caller is responsible for deleting it.

See also

Insert(), Remove()

void wxMenuBar::SetHelpString (int *id,* const wxString & *helpString*)

Sets the help string associated with a menu item.

Parameters

id	Menu item identifier.
helpString	Help string to associate with the menu item.

See also

GetHelpString()

void wxMenuBar::SetLabel (int *id,* const wxString & *label*)

Sets the label of a menu item.

Parameters

id	Menu item identifier.
label	Menu item label.

Remarks

Use only after the menubar has been associated with a frame.

See also

GetLabel()

void wxMenuBar::SetLabelTop (size_t *pos,* const wxString & *label*)

Sets the label of a top-level menu.

Parameters

pos	The position of a menu on the menu bar, starting from zero.
label	The menu label.

Remarks

Use only after the menubar has been associated with a frame.

Deprecated This function has been deprecated in favour of SetMenuLabel().

See also

GetLabelTop()

virtual void wxMenuBar::SetMenuLabel (size_t *pos,* const wxString & *label*) [virtual]

Sets the label of a top-level menu.

Parameters

pos	The position of a menu on the menu bar, starting from zero.
label	The menu label.

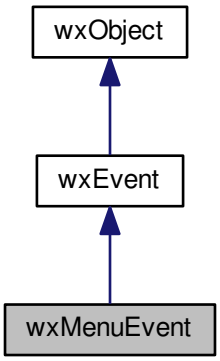
Remarks

Use only after the menubar has been associated with a frame.

21.464 wxMenuEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxMenuEvent:



21.464.1 Detailed Description

This class is used for a variety of menu-related events.

Note that these do not include menu command events, which are handled using wxCommandEvent objects.

Events of this class are generated by both menus that are part of a wxMenuBar, attached to wxFrame, and popup menus shown by wxWindow::PopupMenu(). They are sent to the following objects until one of them handles the event:

```
-# The menu object itself, as returned by GetMenu(), if any.
-# The wxMenuBar to which this menu is attached, if any.
-# The window associated with the menu, e.g. the one calling
PopupMenu() for the popup menus.
-# The top level parent of that window if it's different from the
window itself.
```

This is similar to command events generated by the menu items, but, unlike them, wxMenuEvent are only sent to the window itself and its top level parent but not any intermediate windows in the hierarchy.

The default handler for `wxEVT_MENU_HIGHLIGHT` in wxFrame displays help text in the status bar, see wxFrame::SetStatusBarPane().

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMenuEvent& event)

Event macros:

- EVT_MENU_OPEN(func): A menu is about to be opened. On Windows, this is only sent once for each navigation of the menubar (up until all menus have closed).
- EVT_MENU_CLOSE(func): A menu has been just closed. Notice that this event is currently being sent before the menu selection (`wxEVT_MENU`) event, if any.

- EVT_MENU_HIGHLIGHT(id, func): The menu item with the specified id has been highlighted: used to show help prompts in the status bar by wxFrame
- EVT_MENU_HIGHLIGHT_ALL(func): A menu item has been highlighted, i.e. the currently selected menu item has changed.

Library: wxCore

Category: Events

See also

wxCommandEvent, Events and Event Handling

Public Member Functions

- wxMenuEvent (wxEventType type=wxEVT_NULL, int id=0, wxMenu *menu=NULL)

  Constructor.
- wxMenu * GetMenu () const

  Returns the menu which is being opened or closed.
- int GetMenuId () const

  Returns the menu identifier associated with the event.
- bool IsPopup () const

  Returns true if the menu which is being opened or closed is a popup menu, false if it is a normal one.

Additional Inherited Members

21.464.2 Constructor & Destructor Documentation

wxMenuEvent::wxMenuEvent (wxEventType *type* = wxEVT_NULL, int *id* = 0, wxMenu * *menu* = NULL)

Constructor.

21.464.3 Member Function Documentation

wxMenu* wxMenuEvent::GetMenu ( ) const

Returns the menu which is being opened or closed.

This method can only be used with the OPEN and CLOSE events.

The returned value is never NULL in the ports implementing this function, which currently includes all the major ones.

int wxMenuEvent::GetMenuId ( ) const

Returns the menu identifier associated with the event.

This method should be only used with the HIGHLIGHT events.

bool wxMenuEvent::IsPopup ( ) const

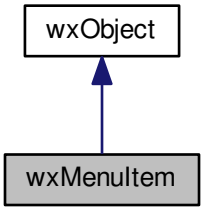
Returns true if the menu which is being opened or closed is a popup menu, false if it is a normal one.

This method should only be used with the `OPEN` and `CLOSE` events.

21.465 wxMenuItem Class Reference

`#include <wx/menuitem.h>`

Inheritance diagram for wxMenuItem:



21.465.1 Detailed Description

A menu item represents an item in a menu.

Note that you usually don't have to deal with it directly as wxMenu methods usually construct an object of this class for you.

Also please note that the methods related to fonts and bitmaps are currently only implemented for Windows, Mac and GTK+.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event) or void handlerFuncName(wxMenuEvent& event)

Event macros for events emitted by this class:

- EVT_MENU(id, func): Process a `wxEVT_MENU` command, which is generated by a menu item. This type of event is sent as wxCommandEvent.
- EVT_MENU_RANGE(id1, id2, func): Process a `wxEVT_MENU` command, which is generated by a range of menu items. This type of event is sent as wxCommandEvent.
- EVT_MENU_OPEN(func): A menu is about to be opened. On Windows, this is only sent once for each navigation of the menubar (up until all menus have closed). This type of event is sent as wxMenuEvent.
- EVT_MENU_CLOSE(func): A menu has been just closed. This type of event is sent as wxMenuEvent.
- EVT_MENU_HIGHLIGHT(id, func): The menu item with the specified id has been highlighted: used to show help prompts in the status bar by wxFrame This type of event is sent as wxMenuEvent.
- EVT_MENU_HIGHLIGHT_ALL(func): A menu item has been highlighted, i.e. the currently selected menu item has changed. This type of event is sent as wxMenuEvent.

Library: wxCore

Category: Menus

See also

wxMenuBar, wxMenu

Getters

- wxColour & GetBackgroundColour () const

  Returns the background colour associated with the menu item.
- virtual const wxBitmap & GetBitmap (bool checked=true) const

  Returns the checked or unchecked bitmap.
- virtual const wxBitmap & GetDisabledBitmap () const

  Returns the bitmap to be used for disabled items.
- wxFont & GetFont () const

  Returns the font associated with the menu item.
- const wxString & GetHelp () const

  Returns the help string associated with the menu item.
- int GetId () const

  Returns the menu item identifier.
- virtual wxString GetItemLabel () const

  Returns the text associated with the menu item including any accelerator characters that were passed to the constructor or SetItemLabel().
- virtual wxString GetItemLabelText () const

  Returns the text associated with the menu item, without any accelerator characters.
- wxItemKind GetKind () const

  Returns the item kind, one of `wxITEM_SEPARATOR`*,* `wxITEM_NORMAL`*,* `wxITEM_CHECK` *or* `wxITEM_RADIO`*.*
- wxString GetLabel () const

  Returns the text associated with the menu item without any accelerator characters it might contain.
- int GetMarginWidth () const

  Gets the width of the menu item checkmark bitmap.
- wxMenu * GetMenu () const

  Returns the menu this menu item is in, or NULL if this menu item is not attached.
- wxString GetName () const

  Returns the text associated with the menu item.
- wxMenu * GetSubMenu () const

  Returns the submenu associated with the menu item, or NULL if there isn't one.
- const wxString & GetText () const

  Returns the text associated with the menu item, such as it was passed to the wxMenuItem constructor, i.e. with any accelerator characters it may contain.
- wxColour & GetTextColour () const

  Returns the text colour associated with the menu item.
- virtual wxAcceleratorEntry * GetAccel () const

  Get our accelerator or NULL (caller must delete the pointer)
- static wxAcceleratorEntry * GetAccelFromString (const wxString &label)

  Extract the accelerator from the given menu string, return NULL if none found.

Public Member Functions

- wxMenuItem (wxMenu *parentMenu=NULL, int id=wxID_SEPARATOR, const wxString &text=wxEmptyString, const wxString &helpString=wxEmptyString, wxItemKind kind=wxITEM_NORMAL, wxMenu *subMenu=NULL)

  Constructs a wxMenuItem object.
- virtual ~wxMenuItem ()

  Destructor.
- virtual void Check (bool check=true)

  Checks or unchecks the menu item.
- virtual void Enable (bool enable=true)

  Enables or disables the menu item.

Checkers

- bool IsCheck () const

  Returns true if the item is a check item.
- bool IsCheckable () const

  Returns true if the item is checkable.
- virtual bool IsChecked () const

  Returns true if the item is checked.
- virtual bool IsEnabled () const

  Returns true if the item is enabled.
- bool IsRadio () const

  Returns true if the item is a radio button.
- bool IsSeparator () const

  Returns true if the item is a separator.
- bool IsSubMenu () const

  Returns true if the item is a submenu.

Setters

- void SetBackgroundColour (const wxColour &colour)

  Sets the background colour associated with the menu item.
- virtual void SetBitmap (const wxBitmap &bmp, bool checked=true)

  Sets the bitmap for the menu item.
- void SetBitmaps (const wxBitmap &checked, const wxBitmap &unchecked=wxNullBitmap)

  Sets the checked/unchecked bitmaps for the menu item.
- void SetDisabledBitmap (const wxBitmap &disabled)

  Sets the to be used for disabled menu items.
- void SetFont (const wxFont &font)

  Sets the font associated with the menu item.
- void SetHelp (const wxString &helpString)

  Sets the help string.
- virtual void SetItemLabel (const wxString &label)

  Sets the label associated with the menu item.
- void SetMarginWidth (int width)

  Sets the width of the menu item checkmark bitmap.
- void SetMenu (wxMenu *menu)

  Sets the parent menu which will contain this menu item.
- void SetSubMenu (wxMenu *menu)

  Sets the submenu of this menu item.
- virtual void SetText (const wxString &text)

  Sets the text associated with the menu item.
- void SetTextColour (const wxColour &colour)

  Sets the text colour associated with the menu item.
- virtual void SetAccel (wxAcceleratorEntry *accel)

  Set the accel for this item - this may also be done indirectly with SetText()

Static Public Member Functions

- static wxString GetLabelFromText (const wxString &text)
- static wxString GetLabelText (const wxString &text)

  Strips all accelerator characters and mnemonics from the given text.

Additional Inherited Members

21.465.2 Constructor & Destructor Documentation

wxMenuItem::wxMenuItem (wxMenu * *parentMenu* = NULL, int *id* = wxID_SEPARATOR, const wxString & *text* = wxEmptyString, const wxString & *helpString* = wxEmptyString, wxItemKind *kind* = wxITEM_NORMAL, wxMenu * *subMenu* = NULL)

Constructs a wxMenuItem object.

Menu items can be standard, or "stock menu items", or custom. For the standard menu items (such as commands to open a file, exit the program and so on, see Stock Items for the full list) it is enough to specify just the stock ID and leave *text* and *helpString* empty. Some platforms (currently wxGTK only, and see the remark in SetBitmap() documentation) will also show standard bitmaps for stock menu items.

Leaving at least *text* empty for the stock menu items is actually strongly recommended as they will have appearance and keyboard interface (including standard accelerators) familiar to the user.

For the custom (non-stock) menu items, *text* must be specified and while *helpString* may be left empty, it's recommended to pass the item description (which is automatically shown by the library in the status bar when the menu item is selected) in this parameter.

Finally note that you can e.g. use a stock menu label without using its stock help string:

```
// use all stock properties:
helpMenu->Append(wxID_ABOUT);

// use the stock label and the stock accelerator but not the stock help string:
helpMenu->Append(wxID_ABOUT, "", "My custom help string");

// use all stock properties except for the bitmap:
wxMenuItem *mymenu = new wxMenuItem(helpMenu, wxID_ABOUT);
mymenu->SetBitmap(wxArtProvider::GetBitmap(
      wxART_WARNING));
helpMenu->Append(mymenu);
```

that is, stock properties are set independently one from the other.

Parameters

parentMenu	Menu that the menu item belongs to. Can be NULL if the item is going to be added to the menu later.
id	Identifier for this menu item. May be wxID_SEPARATOR, in which case the given kind is ignored and taken to be wxITEM_SEPARATOR instead.
text	Text for the menu item, as shown on the menu. See SetItemLabel() for more info.
helpString	Optional help string that will be shown on the status bar.
kind	May be wxITEM_SEPARATOR, wxITEM_NORMAL, wxITEM_CHECK or wxITEM_RADIO.
subMenu	If non-NULL, indicates that the menu item is a submenu.

virtual wxMenuItem::~wxMenuItem ( ) [virtual]

Destructor.

21.465.3 Member Function Documentation

virtual void wxMenuItem::Check (bool *check* = `true`) [virtual]

Checks or unchecks the menu item.

Note that this only works when the item is already appended to a menu.

virtual void wxMenuItem::Enable (bool *enable* = `true`) [virtual]

Enables or disables the menu item.

virtual wxAcceleratorEntry* wxMenuItem::GetAccel ( ) const [virtual]

Get our accelerator or NULL (caller must delete the pointer)

static wxAcceleratorEntry* wxMenuItem::GetAccelFromString (const wxString & *label*) [static]

Extract the accelerator from the given menu string, return NULL if none found.

wxColour& wxMenuItem::GetBackgroundColour ( ) const

Returns the background colour associated with the menu item.

Availability: only available for the wxMSW port.

virtual const wxBitmap& wxMenuItem::GetBitmap (bool *checked* = `true`) const [virtual]

Returns the checked or unchecked bitmap.

Availability: only available for the wxMSW port.

virtual const wxBitmap& wxMenuItem::GetDisabledBitmap ( ) const [virtual]

Returns the bitmap to be used for disabled items.

Availability: only available for the wxMSW port.

wxFont& wxMenuItem::GetFont ( ) const

Returns the font associated with the menu item.

Availability: only available for the wxMSW port.

const wxString& wxMenuItem::GetHelp ( ) const

Returns the help string associated with the menu item.

int wxMenuItem::GetId ( ) const

Returns the menu item identifier.

virtual wxString wxMenuItem::GetItemLabel ( ) const `[virtual]`

Returns the text associated with the menu item including any accelerator characters that were passed to the constructor or SetItemLabel().

See also

GetItemLabelText(), GetLabelText()

virtual wxString wxMenuItem::GetItemLabelText ( ) const `[virtual]`

Returns the text associated with the menu item, without any accelerator characters.

See also

GetItemLabel(), GetLabelText()

wxItemKind wxMenuItem::GetKind ( ) const

Returns the item kind, one of `wxITEM_SEPARATOR`, `wxITEM_NORMAL`, `wxITEM_CHECK` or `wxITEM_RADIO`.

wxString wxMenuItem::GetLabel ( ) const

Returns the text associated with the menu item without any accelerator characters it might contain.

Deprecated This function is deprecated in favour of GetItemLabelText().

See also

GetItemLabelText()

static wxString wxMenuItem::GetLabelFromText (const wxString & *text*) `[static]`

Deprecated This function is deprecated; please use GetLabelText() instead.

See also

GetLabelText()

static wxString wxMenuItem::GetLabelText (const wxString & *text*) `[static]`

Strips all accelerator characters and mnemonics from the given *text*.

For example:

```
wxMenuItem::GetLabelfromText("&Hello\tCtrl-h");
```

will return just `"Hello"`.

See also

GetItemLabelText(), GetItemLabel()

int wxMenuItem::GetMarginWidth ( ) const

Gets the width of the menu item checkmark bitmap.

Availability: only available for the wxMSW port.

wxMenu* wxMenuItem::GetMenu ( ) const

Returns the menu this menu item is in, or NULL if this menu item is not attached.

wxString wxMenuItem::GetName ( ) const

Returns the text associated with the menu item.

Deprecated This function is deprecated. Please use GetItemLabel() or GetItemLabelText() instead.

See also

GetItemLabel(), GetItemLabelText()

wxMenu* wxMenuItem::GetSubMenu ( ) const

Returns the submenu associated with the menu item, or NULL if there isn't one.

const wxString& wxMenuItem::GetText ( ) const

Returns the text associated with the menu item, such as it was passed to the wxMenuItem constructor, i.e. with any accelerator characters it may contain.

Deprecated This function is deprecated in favour of GetItemLabel().

See also

GetLabelFromText()

wxColour& wxMenuItem::GetTextColour ( ) const

Returns the text colour associated with the menu item.

Availability: only available for the wxMSW port.

bool wxMenuItem::IsCheck ( ) const

Returns true if the item is a check item.

Unlike IsCheckable() this doesn't return true for the radio buttons.

Since

2.9.5

bool wxMenuItem::IsCheckable ( ) const

Returns true if the item is checkable.

Notice that the radio buttons are considered to be checkable as well, so this method returns true for them too. Use IsCheck() if you want to test for the check items only.

virtual bool wxMenuItem::IsChecked ( ) const `[virtual]`

Returns true if the item is checked.

virtual bool wxMenuItem::IsEnabled ( ) const `[virtual]`

Returns true if the item is enabled.

bool wxMenuItem::IsRadio ( ) const

Returns true if the item is a radio button.

Since

2.9.5

bool wxMenuItem::IsSeparator ( ) const

Returns true if the item is a separator.

bool wxMenuItem::IsSubMenu ( ) const

Returns true if the item is a submenu.

virtual void wxMenuItem::SetAccel (wxAcceleratorEntry * *accel*) `[virtual]`

Set the accel for this item - this may also be done indirectly with SetText()

void wxMenuItem::SetBackgroundColour (const wxColour & *colour*)

Sets the background colour associated with the menu item.

Availability: only available for the wxMSW port.

virtual void wxMenuItem::SetBitmap (const wxBitmap & *bmp*, bool *checked* = `true`) `[virtual]`

Sets the bitmap for the menu item.

It is equivalent to wxMenuItem::SetBitmaps(bmp, wxNullBitmap) if *checked* is true (default value) or SetBitmaps(wxNullBitmap, bmp) otherwise.

SetBitmap() must be called before the item is appended to the menu, i.e. appending the item without a bitmap and setting one later is not guaranteed to work. But the bitmap can be changed or reset later if it had been set up initially.

Notice that GTK+ uses a global setting called `gtk-menu-images` to determine if the images should be shown in the menus at all. If it is off (which is the case in e.g. Gnome 2.28 by default), no images will be shown, consistently with the native behaviour.

Availability: only available for the wxMSW, wxOSX, wxGTK ports.

void wxMenuItem::SetBitmaps (const wxBitmap & *checked,* const wxBitmap & *unchecked* = wxNullBitmap)

Sets the checked/unchecked bitmaps for the menu item.

The first bitmap is also used as the single bitmap for uncheckable menu items.

Availability: only available for the wxMSW port.

void wxMenuItem::SetDisabledBitmap (const wxBitmap & *disabled*)

Sets the to be used for disabled menu items.

Availability: only available for the wxMSW port.

void wxMenuItem::SetFont (const wxFont & *font*)

Sets the font associated with the menu item.

Availability: only available for the wxMSW port.

void wxMenuItem::SetHelp (const wxString & *helpString*)

Sets the help string.

virtual void wxMenuItem::SetItemLabel (const wxString & *label*) `[virtual]`

Sets the label associated with the menu item.

Note that if the ID of this menu item corresponds to a stock ID, then it is not necessary to specify a label: wxWidgets will automatically use the stock item label associated with that ID. See the constructor for more info.

The label string for the normal menu items (not separators) may include the accelerator which can be used to activate the menu item from keyboard. An accelerator key can be specified using the ampersand `&` character. In order to embed an ampersand character in the menu item text, the ampersand must be doubled.

Optionally you can specify also an accelerator string appending a tab character `\t` followed by a valid key combination (e.g. `CTRL+V`). Its general syntax is any combination of `"CTRL"`, `"RAWCTRL"`, `"ALT"` and `"SHIFT"` strings (case doesn't matter) separated by either `'-'` or `'+'` characters and followed by the accelerator itself. Notice that `CTRL` corresponds to the "Ctrl" key on most platforms but not under Mac OS where it is mapped to "Cmd" key on Mac keyboard. Usually this is exactly what you want in portable code but if you really need to use the (rarely used for this purpose) "Ctrl" key even under Mac, you may use `RAWCTRL` to prevent this mapping. Under the other platforms `RAWCTRL` is the same as plain `CTRL`.

The accelerator may be any alphanumeric character, any function key (from F1 to F12) or one of the special characters listed in the table below (again, case doesn't matter):

- `DEL` or `DELETE`: Delete key
- `BACK` : Backspace key
- `INS` or `INSERT`: Insert key
- `ENTER` or `RETURN`: Enter key
- `PGUP`: PageUp key
- `PGDN`: PageDown key
- `LEFT`: Left cursor arrow key
- `RIGHT`: Right cursor arrow key

- `UP`: Up cursor arrow key
- `DOWN`: Down cursor arrow key
- `HOME`: Home key
- `END`: End key
- `SPACE`: Space
- `TAB`: Tab key
- `ESC` or `ESCAPE`: Escape key (Windows only)

Examples:

```
m_pMyMenuItem->SetItemLabel("My &item\tCTRL+I");
m_pMyMenuItem2->SetItemLabel("Clean && build\tF7");
m_pMyMenuItem3->SetItemLabel("Simple item");
m_pMyMenuItem4->SetItemLabel("Item with &accelerator");
```

Note

In wxGTK using `"SHIFT"` with non-alphabetic characters currently doesn't work, even in combination with other modifiers, due to GTK+ limitation. E.g. `Shift+Ctrl+A` works but `Shift+Ctrl+1` or `Shift+/` do not, so avoid using accelerators of this form in portable code.

See also

GetItemLabel(), GetItemLabelText()

void wxMenuItem::SetMarginWidth (int *width*)

Sets the width of the menu item checkmark bitmap.

Availability: only available for the wxMSW port.

void wxMenuItem::SetMenu (wxMenu * *menu*)

Sets the parent menu which will contain this menu item.

void wxMenuItem::SetSubMenu (wxMenu * *menu*)

Sets the submenu of this menu item.

virtual void wxMenuItem::SetText (const wxString & *text*) `[virtual]`

Sets the text associated with the menu item.

Deprecated This function is deprecated in favour of SetItemLabel().

See also

SetItemLabel().

void wxMenuItem::SetTextColour (const wxColour & *colour*)

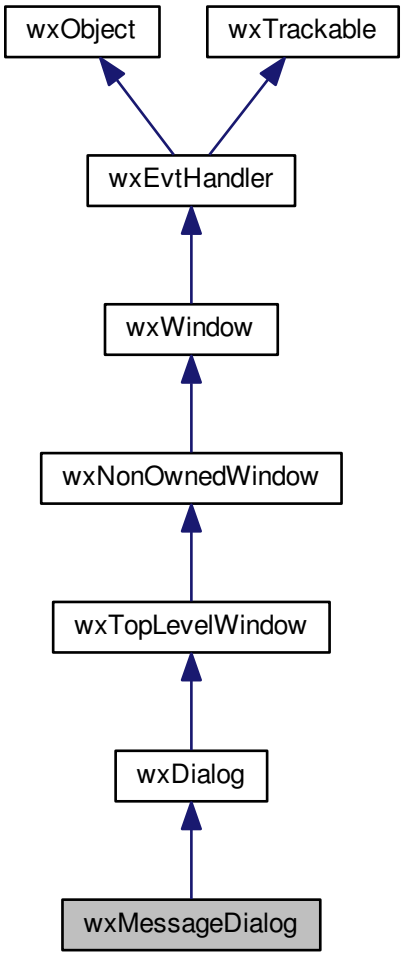
Sets the text colour associated with the menu item.

Availability: only available for the wxMSW port.

21.466 wxMessageDialog Class Reference

`#include <wx/msgdlg.h>`

Inheritance diagram for wxMessageDialog:



21.466.1 Detailed Description

This class represents a dialog that shows a single or multi-line message, with a choice of OK, Yes, No and Cancel buttons.

Styles

This class supports the following styles:

- wxOK: Puts an Ok button in the message box. May be combined with `wxCANCEL`.
- wxCANCEL: Puts a Cancel button in the message box. Must be combined with either `wxOK` or `wxYES_NO`.
- wxYES_NO: Puts Yes and No buttons in the message box. It is recommended to always use `wxCANCEL` with this style as otherwise the message box won't have a close button under wxMSW and the user will be forced to answer it.
- wxHELP: Puts a Help button to the message box. This button can have special appearance or be specially positioned if its label is not changed from the default one. Notice that using this button is not supported when showing a message box from non-main thread in wxOSX/Cocoa and it is not supported in wxOSX/Carbon at all. Available since wxWidgets 2.9.3.
- wxNO_DEFAULT: Makes the "No" button default, can only be used with `wxYES_NO`.
- wxCANCEL_DEFAULT: Makes the "Cancel" button default, can only be used with `wxCANCEL`. This style is currently not supported (and ignored) in wxOSX.
- wxYES_DEFAULT: Makes the "Yes" button default, this is the default behaviour and this flag exists solely for symmetry with `wxNO_DEFAULT`.
- wxOK_DEFAULT: Makes the "OK" button default, this is the default behaviour and this flag exists solely for symmetry with `wxCANCEL_DEFAULT`.
- wxICON_NONE: Displays no icon in the dialog if possible (an icon might still be displayed if the current platform mandates its use). This style may be used to prevent the dialog from using the default icon based on `wxYES_NO` presence as explained in `wxICON_QUESTION` and `wxICON_INFORMATION` documentation below.
- wxICON_EXCLAMATION: Displays an exclamation, or warning, icon in the dialog.
- wxICON_ERROR: Displays an error icon in the dialog.
- wxICON_HAND: Displays an error symbol, this is a MSW-inspired synonym for `wxICON_ERROR`.
- wxICON_QUESTION: Displays a question mark symbol. This icon is automatically used with `wxYES_NO` so it's usually unnecessary to specify it explicitly. This style is not supported for message dialogs under wxMSW when a task dialog is used to implement them (i.e. when running under Windows Vista or later) because `Microsoft guidelines` indicate that no icon should be used for routine confirmations. If it is specified, no icon will be displayed.
- wxICON_INFORMATION: Displays an information symbol. This icon is used by default if `wxYES_NO` is not given so it is usually unnecessary to specify it explicitly.
- wxICON_AUTH_NEEDED: Displays an authentication needed symbol. This style is only supported for message dialogs under wxMSW when a task dialog is used to implement them (i.e. when running under Windows Vista or later). In other cases the default icon selection logic will be used. Note this can be combined with other styles to provide a fallback. For instance, using wxICON_AUTH_NEEDED | wxICON_QUESTION will show a shield symbol on Windows Vista or above and a question symbol on other platforms. Available since wxWidgets 2.9.5
- wxSTAY_ON_TOP: Makes the message box stay on top of all other windows and not only just its parent (currently implemented only under MSW and GTK).
- wxCENTRE: Centre the message box on its parent or on the screen if parent is not specified. Setting this style under MSW makes no differences as the dialog is always centered on the parent.

Library: wxCore

Category: Common Dialogs

See also

wxMessageDialog Overview
wxRichMessageDialog

Classes

- class ButtonLabel

  Helper class allowing to use either stock id or string labels.

Public Member Functions

- wxMessageDialog (wxWindow *parent, const wxString &message, const wxString &caption=wxMessageBoxCaptionStr, long style=wxOK|wxCENTRE, const wxPoint &pos=wxDefaultPosition)

  Constructor specifying the message box properties.
- virtual void SetExtendedMessage (const wxString &extendedMessage)

  Sets the extended message for the dialog: this message is usually an extension of the short message specified in the constructor or set with SetMessage().
- virtual bool SetHelpLabel (const ButtonLabel &help)

  Sets the label for the Help button.
- virtual void SetMessage (const wxString &message)

  Sets the message shown by the dialog.
- virtual bool SetOKCancelLabels (const ButtonLabel &ok, const ButtonLabel &cancel)

  Overrides the default labels of the OK and Cancel buttons.
- virtual bool SetOKLabel (const ButtonLabel &ok)

  Overrides the default label of the OK button.
- virtual bool SetYesNoCancelLabels (const ButtonLabel &yes, const ButtonLabel &no, const ButtonLabel &cancel)

  Overrides the default labels of the Yes, No and Cancel buttons.
- virtual bool SetYesNoLabels (const ButtonLabel &yes, const ButtonLabel &no)

  Overrides the default labels of the Yes and No buttons.
- virtual int ShowModal ()

  Shows the dialog, returning one of wxID_OK, wxID_CANCEL, wxID_YES, wxID_NO or wxID_HELP.
- wxString GetCaption () const
- wxString GetMessage () const
- wxString GetExtendedMessage () const
- long GetMessageDialogStyle () const
- bool HasCustomLabels () const
- wxString GetYesLabel () const
- wxString GetNoLabel () const
- wxString GetOKLabel () const
- wxString GetCancelLabel () const
- wxString GetHelpLabel () const
- long GetEffectiveIcon () const

Additional Inherited Members

21.466.2 Constructor & Destructor Documentation

wxMessageDialog::wxMessageDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption* = wxMessageBoxCaptionStr, long *style* = wxOK|wxCENTRE, const wxPoint & *pos* = wxDefaultPosition)

Constructor specifying the message box properties.

Use ShowModal() to show the dialog.

style may be a bit list of the identifiers described above.

Notice that not all styles are compatible: only one of `wxOK` and `wxYES_NO` may be specified (and one of them must be specified) and at most one default button style can be used and it is only valid if the corresponding button is shown in the message box.

Parameters

parent	Parent window.
message	Message to show in the dialog.
caption	The dialog title.
style	Combination of style flags described above.
pos	Dialog position (ignored under MSW).

21.466.3 Member Function Documentation

wxString wxMessageDialog::GetCancelLabel ( ) const

wxString wxMessageDialog::GetCaption ( ) const

long wxMessageDialog::GetEffectiveIcon ( ) const

wxString wxMessageDialog::GetExtendedMessage ( ) const

wxString wxMessageDialog::GetHelpLabel ( ) const

wxString wxMessageDialog::GetMessage ( ) const

long wxMessageDialog::GetMessageDialogStyle ( ) const

wxString wxMessageDialog::GetNoLabel ( ) const

wxString wxMessageDialog::GetOKLabel ( ) const

wxString wxMessageDialog::GetYesLabel ( ) const

bool wxMessageDialog::HasCustomLabels ( ) const

virtual void wxMessageDialog::SetExtendedMessage (const wxString & *extendedMessage*) `[virtual]`

Sets the extended message for the dialog: this message is usually an extension of the short message specified in the constructor or set with SetMessage().

If it is set, the main message appears highlighted – if supported – and this message appears beneath it in normal font. On the platforms which don't support extended messages, it is simply appended to the normal message with an empty line separating them.

Since

2.9.0

virtual bool wxMessageDialog::SetHelpLabel (const ButtonLabel & *help*) [virtual]

Sets the label for the Help button.

Please see the remarks in SetYesNoLabels() documentation.

Notice that changing the label of the help button resets its special status (if any, this depends on the platform) and it will be treated just like another button in this case.

Since

2.9.3

virtual void wxMessageDialog::SetMessage (const wxString & *message*) [virtual]

Sets the message shown by the dialog.

Since

2.9.0

virtual bool wxMessageDialog::SetOKCancelLabels (const ButtonLabel & *ok,* const ButtonLabel & *cancel*) [virtual]

Overrides the default labels of the OK and Cancel buttons.

Please see the remarks in SetYesNoLabels() documentation.

Since

2.9.0

virtual bool wxMessageDialog::SetOKLabel (const ButtonLabel & *ok*) [virtual]

Overrides the default label of the OK button.

Please see the remarks in SetYesNoLabels() documentation.

Since

2.9.0

virtual bool wxMessageDialog::SetYesNoCancelLabels (const ButtonLabel & *yes,* const ButtonLabel & *no,* const ButtonLabel & *cancel*) [virtual]

Overrides the default labels of the Yes, No and Cancel buttons.

Please see the remarks in SetYesNoLabels() documentation.

Since

2.9.0

virtual bool wxMessageDialog::SetYesNoLabels (const ButtonLabel & *yes*, const ButtonLabel & *no*) [virtual]

Overrides the default labels of the Yes and No buttons.

The arguments of this function can be either strings or one of the standard identifiers, such as wxID_APPLY or wxID_OPEN. Notice that even if the label is specified as an identifier, the return value of the dialog ShowModal() method still remains one of wxID_OK, wxID_CANCEL, wxID_YES or wxID_NO values, i.e. this identifier changes only the label appearance but not the return code generated by the button. It is possible to mix stock identifiers and string labels in the same function call, for example:

```
wxMessageDialog dlg(...);
dlg.SetYesNoLabels(wxID_SAVE, _("&Don't save"));
```

Also notice that this function is not currently available on all platforms (although as of wxWidgets 2.9.0 it is implemented in all major ports), so it may return false to indicate that the labels couldn't be changed. If it returns true, the labels were set successfully.

Typically, if the function was used successfully, the main dialog message may need to be changed, e.g.:

```
wxMessageDialog dlg(...);
if ( dlg.SetYesNoLabels(_("&Quit"), _("&Don't quit")) )
    dlg.SetMessage(_("What do you want to do?"));
else // buttons have standard "Yes"/"No" values, so rephrase the question
    dlg.SetMessage(_("Do you really want to quit?"));
```

Since

2.9.0

virtual int wxMessageDialog::ShowModal ( ) [virtual]

Shows the dialog, returning one of wxID_OK, wxID_CANCEL, wxID_YES, wxID_NO or wxID_HELP.

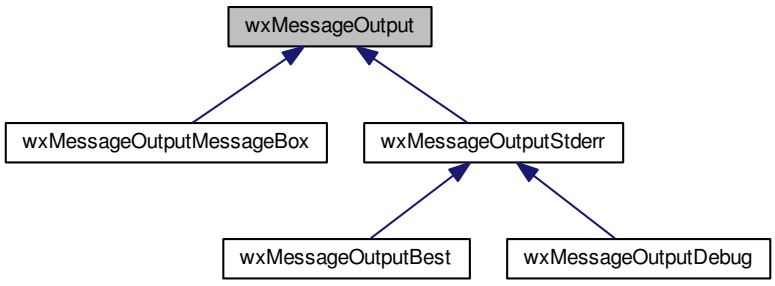
Notice that this method returns the identifier of the button which was clicked unlike wxMessageBox() function.

Reimplemented from wxDialog.

21.467 wxMessageOutput Class Reference

`#include <wx/msgout.h>`

Inheritance diagram for wxMessageOutput:



21.467.1 Detailed Description

Simple class allowing to write strings to various output channels.

wxMessageOutput is a low-level class and doesn't provide any of the conveniences of wxLog. It simply allows to write a message to some output channel: usually file or standard error but possibly also a message box. While use of wxLog and related functions is preferable in many cases sometimes this simple interface may be more convenient.

This class itself is an abstract base class for various concrete derived classes:

- wxMessageOutputStderr
- wxMessageOutputBest
- wxMessageOutputMessageBox
- wxMessageOutputLog

It also provides access to the global message output object which is created by wxAppTraits::CreateMessageOutput() which creates an object of class wxMessageOutputStderr in console applications and wxMessageOutputBest in the GUI ones but may be overridden in user-defined traits class.

Example of using this class:

```
wxMessageOutputDebug().Printf("name=%s, preparing to greet...", name);
wxMessageOutput::Get()->Printf("Hello, %s!", name);
```

Library: wxBase

Category: Logging

Public Member Functions

- void Printf (const wxString &format,...)

  Output a message.
- virtual void Output (const wxString &str)=0

  Method called by Printf() to really output the text.

Static Public Member Functions

- static wxMessageOutput * Get ()

  Return the global message output object.
- static wxMessageOutput * Set (wxMessageOutput *msgout)

  Sets the global message output object.

21.467.2 Member Function Documentation

static wxMessageOutput* wxMessageOutput::Get ( ) `[static]`

Return the global message output object.

This object is never NULL while the program is running but may be NULL during initialization (before wxApp object is instantiated) or shutdown.(after wxApp destruction).

See also

wxAppTraits::CreateMessageOutput()

virtual void wxMessageOutput::Output (const wxString & *str*) `[pure virtual]`

Method called by Printf() to really output the text.

This method is overridden in various derived classes and is also the one you should override if you implement a custom message output object.

It may also be called directly instead of Printf(). This is especially useful when outputting a user-defined string because it can be simply called with this string instead of using

```
msgout.Printf("%s", str);
```

(notice that passing user-defined string to Printf() directly is, of course, a security risk).

void wxMessageOutput::Printf (const wxString & *format, ...*)

Output a message.

This function uses the same conventions as standard `printf()`.

static wxMessageOutput* wxMessageOutput::Set (wxMessageOutput * *msgout*) `[static]`

Sets the global message output object.

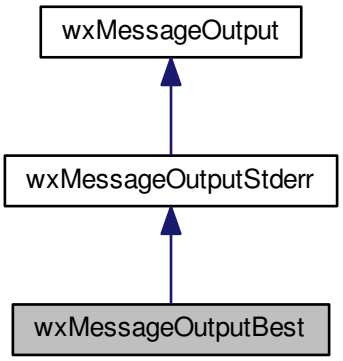
Using this function may be a simpler alternative to changing the message output object used for your program than overriding wxAppTraits::CreateMessageOutput().

Remember to delete the returned pointer or restore it later with another call to Set().

21.468 wxMessageOutputBest Class Reference

`#include <wx/msgout.h>`

Inheritance diagram for wxMessageOutputBest:



21.468.1 Detailed Description

Output messages in the best possible way.

Some systems (e.g. MSW) are capable of showing message boxes even from console programs. If this is the case, this class will use message box if standard error stream is not available (e.g. running console program not from console under Windows) or possibly even always, depending on the value of flags constructor argument.

Library: wxBase

Category: Logging

Public Member Functions

- wxMessageOutputBest (wxMessageOutputFlags flags=wxMSGOUT_PREFER_STDERR)

  Create a new message output object.

Additional Inherited Members

21.468.2 Constructor & Destructor Documentation

wxMessageOutputBest::wxMessageOutputBest (wxMessageOutputFlags *flags* = wxMSGOUT_PREFER_STDERR)

Create a new message output object.

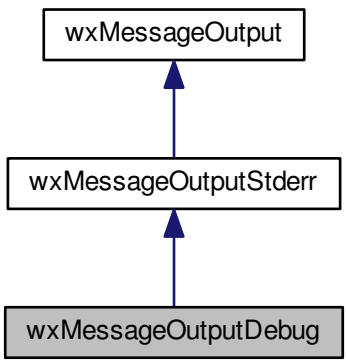
Parameters

flags	May be either `wxMSGOUT_PREFER_STDERR` (default) meaning that standard error will be used if it's available (e.g. program is being run from console under Windows) or `wxMSGOUT_PREFER_MSGBOX` meaning that a message box will always be used if the current system supports showing message boxes from console programs (currently only Windows does).

21.469 wxMessageOutputDebug Class Reference

`#include <wx/msgout.h>`

Inheritance diagram for wxMessageOutputDebug:



21.469.1 Detailed Description

Output messages to the system debug output channel.

Under MSW this class outputs messages to the so called debug output. Under the other systems it simply uses the standard error stream.

Library: wxBase

Category: Logging

Public Member Functions

- wxMessageOutputDebug ()

  Default constructor.

Additional Inherited Members

21.469.2 Constructor & Destructor Documentation

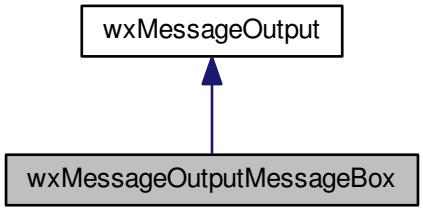
wxMessageOutputDebug::wxMessageOutputDebug ( )

Default constructor.

21.470 wxMessageOutputMessageBox Class Reference

```
#include <wx/msgout.h>
```

Inheritance diagram for wxMessageOutputMessageBox:



21.470.1 Detailed Description

Output messages by showing them in a message box.

This class is only available to GUI applications, unlike all the other wxMessageOutput-derived classes.

Library: wxCore

Category: Logging

Public Member Functions

- wxMessageOutputMessageBox ()

  Default constructor.

Additional Inherited Members

21.470.2 Constructor & Destructor Documentation

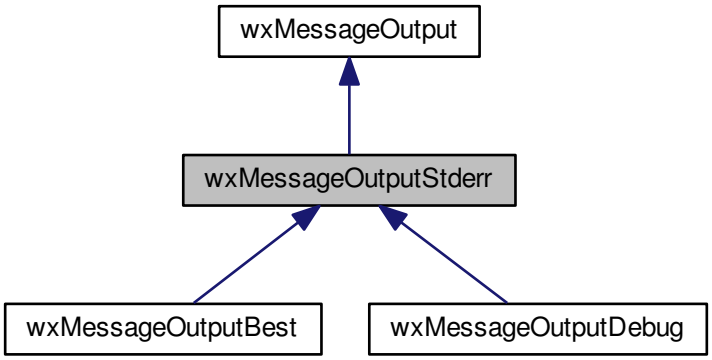
wxMessageOutputMessageBox::wxMessageOutputMessageBox ( )

Default constructor.

21.471 wxMessageOutputStderr Class Reference

`#include <wx/msgout.h>`

Inheritance diagram for wxMessageOutputStderr:



21.471.1 Detailed Description

Output messages to stderr or another STDIO file stream.

Implements wxMessageOutput by using stderr or specified file.

Library: wxBase

Category: Logging

Public Member Functions

- wxMessageOutputStderr (FILE *fp=stderr)

  Create a new message output object associated with standard error stream by default.

Additional Inherited Members

21.471.2 Constructor & Destructor Documentation

wxMessageOutputStderr::wxMessageOutputStderr (FILE * *fp* = `stderr`)

Create a new message output object associated with standard error stream by default.

Parameters

fp	Non-null STDIO file stream. Notice that this object does *not* take ownership of this pointer, i.e. the caller is responsible for both ensuring that its life-time is great er than life-time of this object and for deleting it if necessary.

21.472 wxMessageQueue< T > Class Template Reference

`#include <wx/msgqueue.h>`

21.472.1 Detailed Description

template<typename T>class wxMessageQueue< T >

wxMessageQueue allows passing messages between threads.

This class should be typically used to communicate between the main and worker threads. The main thread calls wxMessageQueue::Post and the worker thread calls wxMessageQueue::Receive.

Template Parameters

T	For this class a message is an object of arbitrary type T.

Notice that often there is a some special message indicating that the thread should terminate as there is no other way to gracefully shutdown a thread waiting on the message queue.

Since

2.9.0

Library: None; this class implementation is entirely header-based.

Category: Threading

See also

wxThread

Public Member Functions

- wxMessageQueue ()

  Default and only constructor.
- wxMessageQueueError Clear ()

  Remove all messages from the queue.
- bool IsOk () const

  Returns true if the object had been initialized successfully, false if an error occurred.
- wxMessageQueueError Post (T const &msg)

  Add a message to this queue and signal the threads waiting for messages (i.e.
- wxMessageQueueError Receive (T &msg)

  Block until a message becomes available in the queue.
- wxMessageQueueError ReceiveTimeout (long timeout, T &msg)

  Block until a message becomes available in the queue, but no more than timeout milliseconds has elapsed.

21.472.2 Constructor & Destructor Documentation

template<typename T > wxMessageQueue< T >::wxMessageQueue ( )

Default and only constructor.

Use wxMessageQueue::IsOk to check if the object was successfully initialized.

21.472.3 Member Function Documentation

template<typename T > wxMessageQueueError wxMessageQueue< T >::Clear ( )

Remove all messages from the queue.

This method is meant to be called from the same thread(s) that call Post() to discard any still pending requests if they became unnecessary.

Since

2.9.1

template<typename T > bool wxMessageQueue< T >::IsOk ( ) const

Returns true if the object had been initialized successfully, false if an error occurred.

template<typename T > wxMessageQueueError wxMessageQueue< T >::Post (T const & *msg*)

Add a message to this queue and signal the threads waiting for messages (i.e.

the threads which called wxMessageQueue::Receive or wxMessageQueue::ReceiveTimeout).

This method is safe to call from multiple threads in parallel.

template<typename T > wxMessageQueueError wxMessageQueue< T >::Receive (T & *msg*)

Block until a message becomes available in the queue.

Waits indefinitely long or until an error occurs.

The message is returned in *msg*.

template<typename T > wxMessageQueueError wxMessageQueue< T >::ReceiveTimeout (long *timeout*, T & *msg*)

Block until a message becomes available in the queue, but no more than *timeout* milliseconds has elapsed.

If no message is available after *timeout* milliseconds then returns **wxMSGQUEUE_TIMEOUT**.

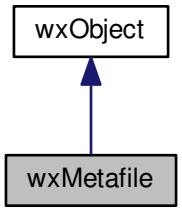
If *timeout* is 0 then checks for any messages present in the queue and returns immediately without waiting.

The message is returned in *msg*.

21.473 wxMetafile Class Reference

`#include <wx/metafile.h>`

Inheritance diagram for wxMetafile:



21.473.1 Detailed Description

A **wxMetafile** represents the MS Windows metafile object, so metafile operations have no effect in X.

In wxWidgets, only sufficient functionality has been provided for copying a graphic to the clipboard; this may be extended in a future version.

Presently, the only way of creating a metafile is to use a wxMetafileDC.

Availability: only available for the wxMSW port.

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxMetafileDC

Public Member Functions

- wxMetafile (const wxString &filename=wxEmptyString)

  Constructor.
- ~wxMetafile ()

  Destructor.

- bool IsOk ()

  Returns true if the metafile is valid.
- bool Play (wxDC *dc)

  Plays the metafile into the given device context, returning true if successful.
- bool SetClipboard (int width=0, int height=0)

  Passes the metafile data to the clipboard.

Additional Inherited Members

21.473.2 Constructor & Destructor Documentation

wxMetafile::wxMetafile (const wxString & *filename* = wxEmptyString)

Constructor.

If a filename is given, the Windows disk metafile is read in. Check whether this was performed successfully by using the IsOk() member.

wxMetafile::~wxMetafile ( )

Destructor.

See Object Destruction for more info.

21.473.3 Member Function Documentation

bool wxMetafile::IsOk ( )

Returns true if the metafile is valid.

bool wxMetafile::Play (wxDC * *dc*)

Plays the metafile into the given device context, returning true if successful.

bool wxMetafile::SetClipboard (int *width* = 0, int *height* = 0)

Passes the metafile data to the clipboard.

The metafile can no longer be used for anything, but the wxMetafile object must still be destroyed by the application.

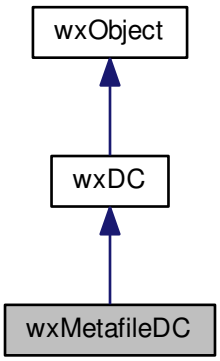
Below is a example of metafile, metafile device context and clipboard use from the `hello.cpp` example. Note the way the metafile dimensions are passed to the clipboard, making use of the device context's ability to keep track of the maximum extent of drawing commands.

```
wxMetafileDC dc;
if (dc.IsOk())
{
    Draw(dc, false);
    wxMetafile *mf = dc.Close();
    if (mf)
    {
        bool success = mf->SetClipboard((int)(dc.MaxX() + 10), (int)(dc.
      MaxY() + 10));
        delete mf;
    }
}
```

21.474 wxMetafileDC Class Reference

`#include <wx/metafile.h>`

Inheritance diagram for wxMetafileDC:



21.474.1 Detailed Description

This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal **wxDC**.

The wxMetafileDC::Close member must be called after drawing into the device context, to return a metafile. The only purpose for this at present is to allow the metafile to be copied to the clipboard (see wxMetafile).

Adding metafile capability to an application should be easy if you already write to a wxDC; simply pass the wxMetafileDC to your drawing function instead. You may wish to conditionally compile this code so it is not compiled under X (although no harm will result if you leave it in).

Note that a metafile saved to disk is in standard Windows metafile format, and cannot be imported into most applications. To make it importable, call the function wxMakeMetafilePlaceable after closing your disk-based metafile device context.

Library: wxCore

Category: Device Contexts

See also

wxMetafile, wxDC

Public Member Functions

- wxMetafileDC (const wxString &filename=wxEmptyString)

  Constructor.
- ~wxMetafileDC ()

  Destructor.

- wxMetafile * Close ()

  This must be called after the device context is finished with.

Additional Inherited Members

21.474.2 Constructor & Destructor Documentation

wxMetafileDC::wxMetafileDC (const wxString & *filename* = wxEmptyString)

Constructor.

If no filename is passed, the metafile is created in memory.

wxMetafileDC::~wxMetafileDC ( )

Destructor.

21.474.3 Member Function Documentation

wxMetafile* wxMetafileDC::Close ( )

This must be called after the device context is finished with.

A metafile is returned, and ownership of it passes to the calling application (so it should be destroyed explicitly).

21.475 wxMimeTypesManager Class Reference

```
#include <wx/mimetype.h>
```

21.475.1 Detailed Description

This class allows the application to retrieve information about all known MIME types from a system-specific location and the filename extensions to the MIME types and vice versa.

MIME stands for "Multipurpose Internet Mail Extensions" and was originally used in mail protocols. It's standardized by several RFCs.

Under Windows, the MIME type information is queried from registry. Under Linux and Unix, it is queried from the XDG data directories.

Currently, wxMimeTypesManager is limited to *reading* MIME type information.

The application should not construct its own manager: it should use the object pointer wxTheMimeTypesManager. The functions GetFileTypeFromMimeType() and GetFileTypeFromExtension() return a wxFileType object which may be further queried for file description, icon and other attributes.

21.475.2 Helper functions

All of these functions are static (i.e. don't need a wxMimeTypesManager object to call them) and provide some useful operations for string representations of MIME types. Their usage is recommended instead of directly working with MIME types using wxString functions.

- wxMimeTypesManager::IsOfType()

21.475.3 Query database

These functions are the heart of this class: they allow to find a file type object from either file extension or MIME type. If the function is successful, it returns a pointer to the wxFileType object which must be deleted by the caller, otherwise NULL will be returned.

- wxMimeTypesManager::GetFileTypeFromMimeType()
- wxMimeTypesManager::GetFileTypeFromExtension()

Library: wxBase

Category: Application and System configuration

See also

wxFileType

Public Member Functions

- wxMimeTypesManager ()

  Constructor puts the object in the "working" state.
- ~wxMimeTypesManager ()

  Destructor is not virtual, so this class should not be derived from.
- void AddFallbacks (const wxFileTypeInfo *fallbacks)

  This function may be used to provide hard-wired fallbacks for the MIME types and extensions that might not be present in the system MIME database.
- wxFileType * GetFileTypeFromExtension (const wxString &extension)

  Gather information about the files with given extension and return the corresponding wxFileType object or NULL if the extension is unknown.
- wxFileType * GetFileTypeFromMimeType (const wxString &mimeType)

  Gather information about the files with given MIME type and return the corresponding wxFileType object or NULL if the MIME type is unknown.
- wxFileType * Associate (const wxFileTypeInfo &ftInfo)

  Create a new association using the fields of wxFileTypeInfo (at least the MIME type and the extension should be set).
- bool Unassociate (wxFileType *ft)

  Undo Associate().
- size_t EnumAllFileTypes (wxArrayString &mimetypes)

  Enumerate all known file types.

Static Public Member Functions

- static bool IsOfType (const wxString &mimeType, const wxString &wildcard)

  This function returns true if either the given mimeType is exactly the same as wildcard or if it has the same category and the subtype of wildcard is ''.*

21.475.4 Constructor & Destructor Documentation

wxMimeTypesManager::wxMimeTypesManager ( )

Constructor puts the object in the "working" state.

wxMimeTypesManager::~wxMimeTypesManager ( )

Destructor is not virtual, so this class should not be derived from.

21.475.5 Member Function Documentation

void wxMimeTypesManager::AddFallbacks (const wxFileTypeInfo * *fallbacks*)

This function may be used to provide hard-wired fallbacks for the MIME types and extensions that might not be present in the system MIME database.

Please see the typetest sample for an example of using it.

wxFileType* wxMimeTypesManager::Associate (const wxFileTypeInfo & *ftInfo*)

Create a new association using the fields of wxFileTypeInfo (at least the MIME type and the extension should be set).

size_t wxMimeTypesManager::EnumAllFileTypes (wxArrayString & *mimetypes*)

Enumerate all known file types.

Returns the number of retrieved items.

wxFileType* wxMimeTypesManager::GetFileTypeFromExtension (const wxString & *extension*)

Gather information about the files with given extension and return the corresponding wxFileType object or NULL if the extension is unknown.

The *extension* parameter may have, or not, the leading dot, if it has it, it is stripped automatically. It must not however be empty.

wxFileType* wxMimeTypesManager::GetFileTypeFromMimeType (const wxString & *mimeType*)

Gather information about the files with given MIME type and return the corresponding wxFileType object or NULL if the MIME type is unknown.

static bool wxMimeTypesManager::IsOfType (const wxString & *mimeType,* const wxString & *wildcard*) `[static]`

This function returns true if either the given *mimeType* is exactly the same as *wildcard* or if it has the same category and the subtype of *wildcard* is '*'.

Note that the '*' wildcard is not allowed in *mimeType* itself.

The comparison done by this function is case insensitive so it is not necessary to convert the strings to the same case before calling it.

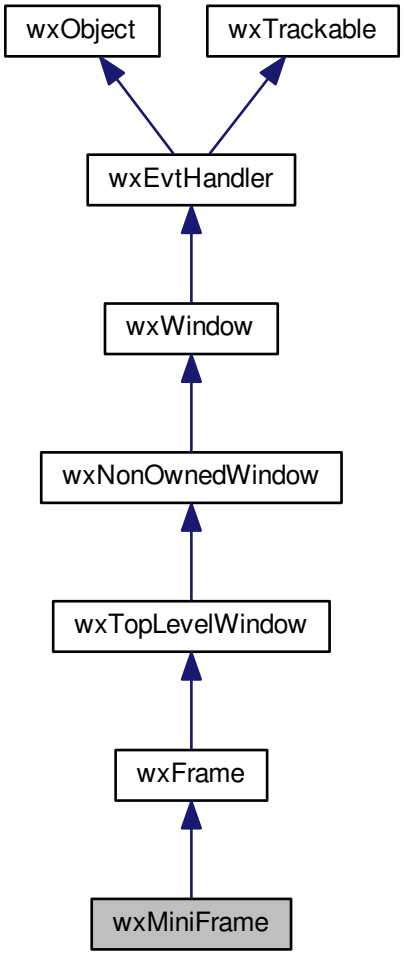
bool wxMimeTypesManager::Unassociate (wxFileType * *ft*)

Undo Associate().

21.476 wxMiniFrame Class Reference

```
#include <wx/minifram.h>
```

Inheritance diagram for wxMiniFrame:



21.476.1 Detailed Description

A miniframe is a frame with a small title bar.

It is suitable for floating toolbars that must not take up too much screen area.

An example of mini frame can be seen in the Dialogs Sample using the "Mini frame" command of the "Generic dialogs" submenu.

Styles

This class supports the following styles:

- wxICONIZE: Display the frame iconized (minimized) (Windows only).
- wxCAPTION: Puts a caption on the frame.
- wxMINIMIZE: Identical to wxICONIZE.

- wxMINIMIZE_BOX: Displays a minimize box on the frame (Windows and Motif only).
- wxMAXIMIZE: Displays the frame maximized (Windows only).
- wxMAXIMIZE_BOX: Displays a maximize box on the frame (Windows and Motif only).
- wxCLOSE_BOX: Displays a close box on the frame.
- wxSTAY_ON_TOP: Stay on top of other windows (Windows only).
- wxSYSTEM_MENU: Displays a system menu (Windows and Motif only).
- wxRESIZE_BORDER: Displays a resizable border around the window.

Remarks

This class has miniframe functionality under Windows and GTK, i.e. the presence of mini frame will not be noted in the task bar and focus behaviour is different. On other platforms, it behaves like a normal frame.

Library: wxCore

Category: Managed Windows

See also

wxMDIParentFrame, wxMDIChildFrame, wxFrame, wxDialog

Public Member Functions

- wxMiniFrame ()

  Default ctor.
- wxMiniFrame (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCAPTION|wxRESIZE_BORDER, const wxString &name=wxFrameNameStr)

  Constructor, creating the window.
- virtual ~wxMiniFrame ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCAPTION|wxRESIZE_BORDER, const wxString &name=wxFrameNameStr)

  Used in two-step frame construction.

Additional Inherited Members

21.476.2 Constructor & Destructor Documentation

wxMiniFrame::wxMiniFrame ( )

Default ctor.

wxMiniFrame::wxMiniFrame (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCAPTION|wxRESIZE_BORDER, const wxString & *name* = `wxFrameNameStr`)

Constructor, creating the window.

Parameters

parent	The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.
id	The window identifier. It may take a value of -1 to indicate a default value.
title	The caption to be displayed on the frame's title bar.
pos	The window position. The value wxDefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.
size	The window size. The value wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.
style	The window style. See wxMiniFrame.
name	The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

Remarks

The frame behaves like a normal frame on non-Windows platforms.

See also

Create()

virtual wxMiniFrame::~wxMiniFrame ( ) [virtual]

Destructor.

Destroys all child windows and menu bar if present.

21.476.3 Member Function Documentation

bool wxMiniFrame::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCAPTION|wxRESIZE_BORDER, const wxString & *name* = wxFrameNameStr)

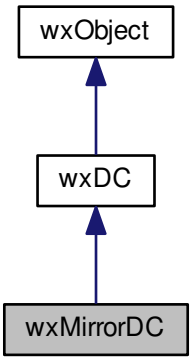
Used in two-step frame construction.

See wxMiniFrame() for further details.

21.477 wxMirrorDC Class Reference

`#include <wx/dcmirror.h>`

Inheritance diagram for wxMirrorDC:



21.477.1 Detailed Description

wxMirrorDC is a simple wrapper class which is always associated with a real wxDC object and either forwards all of its operations to it without changes (no mirroring takes place) or exchanges *x* and *y* coordinates which makes it possible to reuse the same code to draw a figure and its mirror – i.e.

reflection related to the diagonal line x == y.

Since

2.5.0

Library: wxCore

Category: Device Contexts

Public Member Functions

- wxMirrorDC (wxDC &dc, bool mirror)

  Creates a (maybe) mirrored DC associated with the real dc.

Additional Inherited Members

21.477.2 Constructor & Destructor Documentation

wxMirrorDC::wxMirrorDC (wxDC & *dc*, bool *mirror*)

Creates a (maybe) mirrored DC associated with the real *dc*.

Everything drawn on wxMirrorDC will appear (and maybe mirrored) on *dc*.

mirror specifies if we do mirror (if it is true) or not (if it is false).

21.478 wxModalDialogHook Class Reference

`#include <wx/modalhook.h>`

21.478.1 Detailed Description

Allows to intercept all modal dialog calls.

This class can be used to hook into normal modal dialog handling for some special needs. One of the most common use cases is for testing: as automatic tests can't continue if a modal dialog is shown while they run, this class can be used to avoid showing the modal dialogs during unattended execution. wxModalDialogHook can also be used for disabling some background operation while a modal dialog is shown.

To install a modal dialog hook, you need to derive your own class from this one and implement its pure virtual Enter() method. Then simply create an object of your class and call Register() on it to start receiving calls to your overridden Enter() (and possibly Exit()) methods:

```
class MyModalDialogHook : public wxModalDialogHook
{
protected:
    virtual int Enter(wxDialog* dialog)
    {
        // Just for demonstration purposes, intercept all uses of
        // wxFileDialog. Notice that this doesn't provide any real
        // sandboxing, of course, the program can still read and write
        // files by not using wxFileDialog to ask the user for their
        // names.
        if ( wxDynamicCast(dialog, wxFileDialog) )
        {
            wxLogError("Access to file system disallowed.");

            // Skip showing the file dialog entirely.
            return wxID_CANCEL;
        }

        m_lastEnter = wxDateTime::Now();

        // Allow the dialog to be shown as usual.
        return wxID_NONE;
    }

    virtual void Exit(wxDialog* dialog)
    {
        // Again, just for demonstration purposes, show how long did
        // the user take to dismiss the dialog. Notice that we
        // shouldn't use wxLogMessage() here as this would result in
        // another modal dialog call and hence infinite recursion. In
        // general, the hooks should be as unintrusive as possible.
        wxLogDebug("%s dialog took %s to be dismissed",
                   dialog->GetClassInfo()->GetClassName(),
                   (wxDateTime::Now() - m_lastEnter).Format());
    }
};

class MyApp : public wxApp
{
public:
    virtual bool OnInit()
    {
        ...
        m_myHook.Register();
        ...
    }

private:
    MyModalDialogHook m_myHook;
};
```

Since

2.9.5

Public Member Functions

- wxModalDialogHook ()

Default and trivial constructor.

- virtual ~wxModalDialogHook ()

  Destructor unregisters the hook if it's currently active.

- void Register ()

  Register this hook as being active.

- void Unregister ()

  Unregister this hook.

Protected Member Functions

- virtual int Enter (wxDialog *dialog)=0

  Called by wxWidgets before showing any modal dialogs.

- virtual void Exit (wxDialog *dialog)

  Called by wxWidgets after dismissing the modal dialog.

21.478.2 Constructor & Destructor Documentation

wxModalDialogHook::wxModalDialogHook ( )

Default and trivial constructor.

The constructor doesn't do anything, call Register() to make this hook active.

virtual wxModalDialogHook::~wxModalDialogHook ( ) [virtual]

Destructor unregisters the hook if it's currently active.

21.478.3 Member Function Documentation

virtual int wxModalDialogHook::Enter (wxDialog * *dialog*) [protected], [pure virtual]

Called by wxWidgets before showing any modal dialogs.

Override this to be notified whenever a modal dialog is about to be shown.

If the return value of this method is wxID_NONE, the dialog is shown as usual and Exit() will be called when it is dismissed. If the return value is anything else, the dialog is not shown at all and its wxDialog::ShowModal() simply returns with the given result. In this case, Exit() won't be called neither.

Parameters

dialog	The dialog about to be shown, never NULL.

Returns

wxID_NONE to continue with showing the dialog or anything else to skip showing the dialog and just return this value from its ShowModal().

virtual void wxModalDialogHook::Exit (wxDialog * *dialog*) [protected], [virtual]

Called by wxWidgets after dismissing the modal dialog.

Notice that it won't be called if Enter() hadn't been called because another modal hook, registered after this one, intercepted the dialog or if our Enter() was called but returned a value different from wxID_NONE.

Parameters

dialog	The dialog that was shown and dismissed, never NULL.

void wxModalDialogHook::Register ( )

Register this hook as being active.

After registering the hook, its Enter() and Exit() methods will be called whenever a modal dialog is shown.

Notice that the order of registration matters: the last hook registered is called first, and if its Enter() returns a value different from wxID_NONE, the subsequent hooks are skipped.

It is an error to register the same hook twice.

void wxModalDialogHook::Unregister ( )

Unregister this hook.

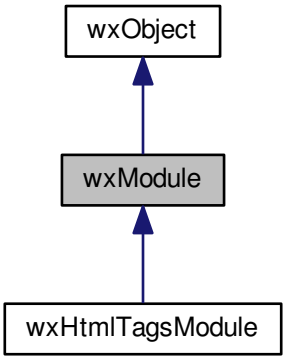
Notice that is done automatically from the destructor, so usually calling this method explicitly is unnecessary.

The hook must be currently registered.

21.479 wxModule Class Reference

`#include <wx/module.h>`

Inheritance diagram for wxModule:



21.479.1 Detailed Description

The module system is a very simple mechanism to allow applications (and parts of wxWidgets itself) to define initialization and cleanup functions that are automatically called on wxWidgets startup and exit.

To define a new kind of module, derive a class from wxModule, override the wxModule::OnInit and wxModule::On↩ Exit functions, and add the wxDECLARE_DYNAMIC_CLASS and wxIMPLEMENT_DYNAMIC_CLASS to header and implementation files (which can be the same file). On initialization, wxWidgets will find all classes derived from

wxModule, create an instance of each, and call each wxModule::OnInit function. On exit, wxWidgets will call the wxModule::OnExit function for each module instance.

Note that your module class does not have to be in a header file.

For example:

```
  // A module to allow DDE initialization/cleanup
// without calling these functions from app.cpp or from
// the user's application.
class wxDDEModule: public wxModule
{
public:
    wxDDEModule() { }
    virtual bool OnInit() { wxDDEInitialize(); return true; };
    virtual void OnExit() { wxDDECleanUp(); };

private:
    wxDECLARE_DYNAMIC_CLASS(wxDDEModule);
};

wxIMPLEMENT_DYNAMIC_CLASS(wxDDEModule, wxModule);

// Another module which uses DDE in its OnInit()
class MyModule: public wxModule
{
public:
    MyModule() { AddDependency(wxCLASSINFO(wxDDEModule)); }
    virtual bool OnInit() { ... code using DDE ... }
    virtual void OnExit() { ... }

private:
    wxDECLARE_DYNAMIC_CLASS(MyModule);
};

wxIMPLEMENT_DYNAMIC_CLASS(MyModule, wxModule);

// Another module which uses DDE in its OnInit()
// but uses a named dependency
class MyModule2: public wxModule
{
public:
    MyModule2() { AddDependency("wxDDEModule"); }
    virtual bool OnInit() { ... code using DDE ... }
    virtual void OnExit() { ... }

private:
    wxDECLARE_DYNAMIC_CLASS(MyModule2)
};

wxIMPLEMENT_DYNAMIC_CLASS(MyModule2, wxModule)
```

Library: wxBase

Category: Application and Process Management

Public Member Functions

- wxModule ()

  Constructs a wxModule object.
- virtual ~wxModule ()

  Destructor.
- virtual void OnExit ()=0

  Provide this function with appropriate cleanup for your module.
- virtual bool OnInit ()=0

  Provide this function with appropriate initialization for your module.

Protected Member Functions

- void AddDependency (wxClassInfo *dep)

Call this function from the constructor of the derived class.

- void AddDependency (const char *classname)

  Call this function from the constructor of the derived class.

Additional Inherited Members

21.479.2 Constructor & Destructor Documentation

wxModule::wxModule ( )

Constructs a wxModule object.

virtual wxModule::~wxModule ( ) [virtual]

Destructor.

21.479.3 Member Function Documentation

void wxModule::AddDependency (wxClassInfo * *dep*) [protected]

Call this function from the constructor of the derived class.

dep must be the wxCLASSINFO() of a wxModule-derived class and the corresponding module will be loaded before and unloaded after this module.

Parameters

dep	The class information object for the dependent module.

void wxModule::AddDependency (const char * *classname*) [protected]

Call this function from the constructor of the derived class.

This overload allows a dependency to be added by name without access to the class info.

This is useful when a module is declared entirely in a source file and there is no header for the declaration of the module needed by wxCLASSINFO(), however errors are not detected until run-time, instead of compile-time, then. Note that circular dependencies are detected and result in a fatal error.

Parameters

classname	The class name of the dependent module.

virtual void wxModule::OnExit ( ) [pure virtual]

Provide this function with appropriate cleanup for your module.

virtual bool wxModule::OnInit ( ) [pure virtual]

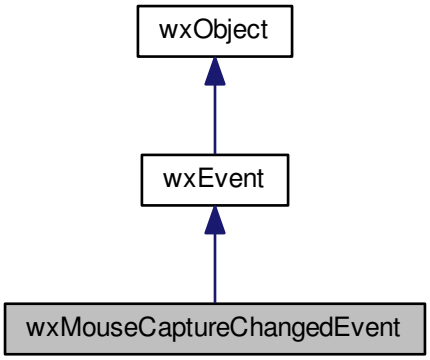
Provide this function with appropriate initialization for your module.

If the function returns false, wxWidgets will exit immediately.

21.480 wxMouseCaptureChangedEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxMouseCaptureChangedEvent:



21.480.1 Detailed Description

An mouse capture changed event is sent to a window that loses its mouse capture.

This is called even if wxWindow::ReleaseMouse was called by the application code. Handling this event allows an application to cater for unexpected capture releases which might otherwise confuse mouse handling code.

Availability: only available for the wxMSW port.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMouseCaptureChangedEvent& event)

Event macros:

- EVT_MOUSE_CAPTURE_CHANGED(func): Process a `wxEVT_MOUSE_CAPTURE_CHANGED` event.

Library: wxCore

Category: Events

See also

wxMouseCaptureLostEvent, Events and Event Handling, wxWindow::CaptureMouse, wxWindow::Release↩ Mouse, wxWindow::GetCapture

Public Member Functions

- wxMouseCaptureChangedEvent (wxWindowID windowId=0, wxWindow *gainedCapture=NULL)

Constructor.

- wxWindow * GetCapturedWindow () const

  Returns the window that gained the capture, or NULL if it was a non-wxWidgets window.

Additional Inherited Members

21.480.2 Constructor & Destructor Documentation

wxMouseCaptureChangedEvent::wxMouseCaptureChangedEvent (wxWindowID *windowId* = 0, wxWindow * *gainedCapture* = NULL)

Constructor.

21.480.3 Member Function Documentation

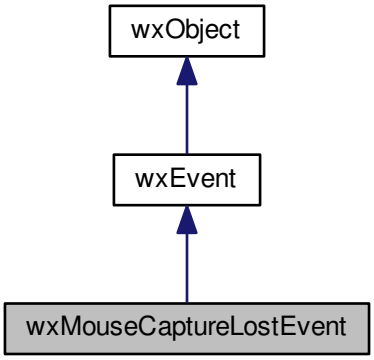
wxWindow* wxMouseCaptureChangedEvent::GetCapturedWindow ( ) const

Returns the window that gained the capture, or NULL if it was a non-wxWidgets window.

21.481 wxMouseCaptureLostEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxMouseCaptureLostEvent:



21.481.1 Detailed Description

A mouse capture lost event is sent to a window that had obtained mouse capture, which was subsequently lost due to an "external" event (for example, when a dialog box is shown or if another application captures the mouse).

If this happens, this event is sent to all windows that are on the capture stack (i.e. called CaptureMouse, but didn't call ReleaseMouse yet). The event is not sent if the capture changes because of a call to CaptureMouse or ReleaseMouse.

This event is currently emitted under Windows only.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMouseCaptureLostEvent& event)

Event macros:

- EVT_MOUSE_CAPTURE_LOST(func): Process a `wxEVT_MOUSE_CAPTURE_LOST` event.

Availability: only available for the wxMSW port.

Library: wxCore

Category: Events

See also

wxMouseCaptureChangedEvent, Events and Event Handling, wxWindow::CaptureMouse, wxWindow::ReleaseMouse, wxWindow::GetCapture

Public Member Functions

- wxMouseCaptureLostEvent (wxWindowID windowId=0)

  Constructor.

Additional Inherited Members

21.481.2 Constructor & Destructor Documentation

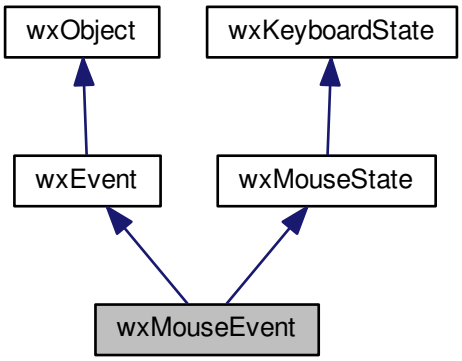
wxMouseCaptureLostEvent::wxMouseCaptureLostEvent (wxWindowID *windowId* = `0`)

Constructor.

21.482 wxMouseEvent Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxMouseEvent:



21.482.1 Detailed Description

This event class contains information about the events generated by the mouse: they include mouse buttons press and release events and mouse move events.

All mouse events involving the buttons use `wxMOUSE_BTN_LEFT` for the left mouse button, `wxMOUSE_BTN_MIDDLE` for the middle one and `wxMOUSE_BTN_RIGHT` for the right one. And if the system supports more buttons, the `wxMOUSE_BTN_AUX1` and `wxMOUSE_BTN_AUX2` events can also be generated. Note that not all mice have even a middle button so a portable application should avoid relying on the events from it (but the right button click can be emulated using the left mouse button with the control key under Mac platforms with a single button mouse).

For the `wxEVT_ENTER_WINDOW` and `wxEVT_LEAVE_WINDOW` events purposes, the mouse is considered to be inside the window if it is in the window client area and not inside one of its children. In other words, the parent window receives `wxEVT_LEAVE_WINDOW` event not only when the mouse leaves the window entirely but also when it enters one of its children.

The position associated with a mouse event is expressed in the window coordinates of the window which generated the event, you can use wxWindow::ClientToScreen() to convert it to screen coordinates and possibly call wxWindow::ScreenToClient() next to convert it to window coordinates of another window.

Note

Note that under Windows CE mouse enter and leave events are not natively supported by the system but are generated by wxWidgets itself. This has several drawbacks: the LEAVE_WINDOW event might be received some time after the mouse left the window and the state variables for it may have changed during this time.
Note the difference between methods like wxMouseEvent::LeftDown and the inherited wxMouseState::LeftIsDown: the former returns true when the event corresponds to the left mouse button click while the latter returns true if the left mouse button is currently being pressed. For example, when the user is dragging the mouse you can use wxMouseEvent::LeftIsDown to test whether the left mouse button is (still) depressed. Also, by convention, if wxMouseEvent::LeftDown returns true, wxMouseEvent::LeftIsDown will also return true in wxWidgets whatever the underlying GUI behaviour is (which is platform-dependent). The same applies, of course, to other mouse buttons as well.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMouseEvent& event)

Event macros:

- EVT_LEFT_DOWN(func): Process a `wxEVT_LEFT_DOWN` event. The handler of this event should normally call event.Skip() to allow the default processing to take place as otherwise the window under mouse wouldn't get the focus.
- EVT_LEFT_UP(func): Process a `wxEVT_LEFT_UP` event.
- EVT_LEFT_DCLICK(func): Process a `wxEVT_LEFT_DCLICK` event.
- EVT_MIDDLE_DOWN(func): Process a `wxEVT_MIDDLE_DOWN` event.
- EVT_MIDDLE_UP(func): Process a `wxEVT_MIDDLE_UP` event.
- EVT_MIDDLE_DCLICK(func): Process a `wxEVT_MIDDLE_DCLICK` event.
- EVT_RIGHT_DOWN(func): Process a `wxEVT_RIGHT_DOWN` event.
- EVT_RIGHT_UP(func): Process a `wxEVT_RIGHT_UP` event.
- EVT_RIGHT_DCLICK(func): Process a `wxEVT_RIGHT_DCLICK` event.
- EVT_MOUSE_AUX1_DOWN(func): Process a `wxEVT_AUX1_DOWN` event.
- EVT_MOUSE_AUX1_UP(func): Process a `wxEVT_AUX1_UP` event.
- EVT_MOUSE_AUX1_DCLICK(func): Process a `wxEVT_AUX1_DCLICK` event.
- EVT_MOUSE_AUX2_DOWN(func): Process a `wxEVT_AUX2_DOWN` event.
- EVT_MOUSE_AUX2_UP(func): Process a `wxEVT_AUX2_UP` event.
- EVT_MOUSE_AUX2_DCLICK(func): Process a `wxEVT_AUX2_DCLICK` event.
- EVT_MOTION(func): Process a `wxEVT_MOTION` event.
- EVT_ENTER_WINDOW(func): Process a `wxEVT_ENTER_WINDOW` event.
- EVT_LEAVE_WINDOW(func): Process a `wxEVT_LEAVE_WINDOW` event.
- EVT_MOUSEWHEEL(func): Process a `wxEVT_MOUSEWHEEL` event.
- EVT_MOUSE_EVENTS(func): Process all mouse events.
- EVT_MAGNIFY(func): Process a `wxEVT_MAGNIFY` event (new since wxWidgets 3.1.0).

Library: wxCore

Category: Events

See also

wxKeyEvent

Public Member Functions

- wxMouseEvent (wxEventType mouseEventType=wxEVT_NULL)

  Constructor.
- bool Aux1DClick () const

  Returns true if the event was a first extra button double click.
- bool Aux1Down () const

  Returns true if the first extra button mouse button changed to down.

- bool Aux1Up () const

  Returns true if the first extra button mouse button changed to up.
- bool Aux2DClick () const

  Returns true if the event was a second extra button double click.
- bool Aux2Down () const

  Returns true if the second extra button mouse button changed to down.
- bool Aux2Up () const

  Returns true if the second extra button mouse button changed to up.
- bool Button (wxMouseButton but) const

  Returns true if the event was generated by the specified button.
- bool ButtonDClick (wxMouseButton but=wxMOUSE_BTN_ANY) const

  If the argument is omitted, this returns true if the event was a mouse double click event.
- bool ButtonDown (wxMouseButton but=wxMOUSE_BTN_ANY) const

  If the argument is omitted, this returns true if the event was a mouse button down event.
- bool ButtonUp (wxMouseButton but=wxMOUSE_BTN_ANY) const

  If the argument is omitted, this returns true if the event was a mouse button up event.
- bool Dragging () const

  Returns true if this was a dragging event (motion while a button is depressed).
- bool Entering () const

  Returns true if the mouse was entering the window.
- int GetButton () const

  Returns the mouse button which generated this event or `wxMOUSE_BTN_NONE` if no button is involved (for mouse move, enter or leave event, for example).
- int GetClickCount () const

  Returns the number of mouse clicks for this event: 1 for a simple click, 2 for a double-click, 3 for a triple-click and so on.
- int GetLinesPerAction () const

  Returns the configured number of lines (or whatever) to be scrolled per wheel action.
- int GetColumnsPerAction () const

  Returns the configured number of columns (or whatever) to be scrolled per wheel action.
- wxPoint GetLogicalPosition (const wxDC &dc) const

  Returns the logical mouse position in pixels (i.e. translated according to the translation set for the DC, which usually indicates that the window has been scrolled).
- float GetMagnification () const

  For magnify (pinch to zoom) events: returns the change in magnification.
- int GetWheelDelta () const

  Get wheel delta, normally 120.
- int GetWheelRotation () const

  Get wheel rotation, positive or negative indicates direction of rotation.
- wxMouseWheelAxis GetWheelAxis () const

  Gets the axis the wheel operation concerns.
- bool IsButton () const

  Returns true if the event was a mouse button event (not necessarily a button down event - that may be tested using ButtonDown()).
- bool IsPageScroll () const

  Returns true if the system has been setup to do page scrolling with the mouse wheel instead of line scrolling.
- bool Leaving () const

  Returns true if the mouse was leaving the window.
- bool LeftDClick () const

  Returns true if the event was a left double click.
- bool LeftDown () const

Returns true if the left mouse button changed to down.

- bool LeftUp () const

  Returns true if the left mouse button changed to up.

- bool Magnify () const

  Returns true if the event is a magnify (i.e. pinch to zoom) event.

- bool MetaDown () const

  Returns true if the Meta key was down at the time of the event.

- bool MiddleDClick () const

  Returns true if the event was a middle double click.

- bool MiddleDown () const

  Returns true if the middle mouse button changed to down.

- bool MiddleUp () const

  Returns true if the middle mouse button changed to up.

- bool Moving () const

  Returns true if this was a motion event and no mouse buttons were pressed.

- bool RightDClick () const

  Returns true if the event was a right double click.

- bool RightDown () const

  Returns true if the right mouse button changed to down.

- bool RightUp () const

  Returns true if the right mouse button changed to up.

Additional Inherited Members

21.482.2 Constructor & Destructor Documentation

wxMouseEvent::wxMouseEvent (wxEventType *mouseEventType* = wxEVT_NULL)

Constructor.

Valid event types are:

- `wxEVT_ENTER_WINDOW`
- `wxEVT_LEAVE_WINDOW`
- `wxEVT_LEFT_DOWN`
- `wxEVT_LEFT_UP`
- `wxEVT_LEFT_DCLICK`
- `wxEVT_MIDDLE_DOWN`
- `wxEVT_MIDDLE_UP`
- `wxEVT_MIDDLE_DCLICK`
- `wxEVT_RIGHT_DOWN`
- `wxEVT_RIGHT_UP`
- `wxEVT_RIGHT_DCLICK`
- `wxEVT_AUX1_DOWN`
- `wxEVT_AUX1_UP`
- `wxEVT_AUX1_DCLICK`

- wxEVT_AUX2_DOWN
- wxEVT_AUX2_UP
- wxEVT_AUX2_DCLICK
- wxEVT_MOTION
- wxEVT_MOUSEWHEEL
- wxEVT_MAGNIFY

21.482.3 Member Function Documentation

bool wxMouseEvent::Aux1DClick ( ) const

Returns true if the event was a first extra button double click.

bool wxMouseEvent::Aux1Down ( ) const

Returns true if the first extra button mouse button changed to down.

bool wxMouseEvent::Aux1Up ( ) const

Returns true if the first extra button mouse button changed to up.

bool wxMouseEvent::Aux2DClick ( ) const

Returns true if the event was a second extra button double click.

bool wxMouseEvent::Aux2Down ( ) const

Returns true if the second extra button mouse button changed to down.

bool wxMouseEvent::Aux2Up ( ) const

Returns true if the second extra button mouse button changed to up.

bool wxMouseEvent::Button (wxMouseButton *but*) const

Returns true if the event was generated by the specified button.

See also

wxMouseState::ButtoinIsDown()

bool wxMouseEvent::ButtonDClick (wxMouseButton *but* = wxMOUSE_BTN_ANY) const

If the argument is omitted, this returns true if the event was a mouse double click event.

Otherwise the argument specifies which double click event was generated (see Button() for the possible values).

bool wxMouseEvent::ButtonDown (wxMouseButton *but* = wxMOUSE_BTN_ANY) const

If the argument is omitted, this returns true if the event was a mouse button down event.

Otherwise the argument specifies which button-down event was generated (see Button() for the possible values).

bool wxMouseEvent::ButtonUp (wxMouseButton *but* = wxMOUSE_BTN_ANY) const

If the argument is omitted, this returns true if the event was a mouse button up event.

Otherwise the argument specifies which button-up event was generated (see Button() for the possible values).

bool wxMouseEvent::Dragging ( ) const

Returns true if this was a dragging event (motion while a button is depressed).

See also

Moving()

bool wxMouseEvent::Entering ( ) const

Returns true if the mouse was entering the window.

See also

Leaving()

int wxMouseEvent::GetButton ( ) const

Returns the mouse button which generated this event or `wxMOUSE_BTN_NONE` if no button is involved (for mouse move, enter or leave event, for example).

Otherwise `wxMOUSE_BTN_LEFT` is returned for the left button down, up and double click events, `wxMOUSE_BTN_MIDDLE` and `wxMOUSE_BTN_RIGHT` for the same events for the middle and the right buttons respectively.

int wxMouseEvent::GetClickCount ( ) const

Returns the number of mouse clicks for this event: 1 for a simple click, 2 for a double-click, 3 for a triple-click and so on.

Currently this function is implemented only in wxMac and returns -1 for the other platforms (you can still distinguish simple clicks from double-clicks as they generate different kinds of events however).

Since

2.9.0

int wxMouseEvent::GetColumnsPerAction ( ) const

Returns the configured number of columns (or whatever) to be scrolled per wheel action.

Default value under most platforms is three.

See also

GetLinesPerAction()

Since

2.9.5

int wxMouseEvent::GetLinesPerAction ( ) const

Returns the configured number of lines (or whatever) to be scrolled per wheel action.

Default value under most platforms is three.

See also

GetColumnsPerAction()

wxPoint wxMouseEvent::GetLogicalPosition (const wxDC & *dc*) const

Returns the logical mouse position in pixels (i.e. translated according to the translation set for the DC, which usually indicates that the window has been scrolled).

float wxMouseEvent::GetMagnification ( ) const

For magnify (pinch to zoom) events: returns the change in magnification.

A value of 0 means no change, a positive value means we should enlarge (or zoom in), a negative value means we should shrink (or zoom out).

This method is only valid to call for `wxEVT_MAGNIFY` events which are currently only generated under OS X.

See also

Magnify()

Since

3.1.0

wxMouseWheelAxis wxMouseEvent::GetWheelAxis ( ) const

Gets the axis the wheel operation concerns.

Usually the mouse wheel is used to scroll vertically so `wxMOUSE_WHEEL_VERTICAL` is returned but some mice (and most trackpads) also allow to use the wheel to scroll horizontally in which case `wxMOUSE_WHEEL_HORIZ↩ ONTAL` is returned.

Notice that before wxWidgets 2.9.4 this method returned `int`.

int wxMouseEvent::GetWheelDelta ( ) const

Get wheel delta, normally 120.

This is the threshold for action to be taken, and one such action (for example, scrolling one increment) should occur for each delta.

int wxMouseEvent::GetWheelRotation ( ) const

Get wheel rotation, positive or negative indicates direction of rotation.

Current devices all send an event when rotation is at least +/-WheelDelta, but finer resolution devices can be created in the future.

Because of this you shouldn't assume that one event is equal to 1 line, but you should be able to either do partial line scrolling or wait until several events accumulate before scrolling.

bool wxMouseEvent::IsButton ( ) const

Returns true if the event was a mouse button event (not necessarily a button down event - that may be tested using ButtonDown()).

bool wxMouseEvent::IsPageScroll ( ) const

Returns true if the system has been setup to do page scrolling with the mouse wheel instead of line scrolling.

bool wxMouseEvent::Leaving ( ) const

Returns true if the mouse was leaving the window.

See also

Entering().

bool wxMouseEvent::LeftDClick ( ) const

Returns true if the event was a left double click.

bool wxMouseEvent::LeftDown ( ) const

Returns true if the left mouse button changed to down.

bool wxMouseEvent::LeftUp ( ) const

Returns true if the left mouse button changed to up.

bool wxMouseEvent::Magnify ( ) const

Returns true if the event is a magnify (i.e. pinch to zoom) event.

Such events are currently generated only under OS X.

See also

GetMagnification()

Since

3.1.0

bool wxMouseEvent::MetaDown ( ) const

Returns true if the Meta key was down at the time of the event.

bool wxMouseEvent::MiddleDClick ( ) const

Returns true if the event was a middle double click.

bool wxMouseEvent::MiddleDown ( ) const

Returns true if the middle mouse button changed to down.

bool wxMouseEvent::MiddleUp ( ) const

Returns true if the middle mouse button changed to up.

bool wxMouseEvent::Moving ( ) const

Returns true if this was a motion event and no mouse buttons were pressed.

If any mouse button is held pressed, then this method returns false and Dragging() returns true.

bool wxMouseEvent::RightDClick ( ) const

Returns true if the event was a right double click.

bool wxMouseEvent::RightDown ( ) const

Returns true if the right mouse button changed to down.

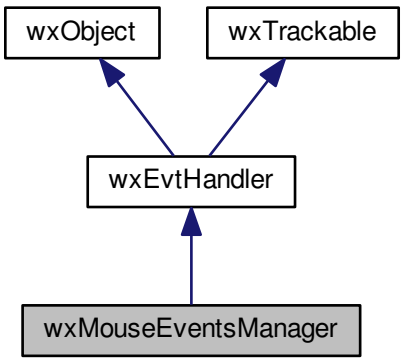
bool wxMouseEvent::RightUp ( ) const

Returns true if the right mouse button changed to up.

21.483 wxMouseEventsManager Class Reference

`#include <wx/mousemanager.h>`

Inheritance diagram for wxMouseEventsManager:



21.483.1 Detailed Description

Helper for handling mouse input events in windows containing multiple items.

This class handles mouse events and synthesizes high-level notifications such as clicks and drag events from low level mouse button presses and mouse movement events. It is useful because handling the mouse events is less obvious than might seem at a first glance: for example, clicks on an object should only be generated if the mouse was both pressed and released over it and not just released (so it requires storing the previous state) and dragging shouldn't start before the mouse moves away far enough.

This class encapsulates all these dull details for controls containing multiple items which can be identified by a positive integer index and you just need to implement its pure virtual functions to use it.

Notice that this class supposes that all items can be identified by an integer "index" but it doesn't need to be an ordinal index of the item (although this is the most common case) – it can be any value which can be used to uniquely identify an item.

Library: wxCore

Category: Events

Public Member Functions

- wxMouseEventsManager ()

  Default constructor.
- wxMouseEventsManager (wxWindow *win)

  Constructor creates the manager for the window.
- bool Create (wxWindow *win)

  Finishes initialization of the object created using default constructor.

Protected Member Functions

- virtual int MouseHitTest (const wxPoint &pos)=0

Must be overridden to return the item at the given position.

- virtual bool MouseClicked (int item)=0

  Must be overridden to react to mouse clicks.
- virtual bool MouseDragBegin (int item, const wxPoint &pos)=0

  Must be overridden to allow or deny dragging of the item.
- virtual void MouseDragging (int item, const wxPoint &pos)=0

  Must be overridden to provide feed back while an item is being dragged.
- virtual void MouseDragEnd (int item, const wxPoint &pos)=0

  Must be overridden to handle item drop.
- virtual void MouseDragCancelled (int item)=0

  Must be overridden to handle cancellation of mouse dragging.
- virtual void MouseClickBegin (int item)

  May be overridden to update the state of an item when it is pressed.
- virtual void MouseClickCancelled (int item)

  Must be overridden to reset the item appearance changed by MouseClickBegin().

Additional Inherited Members

21.483.2 Constructor & Destructor Documentation

wxMouseEventsManager::wxMouseEventsManager ( )

Default constructor.

You must call Create() to finish initializing the mouse events manager. If possible, avoid the use of this constructor in favour of the other one which fully initializes the mouse events manager immediately.

wxMouseEventsManager::wxMouseEventsManager (wxWindow * *win*)

Constructor creates the manager for the window.

A mouse event manager is always associated with a window and must be destroyed by the window when it is destroyed (it doesn't need to be allocated on the heap however).

21.483.3 Member Function Documentation

bool wxMouseEventsManager::Create (wxWindow * *win*)

Finishes initialization of the object created using default constructor.

Currently always returns true.

virtual void wxMouseEventsManager::MouseClickBegin (int *item*) `[protected], [virtual]`

May be overridden to update the state of an item when it is pressed.

This method is called when the item is becomes pressed and can be used to change its appearance when this happens. It is mostly useful for button-like items and doesn't need to be overridden if the items shouldn't change their appearance when pressed.

Parameters

item	The item being pressed.

virtual void wxMouseEventsManager::MouseClickCancelled (int *item*) `[protected], [virtual]`

Must be overridden to reset the item appearance changed by MouseClickBegin().

This method is called if the mouse capture was lost while the item was pressed and must be overridden to restore the default item appearance if it was changed in MouseClickBegin().

See also

MouseDragCancelled(), wxMouseCaptureLostEvent

virtual bool wxMouseEventsManager::MouseClicked (int *item*) `[protected], [pure virtual]`

Must be overridden to react to mouse clicks.

This method is called when the user clicked (i.e. pressed and released mouse over the *same* item) and should normally generate a notification about this click and return true if it was handled or false otherwise, determining whether the original mouse event is skipped or not.

Parameters

item	The item which was clicked.

Returns

true if the mouse event was processed and false otherwise.

virtual bool wxMouseEventsManager::MouseDragBegin (int *item*, const wxPoint & *pos*) `[protected], [pure virtual]`

Must be overridden to allow or deny dragging of the item.

This method is called when the user attempts to start dragging the given item.

Parameters

item	The item which is going to be dragged.
pos	The position from where it is being dragged.

Returns

true to allow the item to be dragged (in which case MouseDragging() and MouseDragEnd() will be called later, unless MouseDragCancelled() is called instead) or false to forbid it.

virtual void wxMouseEventsManager::MouseDragCancelled (int *item*) `[protected], [pure virtual]`

Must be overridden to handle cancellation of mouse dragging.

This method is called when mouse capture is lost while dragging the item and normally should remove the visual feedback drawn by MouseDragging() as well as reset any internal variables set in MouseDragBegin().

See also

wxMouseCaptureLostEvent

virtual void wxMouseEventsManager::MouseDragEnd (int *item,* const wxPoint & *pos*) `[protected], [pure virtual]`

Must be overridden to handle item drop.

This method is called when the mouse is released after dragging the item. Normally the item should be positioned at the new location.

Parameters

item	The item which was dragged and now dropped.
pos	The position at which the item was dropped.

See also

MouseDragBegin(), MouseDragging()

virtual void wxMouseEventsManager::MouseDragging (int *item,* const wxPoint & *pos*) `[protected], [pure virtual]`

Must be overridden to provide feed back while an item is being dragged.

This method is called while the item is being dragged and should normally update the feedback shown on screen (usually this is done using wxOverlay).

Notice that this method will never be called for the items for which MouseDragBegin() returns false. Consequently, if MouseDragBegin() always returns false you can do nothing in this method.

Parameters

item	The item being dragged.
pos	The current position of the item.

See also

MouseDragEnd()

virtual int wxMouseEventsManager::MouseHitTest (const wxPoint & *pos*) `[protected], [pure virtual]`

Must be overridden to return the item at the given position.

Parameters

pos	The position to test, in physical coordinates.

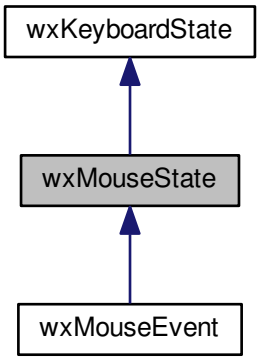
Returns

The index of the item at the given position or wxNOT_FOUND if there is no item there.

21.484 wxMouseState Class Reference

`#include <wx/mousestate.h>`

Inheritance diagram for wxMouseState:



21.484.1 Detailed Description

Represents the mouse state.

This class is used as a base class by wxMouseEvent and so its methods may be used to obtain information about the mouse state for the mouse events. It also inherits from wxKeyboardState and so carries information about the keyboard state and not only the mouse one.

This class is implemented entirely inline in <wx/mousestate.h> and thus has no linking requirements.

Library: None; this class implementation is entirely header-based.

Category: Events

See also

wxGetMouseState(), wxMouseEvent

Public Member Functions

- wxMouseState ()

  Default constructor.
- wxCoord GetX () const

  Returns X coordinate of the physical mouse event position.
- wxCoord GetY () const

  Returns Y coordinate of the physical mouse event position.
- bool LeftIsDown () const

  Returns true if the left mouse button is currently down.
- bool MiddleIsDown () const

  Returns true if the middle mouse button is currently down.
- bool RightIsDown () const

  Returns true if the right mouse button is currently down.

- bool Aux1IsDown () const

  Returns true if the first extra button mouse button is currently down.
- bool Aux2IsDown () const

  Returns true if the second extra button mouse button is currently down.
- void SetX (wxCoord x)
- void SetY (wxCoord y)
- void SetPosition (wxPoint pos)
- void SetLeftDown (bool down)
- void SetMiddleDown (bool down)
- void SetRightDown (bool down)
- void SetAux1Down (bool down)
- void SetAux2Down (bool down)
- void SetState (const wxMouseState &state)

- wxPoint GetPosition () const

  Returns the physical mouse position.
- void GetPosition (int *x, int *y) const

  Returns the physical mouse position.

21.484.2 Constructor & Destructor Documentation

wxMouseState::wxMouseState ( )

Default constructor.

21.484.3 Member Function Documentation

bool wxMouseState::Aux1IsDown ( ) const

Returns true if the first extra button mouse button is currently down.

bool wxMouseState::Aux2IsDown ( ) const

Returns true if the second extra button mouse button is currently down.

wxPoint wxMouseState::GetPosition ( ) const

Returns the physical mouse position.

void wxMouseState::GetPosition (int * *x*, int * *y*) const

Returns the physical mouse position.

wxCoord wxMouseState::GetX ( ) const

Returns X coordinate of the physical mouse event position.

wxCoord wxMouseState::GetY ( ) const

Returns Y coordinate of the physical mouse event position.

bool wxMouseState::LeftIsDown ( ) const

Returns true if the left mouse button is currently down.

bool wxMouseState::MiddleIsDown ( ) const

Returns true if the middle mouse button is currently down.

bool wxMouseState::RightIsDown ( ) const

Returns true if the right mouse button is currently down.

void wxMouseState::SetAux1Down (bool *down*)

void wxMouseState::SetAux2Down (bool *down*)

void wxMouseState::SetLeftDown (bool *down*)

void wxMouseState::SetMiddleDown (bool *down*)

void wxMouseState::SetPosition (wxPoint *pos*)

void wxMouseState::SetRightDown (bool *down*)

void wxMouseState::SetState (const wxMouseState & *state*)

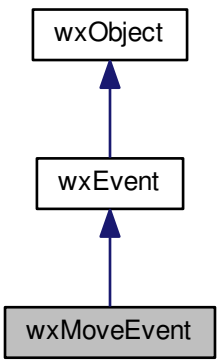
void wxMouseState::SetX (wxCoord *x*)

void wxMouseState::SetY (wxCoord *y*)

21.485 wxMoveEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxMoveEvent:



21.485.1 Detailed Description

A move event holds information about wxTopLevelWindow move change events.

These events are currently only generated by wxMSW port.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxMoveEvent& event)

Event macros:

- EVT_MOVE(func): Process a `wxEVT_MOVE` event, which is generated when a window is moved.
- EVT_MOVE_START(func): Process a `wxEVT_MOVE_START` event, which is generated when the user starts to move or size a window. wxMSW only.
- EVT_MOVING(func): Process a `wxEVT_MOVING` event, which is generated while the user is moving the window. wxMSW only.
- EVT_MOVE_END(func): Process a `wxEVT_MOVE_END` event, which is generated when the user stops moving or sizing a window. wxMSW only.

Library: wxCore

Category: Events

See also

wxPoint, Events and Event Handling

Public Member Functions

- wxMoveEvent (const wxPoint &pt, int id=0)

  Constructor.
- wxPoint GetPosition () const

  Returns the position of the window generating the move change event.
- wxRect GetRect () const
- void SetRect (const wxRect &rect)
- void SetPosition (const wxPoint &pos)

Additional Inherited Members

21.485.2 Constructor & Destructor Documentation

wxMoveEvent::wxMoveEvent (const wxPoint & *pt*, int *id* = `0`)

Constructor.

21.485.3 Member Function Documentation

wxPoint wxMoveEvent::GetPosition ( ) const

Returns the position of the window generating the move change event.

wxRect wxMoveEvent::GetRect ( ) const

void wxMoveEvent::SetPosition (const wxPoint & *pos*)

void wxMoveEvent::SetRect (const wxRect & *rect*)

21.486 wxMsgCatalog Class Reference

`#include <wx/translation.h>`

21.486.1 Detailed Description

Represents a loaded translations message catalog.

This class should only be used directly by wxTranslationsLoader implementations.

Since

2.9.1

Static Public Member Functions

- static wxMsgCatalog * CreateFromFile (const wxString &filename, const wxString &domain)

  Creates catalog loaded from a MO file.
- static wxMsgCatalog * CreateFromData (const wxScopedCharBuffer &data, const wxString &domain)

  Creates catalog from MO file data in memory buffer.

21.486.2 Member Function Documentation

static wxMsgCatalog* wxMsgCatalog::CreateFromData (const wxScopedCharBuffer & *data,* const wxString & *domain*) [static]

Creates catalog from MO file data in memory buffer.

Parameters

data	Data in MO file format.
domain	Catalog's domain. This typically matches the *filename*.

Returns

Successfully loaded catalog or NULL on failure.

static wxMsgCatalog* wxMsgCatalog::CreateFromFile (const wxString & *filename,* const wxString & *domain*) [static]

Creates catalog loaded from a MO file.

Parameters

filename	Path to the MO file to load.

domain	Catalog's domain. This typically matches the *filename*.

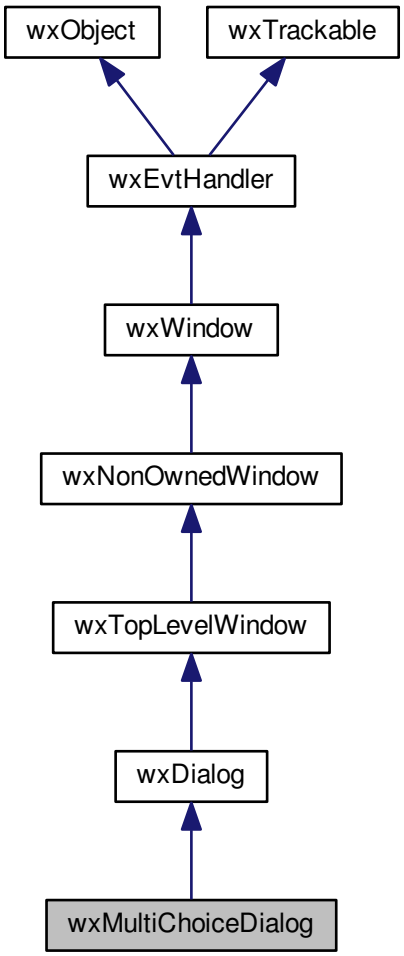
Returns

Successfully loaded catalog or NULL on failure.

21.487 wxMultiChoiceDialog Class Reference

`#include <wx/choicdlg.h>`

Inheritance diagram for wxMultiChoiceDialog:



21.487.1 Detailed Description

This class represents a dialog that shows a list of strings, and allows the user to select one or more.

Styles

This class supports the following styles:

- wxOK: Show an OK button.
- wxCANCEL: Show a Cancel button.
- wxCENTRE: Centre the message. Not Windows.

Library: wxBase

Category: Common Dialogs

See also

wxMultiChoiceDialog Overview, wxSingleChoiceDialog

Public Member Functions

- wxArrayInt GetSelections () const

  Returns array with indexes of selected items.
- void SetSelections (const wxArrayInt &selections)

  Sets selected items from the array of selected items' indexes.
- int ShowModal ()

  Shows the dialog, returning either wxID_OK or wxID_CANCEL.

- wxMultiChoiceDialog (wxWindow *parent, const wxString &message, const wxString &caption, int n, const wxString *choices, long style=wxCHOICEDLG_STYLE, const wxPoint &pos=wxDefaultPosition)

  Constructor taking an array of wxString choices.
- wxMultiChoiceDialog (wxWindow *parent, const wxString &message, const wxString &caption, const wxArrayString &choices, long style=wxCHOICEDLG_STYLE, const wxPoint &pos=wxDefaultPosition)

  Constructor taking an array of wxString choices.

Additional Inherited Members

21.487.2 Constructor & Destructor Documentation

wxMultiChoiceDialog::wxMultiChoiceDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption*, int *n*, const wxString * *choices*, long *style* = wxCHOICEDLG_STYLE, const wxPoint & *pos* = wxDefaultPosition)

Constructor taking an array of wxString choices.

Parameters

parent	Parent window.
message	Message to show on the dialog.
caption	The dialog caption.
n	The number of choices.

choices	An array of strings, or a string list, containing the choices.
style	A dialog style (bitlist) containing flags chosen from standard dialog styles and the ones listed in the class documentation. The default value is equivalent to wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER | wxOK | wxCANCEL | wxCENTRE.
pos	Dialog position. Not Windows.

Remarks

Use ShowModal() to show the dialog.

wxPerl Note: Not supported by wxPerl.

wxMultiChoiceDialog::wxMultiChoiceDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption*, const wxArrayString & *choices*, long *style* = wxCHOICEDLG_STYLE, const wxPoint & *pos* = wxDefaultPosition)

Constructor taking an array of wxString choices.

Parameters

parent	Parent window.
message	Message to show on the dialog.
caption	The dialog caption.
choices	An array of strings, or a string list, containing the choices.
style	A dialog style (bitlist) containing flags chosen from standard dialog styles and the ones listed in the class documentation. The default value is equivalent to wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER | wxOK | wxCANCEL | wxCENTRE.
pos	Dialog position. Not Windows.

Remarks

Use ShowModal() to show the dialog.

wxPerl Note: Use an array reference for the *choices* parameter.

21.487.3 Member Function Documentation

wxArrayInt wxMultiChoiceDialog::GetSelections ( ) const

Returns array with indexes of selected items.

void wxMultiChoiceDialog::SetSelections (const wxArrayInt & *selections*)

Sets selected items from the array of selected items' indexes.

int wxMultiChoiceDialog::ShowModal ( ) [virtual]

Shows the dialog, returning either wxID_OK or wxID_CANCEL.

Reimplemented from wxDialog.

21.488 wxMutex Class Reference

```
#include <wx/thread.h>
```

21.488.1 Detailed Description

A mutex object is a synchronization object whose state is set to signaled when it is not owned by any thread, and nonsignaled when it is owned.

Its name comes from its usefulness in coordinating mutually-exclusive access to a shared resource as only one thread at a time can own a mutex object.

Mutexes may be recursive in the sense that a thread can lock a mutex which it had already locked before (instead of dead locking the entire process in this situation by starting to wait on a mutex which will never be released while the thread is waiting) but using them is not recommended under Unix and they are **not** recursive by default. The reason for this is that recursive mutexes are not supported by all Unix flavours and, worse, they cannot be used with wxCondition.

For example, when several threads use the data stored in the linked list, modifications to the list should only be allowed to one thread at a time because during a new node addition the list integrity is temporarily broken (this is also called *program invariant*).

```
// this variable has an "s_" prefix because it is static: seeing an "s_" in
// a multithreaded program is in general a good sign that you should use a
// mutex (or a critical section)
static wxMutex *s_mutexProtectingTheGlobalData;

// we store some numbers in this global array which is presumably used by
// several threads simultaneously
wxArrayInt s_data;

void MyThread::AddNewNode(int num)
{
    // ensure that no other thread accesses the list
    s_mutexProtectingTheGlobalList->Lock();

    s_data.Add(num);

    s_mutexProtectingTheGlobalList->Unlock();
}

// return true if the given number is greater than all array elements
bool MyThread::IsGreater(int num)
{
    // before using the list we must acquire the mutex
    wxMutexLocker lock(s_mutexProtectingTheGlobalData);

    size_t count = s_data.Count();
    for ( size_t n = 0; n < count; n++ )
    {
        if ( s_data[n] > num )
            return false;
    }

    return true;
}
```

Notice how wxMutexLocker was used in the second function to ensure that the mutex is unlocked in any case: whether the function returns true or false (because the destructor of the local object *lock* is always called). Using this class instead of directly using wxMutex is, in general, safer and is even more so if your program uses C++ exceptions.

Library: wxBase

Category: Threading

See also

wxThread, wxCondition, wxMutexLocker, wxCriticalSection

Public Member Functions

- wxMutex (wxMutexType type=wxMUTEX_DEFAULT)

Default constructor.

- ~wxMutex ()

  Destroys the wxMutex object.

- wxMutexError Lock ()

  Locks the mutex object.

- wxMutexError LockTimeout (unsigned long msec)

  Try to lock the mutex object during the specified time interval.

- wxMutexError TryLock ()

  Tries to lock the mutex object.

- wxMutexError Unlock ()

  Unlocks the mutex object.

21.488.2 Constructor & Destructor Documentation

wxMutex::wxMutex (wxMutexType *type* = wxMUTEX_DEFAULT)

Default constructor.

wxMutex::~wxMutex ( )

Destroys the wxMutex object.

21.488.3 Member Function Documentation

wxMutexError wxMutex::Lock ( )

Locks the mutex object.

This is equivalent to LockTimeout() with infinite timeout.

Note that if this mutex is already locked by the caller thread, this function doesn't block but rather immediately returns.

Returns

One of: `wxMUTEX_NO_ERROR`, `wxMUTEX_DEAD_LOCK`.

wxMutexError wxMutex::LockTimeout (unsigned long *msec*)

Try to lock the mutex object during the specified time interval.

Returns

One of: `wxMUTEX_NO_ERROR`, `wxMUTEX_DEAD_LOCK`, `wxMUTEX_TIMEOUT`.

wxMutexError wxMutex::TryLock ( )

Tries to lock the mutex object.

If it can't, returns immediately with an error.

Returns

One of: `wxMUTEX_NO_ERROR`, `wxMUTEX_BUSY`.

wxMutexError wxMutex::Unlock ( )

Unlocks the mutex object.

Returns

One of: `wxMUTEX_NO_ERROR`, `wxMUTEX_UNLOCKED`.

21.489 wxMutexLocker Class Reference

```
#include <wx/thread.h>
```

21.489.1 Detailed Description

This is a small helper class to be used with wxMutex objects.

A wxMutexLocker acquires a mutex lock in the constructor and releases (or unlocks) the mutex in the destructor making it much more difficult to forget to release a mutex (which, in general, will promptly lead to serious problems). See wxMutex for an example of wxMutexLocker usage.

Library: wxBase

Category: Threading

See also

wxMutex, wxCriticalSectionLocker

Public Member Functions

- wxMutexLocker (wxMutex &mutex)

  Constructs a wxMutexLocker object associated with mutex and locks it.
- ~wxMutexLocker ()

  Destructor releases the mutex if it was successfully acquired in the ctor.
- bool IsOk () const

  Returns true if mutex was acquired in the constructor, false otherwise.

21.489.2 Constructor & Destructor Documentation

wxMutexLocker::wxMutexLocker (wxMutex & *mutex*)

Constructs a wxMutexLocker object associated with mutex and locks it.

Call IsOk() to check if the mutex was successfully locked.

wxMutexLocker::~wxMutexLocker ( )

Destructor releases the mutex if it was successfully acquired in the ctor.

21.489.3 Member Function Documentation

bool wxMutexLocker::IsOk ( ) const

Returns true if mutex was acquired in the constructor, false otherwise.

21.490 wxNativeFontInfo Class Reference

```
#include <wx/fontutil.h>
```

21.490.1 Detailed Description

wxNativeFontInfo is platform-specific font representation: this class should be considered as an opaque font description only used by the native functions, the user code can only get the objects of this type from somewhere and pass it somewhere else (possibly save them somewhere using ToString() and restore them using FromString())

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- wxNativeFontInfo ()
- wxNativeFontInfo (const wxNativeFontInfo &info)
- ~wxNativeFontInfo ()
- wxNativeFontInfo & operator= (const wxNativeFontInfo &info)
- void Init ()
- void InitFromFont (const wxFont &font)
- int GetPointSize () const
- wxSize GetPixelSize () const
- wxFontStyle GetStyle () const
- wxFontWeight GetWeight () const
- bool GetUnderlined () const
- wxString GetFaceName () const
- wxFontFamily GetFamily () const
- wxFontEncoding GetEncoding () const
- void SetPointSize (int pointsize)
- void SetPixelSize (const wxSize &pixelSize)
- void SetStyle (wxFontStyle style)
- void SetWeight (wxFontWeight weight)
- void SetUnderlined (bool underlined)
- bool SetFaceName (const wxString &facename)
- void SetFamily (wxFontFamily family)
- void SetEncoding (wxFontEncoding encoding)
- void SetFaceName (const wxArrayString &facenames)
- bool FromString (const wxString &s)
- wxString ToString () const
- bool FromUserString (const wxString &s)
- wxString ToUserString () const

21.490.2 Constructor & Destructor Documentation

wxNativeFontInfo::wxNativeFontInfo ( )

wxNativeFontInfo::wxNativeFontInfo (const **wxNativeFontInfo** & *info*)

wxNativeFontInfo::~wxNativeFontInfo ( )

21.490.3 Member Function Documentation

bool wxNativeFontInfo::FromString (const **wxString** & *s*)

bool wxNativeFontInfo::FromUserString (const **wxString** & *s*)

wxFontEncoding wxNativeFontInfo::GetEncoding ( ) const

wxString wxNativeFontInfo::GetFaceName ( ) const

wxFontFamily wxNativeFontInfo::GetFamily ( ) const

wxSize wxNativeFontInfo::GetPixelSize ( ) const

int wxNativeFontInfo::GetPointSize ( ) const

wxFontStyle wxNativeFontInfo::GetStyle ( ) const

bool wxNativeFontInfo::GetUnderlined ( ) const

wxFontWeight wxNativeFontInfo::GetWeight ( ) const

void wxNativeFontInfo::Init ( )

void wxNativeFontInfo::InitFromFont (const **wxFont** & *font*)

wxNativeFontInfo& wxNativeFontInfo::operator= (const **wxNativeFontInfo** & *info*)

void wxNativeFontInfo::SetEncoding (**wxFontEncoding** *encoding*)

bool wxNativeFontInfo::SetFaceName (const **wxString** & *facename*)

void wxNativeFontInfo::SetFaceName (const **wxArrayString** & *facenames*)

void wxNativeFontInfo::SetFamily (**wxFontFamily** *family*)

void wxNativeFontInfo::SetPixelSize (const **wxSize** & *pixelSize*)

void wxNativeFontInfo::SetPointSize (int *pointsize*)

void wxNativeFontInfo::SetStyle (**wxFontStyle** *style*)

void wxNativeFontInfo::SetUnderlined (bool *underlined*)

void wxNativeFontInfo::SetWeight (**wxFontWeight** *weight*)

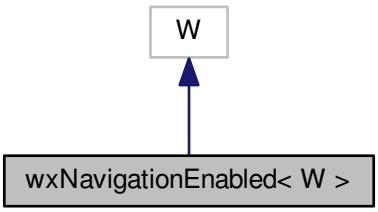
wxString wxNativeFontInfo::ToString ( ) const

wxString wxNativeFontInfo::ToUserString ( ) const

21.491 wxNavigationEnabled< W > Class Template Reference

`#include <wx/containr.h>`

Inheritance diagram for wxNavigationEnabled< W >:



21.491.1 Detailed Description

template<class W>class wxNavigationEnabled< W >

A helper class implementing TAB navigation among the window children.

This class contains the functionality needed to correctly implement TAB navigation among the children of the window. Its exact contents is not important and is intentionally not documented as the only way to use this class is to inherit from it instead of inheriting from the usual base class directly. For example, if some class needs to inherit from wxControl but contains multiple sub-windows and needs to support keyboard navigation, it is enough to declare it in the following way:

```
class MyControlWithSubChildren :
    public wxNavigationEnabled<wxControl>
{
public:
    // Default constructor is implemented in the same way as always.
    MyControlWithSubChildren() { }

    // Non-default constructor can't use wxControl ctor any more as
    // wxControl is not its direct base class, but it can use Create().
    MyControlWithSubChildren(wxWindow *parent, wxWindowID winid)
    {
        wxControl::Create(parent, winid);

        // More creation code...
    }

    // Everything else as usual ...
};
```

Library: wxCore

Since

2.9.3

Public Types

- typedef W BaseWindowClass

  The name of the real base window class that this class derives from.

Public Member Functions

- wxNavigationEnabled ()

  Default constructor.

21.491.2 Member Typedef Documentation

template<class W > typedef W wxNavigationEnabled< W >::BaseWindowClass

The name of the real base window class that this class derives from.

21.491.3 Constructor & Destructor Documentation

template<class W > wxNavigationEnabled< W >::wxNavigationEnabled ( )

Default constructor.

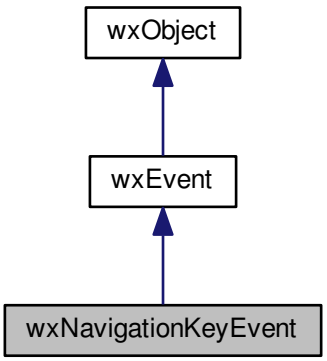
This class provides only the default constructor as it's not possible, in general, to provide all the constructors of the real base class BaseWindowClass.

This is however not usually a problem for wxWindow-derived classes as, by convention, they always define a Create() method such that calling it on an object initialized using the default constructor is equivalent to using a non-default constructor directly. So the classes inheriting from wxNavigationEnabled<W> should simply call W::↩ Create() in their constructors.

21.492 wxNavigationKeyEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxNavigationKeyEvent:



21.492.1 Detailed Description

This event class contains information about navigation events, generated by navigation keys such as tab and page down.

This event is mainly used by wxWidgets implementations. A wxNavigationKeyEvent handler is automatically provided by wxWidgets when you enable keyboard navigation inside a window by inheriting it from wxNavigationEnabled<>.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxNavigationKeyEvent& event)

Event macros:

- EVT_NAVIGATION_KEY(func): Process a navigation key event.

Library: wxCore

Category: Events

See also

wxWindow::Navigate, wxWindow::NavigateIn

Public Types

- enum wxNavigationKeyEventFlags {
  IsBackward = 0x0000,
  IsForward = 0x0001,
  WinChange = 0x0002,
  FromTab = 0x0004 }

  Flags which can be used with wxNavigationKeyEvent.

Public Member Functions

- wxNavigationKeyEvent ()
- wxNavigationKeyEvent (const wxNavigationKeyEvent &event)
- wxWindow * GetCurrentFocus () const

  Returns the child that has the focus, or NULL.
- bool GetDirection () const

  Returns true if the navigation was in the forward direction.
- bool IsFromTab () const

  Returns true if the navigation event was from a tab key.
- bool IsWindowChange () const

  Returns true if the navigation event represents a window change (for example, from Ctrl-Page Down in a notebook).
- void SetCurrentFocus (wxWindow *currentFocus)

  Sets the current focus window member.
- void SetDirection (bool direction)

  Sets the direction to forward if direction is true, or backward if false.
- void SetFlags (long flags)

Sets the flags for this event.

- void SetFromTab (bool fromTab)

  Marks the navigation event as from a tab key.

- void SetWindowChange (bool windowChange)

  Marks the event as a window change event.

Additional Inherited Members

21.492.2 Member Enumeration Documentation

enum wxNavigationKeyEvent::wxNavigationKeyEventFlags

Flags which can be used with wxNavigationKeyEvent.

Enumerator

IsBackward

IsForward

WinChange

FromTab

21.492.3 Constructor & Destructor Documentation

wxNavigationKeyEvent::wxNavigationKeyEvent ( )

wxNavigationKeyEvent::wxNavigationKeyEvent (const wxNavigationKeyEvent & *event*)

21.492.4 Member Function Documentation

wxWindow* wxNavigationKeyEvent::GetCurrentFocus ( ) const

Returns the child that has the focus, or NULL.

bool wxNavigationKeyEvent::GetDirection ( ) const

Returns true if the navigation was in the forward direction.

bool wxNavigationKeyEvent::IsFromTab ( ) const

Returns true if the navigation event was from a tab key.

This is required for proper navigation over radio buttons.

bool wxNavigationKeyEvent::IsWindowChange ( ) const

Returns true if the navigation event represents a window change (for example, from Ctrl-Page Down in a notebook).

void wxNavigationKeyEvent::SetCurrentFocus (wxWindow * *currentFocus*)

Sets the current focus window member.

void wxNavigationKeyEvent::SetDirection (bool *direction*)

Sets the direction to forward if *direction* is true, or backward if false.

void wxNavigationKeyEvent::SetFlags (long *flags*)

Sets the flags for this event.

The *flags* can be a combination of the wxNavigationKeyEvent::wxNavigationKeyEventFlags values.

void wxNavigationKeyEvent::SetFromTab (bool *fromTab*)

Marks the navigation event as from a tab key.

void wxNavigationKeyEvent::SetWindowChange (bool *windowChange*)

Marks the event as a window change event.

21.493 wxNode< T > Class Template Reference

`#include <wx/list.h>`

21.493.1 Detailed Description

template<typename T>class wxNode< T >

wxNode<T> is the node structure used in linked lists (see wxList) and derived classes.

You should never use wxNode<T> class directly, however, because it works with untyped (`void *`) data and this is unsafe. Use wxNode<T>-derived classes which are automatically defined by WX_DECLARE_LIST and WX_DEFINE_LIST macros instead as described in wxList documentation (see example there).

Also note that although there is a class called wxNode, it is defined for backwards compatibility only and usage of this class is strongly deprecated.

In the documentation below, the type `T` should be thought of as a "template" parameter: this is the type of data stored in the linked list or, in other words, the first argument of WX_DECLARE_LIST macro. Also, wxNode is written as wxNodeT even though it isn't really a template class – but it helps to think of it as if it were.

Template Parameters

T	The type stored in the wxNode.

Library: wxBase

Category: Data Structures

See also

wxList<T>, wxHashTable

Public Member Functions

- T * GetData () const

Retrieves the client data pointer associated with the node.

- wxNode< T > * GetNext () const

  Retrieves the next node or NULL if this node is the last one.

- wxNode< T > * GetPrevious ()

  Retrieves the previous node or NULL if this node is the first one in the list.

- int IndexOf ()

  Returns the zero-based index of this node within the list.

- void SetData (T *data)

  Sets the data associated with the node (usually the pointer will have been set when the node was created).

21.493.2 Member Function Documentation

template<typename T > T* wxNode< T >::GetData ( ) const

Retrieves the client data pointer associated with the node.

template<typename T > wxNode<T>* wxNode< T >::GetNext ( ) const

Retrieves the next node or NULL if this node is the last one.

template<typename T > wxNode<T>* wxNode< T >::GetPrevious ( )

Retrieves the previous node or NULL if this node is the first one in the list.

template<typename T > int wxNode< T >::IndexOf ( )

Returns the zero-based index of this node within the list.

The return value will be `wxNOT_FOUND` if the node has not been added to a list yet.

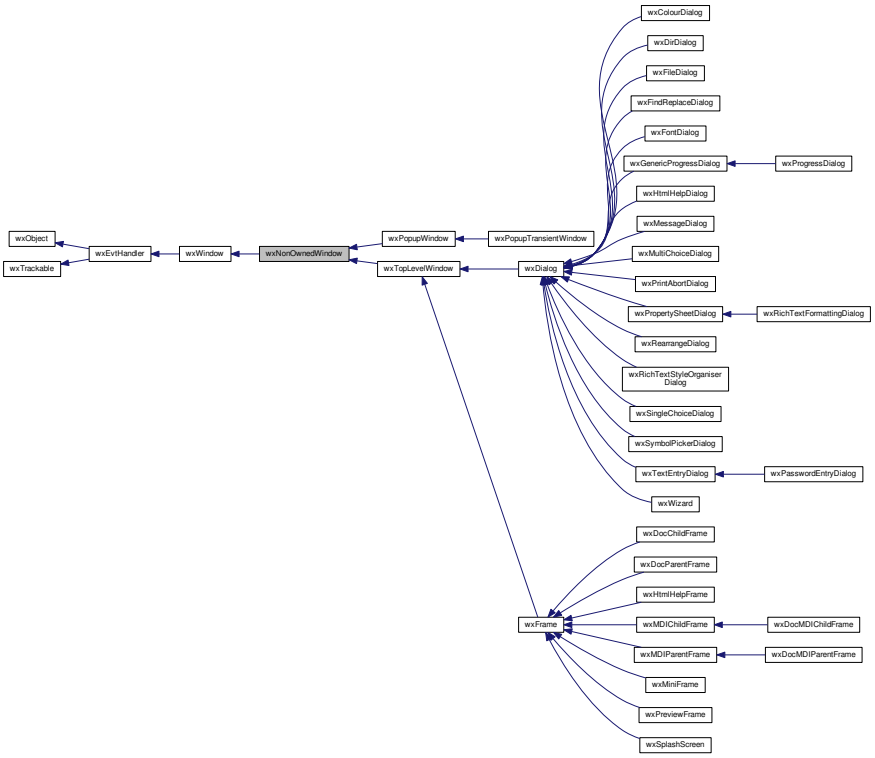
template<typename T > void wxNode< T >::SetData (T * *data*)

Sets the data associated with the node (usually the pointer will have been set when the node was created).

21.494 wxNonOwnedWindow Class Reference

`#include <wx/nonownedwnd.h>`

Inheritance diagram for wxNonOwnedWindow:



21.494.1 Detailed Description

Common base class for all non-child windows.

This is the common base class of wxTopLevelWindow and wxPopupWindow and is not used directly.

Currently the only additional functionality it provides, compared to base wxWindow class, is the ability to set the window shape.

Since

2.9.3

Public Member Functions

- bool SetShape (const wxRegion &region)

  If the platform supports it, sets the shape of the window to that depicted by region.
- bool SetShape (const wxGraphicsPath &path)

  Set the window shape to the given path.

Additional Inherited Members

21.494.2 Member Function Documentation

bool wxNonOwnedWindow::SetShape (const wxRegion & *region*)

If the platform supports it, sets the shape of the window to that depicted by *region*.

The system will not display or respond to any mouse event for the pixels that lie outside of the region. To reset the window to the normal rectangular shape simply call SetShape() again with an empty wxRegion. Returns true if the operation is successful.

This method is available in this class only since wxWidgets 2.9.3, previous versions only provided it in wxTopLevelWindow.

bool wxNonOwnedWindow::SetShape (const wxGraphicsPath & *path*)

Set the window shape to the given path.

Set the window shape to the interior of the given path and also draw the window border along the specified path.

For example, to make a clock-like circular window you could use

```
wxSize size = GetSize();
wxGraphicsPath
    path = wxGraphicsRenderer::GetDefaultRenderer()->
      CreatePath();
path.AddCircle(size.x/2, size.y/2, 30);
SetShape(path);
```

As the overload above, this method is not guaranteed to work on all platforms but currently does work in wxMSW, wxOSX/Cocoa and wxGTK (with the appropriate but almost always present X11 extensions) ports.

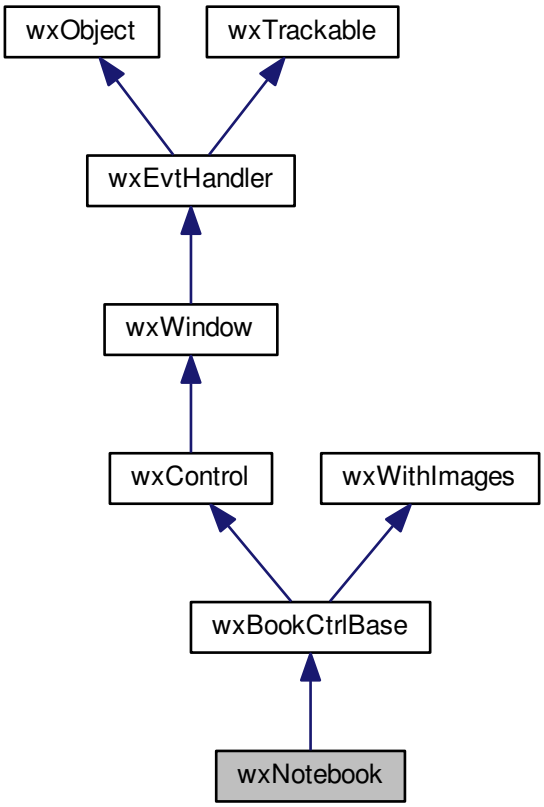
Since

2.9.3

21.495 wxNotebook Class Reference

```
#include <wx/notebook.h>
```

Inheritance diagram for wxNotebook:



21.495.1 Detailed Description

This class represents a notebook control, which manages multiple windows with associated tabs.

To use the class, create a wxNotebook object and call wxNotebook::AddPage or wxNotebook::InsertPage, passing a window to be used as the page. Do not explicitly delete the window for a page that is currently managed by wxNotebook.

wxNotebookPage is a typedef for wxWindow.

Styles

This class supports the following styles:

- wxNB_TOP: Place tabs on the top side.
- wxNB_LEFT: Place tabs on the left side.
- wxNB_RIGHT: Place tabs on the right side.
- wxNB_BOTTOM: Place tabs under instead of above the notebook pages.
- wxNB_FIXEDWIDTH: (Windows only) All tabs will have same width.

- wxNB_MULTILINE: (Windows only) There can be several rows of tabs.
- wxNB_NOPAGETHEME: (Windows only) Display a solid colour on notebook pages, and not a gradient, which can reduce performance.
- wxNB_FLAT: (Windows CE only) Show tabs in a flat style.

The styles wxNB_LEFT, RIGHT and BOTTOM are not supported under Microsoft Windows XP when using visual themes.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxBookCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_NOTEBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_NOTEBOOK_PAGE_CHANGED` event.
- EVT_NOTEBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_NOTEBOOK_PAGE_CHANGING` event. This event can be vetoed.

21.495.2 Page backgrounds

On Windows XP, the default theme paints a gradient on the notebook's pages. If you wish to suppress this theme, for aesthetic or performance reasons, there are three ways of doing it. You can use `wxNB_NOPAGETHEME` to disable themed drawing for a particular notebook, you can call wxSystemOptions::SetOption to disable it for the whole application, or you can disable it for individual pages by using SetBackgroundColour().

To disable themed pages globally:

```
wxSystemOptions::SetOption("msw.notebook.themed-background", 0);
```

Set the value to 1 to enable it again. To give a single page a solid background that more or less fits in with the overall theme, use:

```
wxColour col = notebook->GetThemeBackgroundColour();
if (col.IsOk())
{
    page->SetBackgroundColour(col);
}
```

On platforms other than Windows, or if the application is not using Windows themes, GetThemeBackgroundColour() will return an uninitialised colour object, and the above code will therefore work on all platforms.

Library: wxCore

Category: Book Controls

See also

wxBookCtrl, wxBookCtrlEvent, wxImageList, Notebook Sample

Public Member Functions

- wxNotebook ()

  Constructs a notebook control.
- wxNotebook (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxNotebookNameStr)

  Constructs a notebook control.
- virtual ~wxNotebook ()

  Destroys the wxNotebook object.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxNotebookNameStr)

  Creates a notebook control.
- virtual int GetRowCount () const

  Returns the number of rows in the notebook control.
- virtual wxColour GetThemeBackgroundColour () const

  If running under Windows and themes are enabled for the application, this function returns a suitable colour for painting the background of a notebook page, and can be passed to SetBackgroundColour().
- void OnSelChange (wxBookCtrlEvent &event)

  An event handler function, called when the page selection is changed.
- virtual void SetPadding (const wxSize &padding)

  Sets the amount of space around each page's icon and label, in pixels.
- virtual int GetPageImage (size_t nPage) const

  Returns the image index for the given page.
- virtual bool SetPageImage (size_t page, int image)

  Sets the image index for the given page.
- virtual wxString GetPageText (size_t nPage) const

  Returns the string for the given page.
- virtual bool SetPageText (size_t page, const wxString &text)

  Sets the text for the given page.
- virtual int GetSelection () const

  Returns the currently selected page, or `wxNOT_FOUND` if none was selected.
- virtual int SetSelection (size_t page)

  Sets the selection to the given page, returning the previous selection.
- virtual int ChangeSelection (size_t page)

  Changes the selection to the given page, returning the previous selection.
- virtual bool InsertPage (size_t index, wxWindow *page, const wxString &text, bool select=false, int image↩ Id=NO_IMAGE)

  Inserts a new page at the specified position.

Additional Inherited Members

21.495.3 Constructor & Destructor Documentation

wxNotebook::wxNotebook ( )

Constructs a notebook control.

wxNotebook::wxNotebook (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxNotebookNameStr`)

Constructs a notebook control.

Note that sometimes you can reduce flicker by passing the wxCLIP_CHILDREN window style.

Parameters

parent	The parent window. Must be non-NULL.
id	The window identifier.
pos	The window position.
size	The window size.
style	The window style. See wxNotebook.
name	The name of the control.

virtual wxNotebook::~wxNotebook ( ) [virtual]

Destroys the wxNotebook object.

21.495.4 Member Function Documentation

virtual int wxNotebook::ChangeSelection (size_t *page*) [virtual]

Changes the selection to the given page, returning the previous selection.

This function behaves as SetSelection() but does *not* generate the page changing events.

See User Generated Events vs Programmatically Generated Events for more information.

Implements wxBookCtrlBase.

bool wxNotebook::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxNotebookNameStr)

Creates a notebook control.

See wxNotebook() for a description of the parameters.

virtual int wxNotebook::GetPageImage (size_t *nPage*) const [virtual]

Returns the image index for the given page.

Implements wxBookCtrlBase.

virtual wxString wxNotebook::GetPageText (size_t *nPage*) const [virtual]

Returns the string for the given page.

Implements wxBookCtrlBase.

virtual int wxNotebook::GetRowCount ( ) const [virtual]

Returns the number of rows in the notebook control.

virtual int wxNotebook::GetSelection ( ) const [virtual]

Returns the currently selected page, or wxNOT_FOUND if none was selected.

Note that this method may return either the previously or newly selected page when called from the EVT_BOOKCTRL_PAGE_CHANGED handler depending on the platform and so wxBookCtrlEvent::GetSelection should be used instead in this case.

Implements wxBookCtrlBase.

virtual wxColour wxNotebook::GetThemeBackgroundColour ( ) const `[virtual]`

If running under Windows and themes are enabled for the application, this function returns a suitable colour for painting the background of a notebook page, and can be passed to SetBackgroundColour().

Otherwise, an uninitialised colour will be returned.

virtual bool wxNotebook::InsertPage (size_t *index*, wxWindow * *page*, const wxString & *text*, bool *select* = `false`, int *imageId* = NO_IMAGE) `[virtual]`

Inserts a new page at the specified position.

Parameters

index	Specifies the position for the new page.
page	Specifies the new page.
text	Specifies the text for the new page.
select	Specifies whether the page should be selected.
imageId	Specifies the optional image index for the new page.

Returns

true if successful, false otherwise.

Remarks

Do not delete the page, it will be deleted by the book control.

See also

AddPage()

Implements wxBookCtrlBase.

void wxNotebook::OnSelChange (wxBookCtrlEvent & *event*)

An event handler function, called when the page selection is changed.

See also

wxBookCtrlEvent

virtual void wxNotebook::SetPadding (const wxSize & *padding*) `[virtual]`

Sets the amount of space around each page's icon and label, in pixels.

Note

The vertical padding cannot be changed in wxGTK.

virtual bool wxNotebook::SetPageImage (size_t *page*, int *image*) `[virtual]`

Sets the image index for the given page.

image is an index into the image list which was set with SetImageList().

Implements wxBookCtrlBase.

virtual bool wxNotebook::SetPageText (size_t *page,* const wxString & *text*) `[virtual]`

Sets the text for the given page.

Implements wxBookCtrlBase.

virtual int wxNotebook::SetSelection (size_t *page*) `[virtual]`

Sets the selection to the given page, returning the previous selection.

Notice that the call to this function generates the page changing events, use the ChangeSelection() function if you don't want these events to be generated.

See also

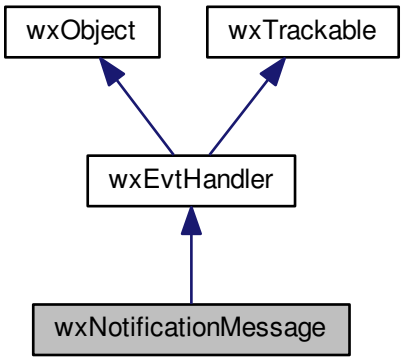
GetSelection()

Implements wxBookCtrlBase.

21.496 wxNotificationMessage Class Reference

`#include <wx/notifmsg.h>`

Inheritance diagram for wxNotificationMessage:



21.496.1 Detailed Description

This class allows to show the user a message non intrusively.

Currently it is implemented natively for Windows and GTK and uses (non-modal) dialogs for the display of the notifications under the other platforms.

Notice that this class is not a window and so doesn't derive from wxWindow.

Library: wxAdvanced

Category: Miscellaneous

Public Types

- enum {
  Timeout_Auto = -1,
  Timeout_Never = 0 }

  Possible values for Show() timeout.

Public Member Functions

- wxNotificationMessage ()

  Default constructor, use SetParent(), SetTitle() and SetMessage() to initialize the object before showing it.
- wxNotificationMessage (const wxString &title, const wxString &message=wxEmptyString, wxWindow *parent=NULL, int flags=wxICON_INFORMATION)

  Create a notification object with the given attributes.
- virtual ~wxNotificationMessage ()

  Destructor does not hide the notification.
- virtual bool Close ()

  Hides the notification.
- void SetFlags (int flags)

  This parameter can be currently used to specify the icon to show in the notification.
- void SetMessage (const wxString &message)

  Set the main text of the notification.
- void SetParent (wxWindow *parent)

  Set the parent for this notification: the notification will be associated with the top level parent of this window or, if this method is not called, with the main application window by default.
- void SetTitle (const wxString &title)

  Set the title, it must be a concise string (not more than 64 characters), use SetMessage() to give the user more details.
- virtual bool Show (int timeout=Timeout_Auto)

  Show the notification to the user and hides it after timeout seconds are elapsed.

Additional Inherited Members

21.496.2 Member Enumeration Documentation

anonymous enum

Possible values for Show() timeout.

Enumerator

Timeout_Auto Notification will be hidden automatically.

Timeout_Never Notification will never time out.

21.496.3 Constructor & Destructor Documentation

wxNotificationMessage::wxNotificationMessage ( )

Default constructor, use SetParent(), SetTitle() and SetMessage() to initialize the object before showing it.

wxNotificationMessage::wxNotificationMessage (const wxString & *title*, const wxString & *message* = wxEmptyString, wxWindow * *parent* = NULL, int *flags* = wxICON_INFORMATION)

Create a notification object with the given attributes.

See SetTitle(), SetMessage(), SetParent() and SetFlags() for the description of the corresponding parameters.

virtual wxNotificationMessage::~wxNotificationMessage ( ) [virtual]

Destructor does not hide the notification.

The notification can continue to be shown even after the C++ object was destroyed, call Close() explicitly if it needs to be hidden.

21.496.4 Member Function Documentation

virtual bool wxNotificationMessage::Close ( ) [virtual]

Hides the notification.

Returns true if it was hidden or false if it couldn't be done (e.g. on some systems automatically hidden notifications can't be hidden manually).

void wxNotificationMessage::SetFlags (int *flags*)

This parameter can be currently used to specify the icon to show in the notification.

Valid values are wxICON_INFORMATION, wxICON_WARNING and wxICON_ERROR (notice that wxICON_QUESTION is not allowed here). Some implementations of this class may not support the icons.

void wxNotificationMessage::SetMessage (const wxString & *message*)

Set the main text of the notification.

This should be a more detailed description than the title but still limited to reasonable length (not more than 256 characters).

void wxNotificationMessage::SetParent (wxWindow * *parent*)

Set the parent for this notification: the notification will be associated with the top level parent of this window or, if this method is not called, with the main application window by default.

void wxNotificationMessage::SetTitle (const wxString & *title*)

Set the title, it must be a concise string (not more than 64 characters), use SetMessage() to give the user more details.

virtual bool wxNotificationMessage::Show (int *timeout* = Timeout_Auto) [virtual]

Show the notification to the user and hides it after *timeout* seconds are elapsed.

Special values Timeout_Auto and Timeout_Never can be used here, notice that you shouldn't rely on *timeout* being exactly respected because the current platform may only support default timeout value and also because the user may be able to close the notification.

Note

When using native notifications in wxGTK, the timeout is ignored for the notifications with `wxICON_WARNING` or `wxICON_ERROR` flags, they always remain shown unless they're explicitly hidden by the user, i.e. behave as if Timeout_Auto were given.

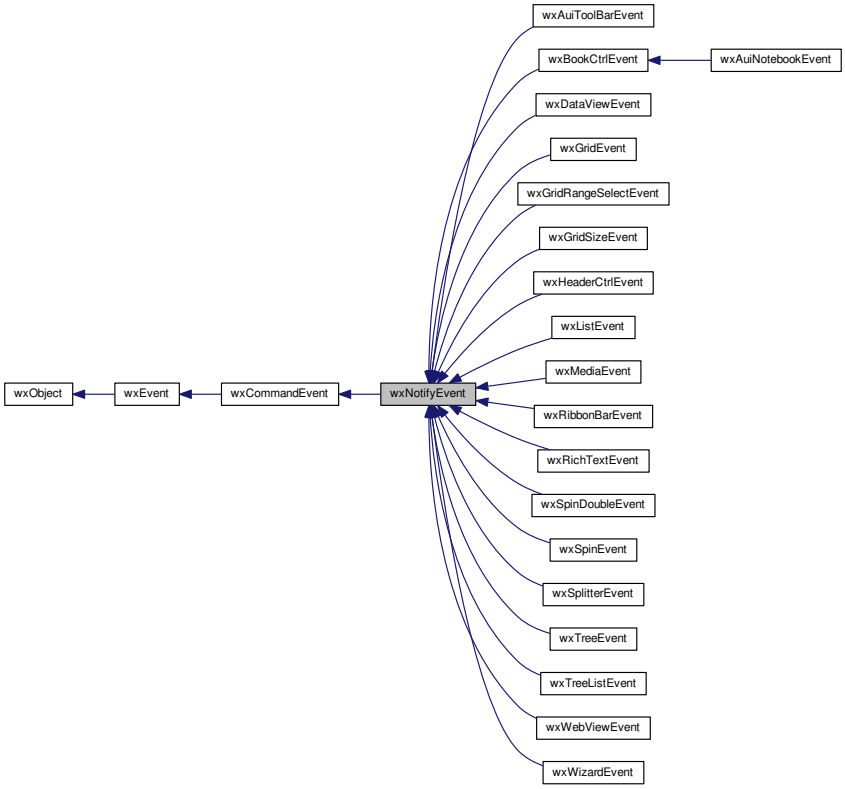
Returns

false if an error occurred.

21.497 wxNotifyEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxNotifyEvent:



21.497.1 Detailed Description

This class is not used by the event handlers by itself, but is a base class for other event classes (such as wxBookCtrlEvent).

It (or an object of a derived class) is sent when the controls state is being changed and allows the program to wxNotifyEvent::Veto() this change if it wants to prevent it from happening.

Library: wxCore

Category: Events

See also

wxBookCtrlEvent

Public Member Functions

- wxNotifyEvent (wxEventType eventType=wxEVT_NULL, int id=0)

  Constructor (used internally by wxWidgets only).
- void Allow ()

  This is the opposite of Veto(): it explicitly allows the event to be processed.
- bool IsAllowed () const

  Returns true if the change is allowed (Veto() hasn't been called) or false otherwise (if it was).
- void Veto ()

  Prevents the change announced by this event from happening.

Additional Inherited Members

21.497.2 Constructor & Destructor Documentation

wxNotifyEvent::wxNotifyEvent (wxEventType *eventType* = wxEVT_NULL, int *id* = 0)

Constructor (used internally by wxWidgets only).

21.497.3 Member Function Documentation

void wxNotifyEvent::Allow ( )

This is the opposite of Veto(): it explicitly allows the event to be processed.

For most events it is not necessary to call this method as the events are allowed anyhow but some are forbidden by default (this will be mentioned in the corresponding event description).

bool wxNotifyEvent::IsAllowed ( ) const

Returns true if the change is allowed (Veto() hasn't been called) or false otherwise (if it was).

void wxNotifyEvent::Veto ( )

Prevents the change announced by this event from happening.

It is in general a good idea to notify the user about the reasons for vetoing the change because otherwise the applications behaviour (which just refuses to do what the user wants) might be quite surprising.

21.498 wxNumberFormatter Class Reference

```
#include <wx/numformatter.h>
```

21.498.1 Detailed Description

Helper class for formatting and parsing numbers with thousands separators.

This class contains only static functions, so users must not create instances but directly call the member functions.

Since

2.9.2

Library: wxBase

Public Types

- enum Style {
  Style_None = 0x00,
  Style_WithThousandsSep = 0x01,
  Style_NoTrailingZeroes = 0x02 }
  Bit masks used with ToString().

Static Public Member Functions

- static wxString ToString (long val, int flags=Style_WithThousandsSep)
  Returns string representation of an integer number.
- static wxString ToString (double val, int precision, int flags=Style_WithThousandsSep)
  Returns string representation of a floating point number.
- static wxChar GetDecimalSeparator ()
  Get the decimal separator for the current locale.
- static bool GetThousandsSeparatorIfUsed (wxChar *sep)
  Get the thousands separator if grouping of the digits is used by the current locale.
- static bool FromString (wxString s, long *val)
  Parse a string representation of a number possibly including thousands separators.
- static bool FromString (wxString s, double *val)
  Parse a string representation of a number possibly including thousands separators.

21.498.2 Member Enumeration Documentation

enum wxNumberFormatter::Style

Bit masks used with ToString().

Enumerator

Style_None This flag can be used to indicate absence of any other flags below.

Style_WithThousandsSep If this flag is given, thousands separators will be inserted in the number string representation as defined by the current locale.

Style_NoTrailingZeroes If this flag is given, trailing zeroes in a floating point number string representation will be omitted. If the number is actually integer, the decimal separator will be omitted as well. To give an example, formatting the number `1.23` with precision 5 will normally yield "1.23000" but with this flag it would return "1.23". And formatting `123` with this flag will return just "123" for any precision.

This flag can't be used with ToString() overload taking the integer value.

21.498.3 Member Function Documentation

static bool wxNumberFormatter::FromString (wxString *s,* long * *val*) [static]

Parse a string representation of a number possibly including thousands separators.

These functions parse number representation in the current locale. On success they return true and store the result at the location pointed to by *val* (which can't be NULL), otherwise false is returned.

See also

wxString::ToLong(), wxString::ToDouble()

static bool wxNumberFormatter::FromString (wxString *s,* double * *val*) [static]

Parse a string representation of a number possibly including thousands separators.

These functions parse number representation in the current locale. On success they return true and store the result at the location pointed to by *val* (which can't be NULL), otherwise false is returned.

See also

wxString::ToLong(), wxString::ToDouble()

static wxChar wxNumberFormatter::GetDecimalSeparator ( ) [static]

Get the decimal separator for the current locale.

Decimal separators is always defined and we fall back to returning '.' in case of an error.

static bool wxNumberFormatter::GetThousandsSeparatorIfUsed (wxChar * *sep*) [static]

Get the thousands separator if grouping of the digits is used by the current locale.

The value returned in *sep* should be only used if the function returns true, otherwise no thousands separator should be used at all.

Parameters

sep	Points to the variable receiving the thousands separator character if it is used by the current locale. May be NULL if only the function return value is needed.

static wxString wxNumberFormatter::ToString (long *val,* int *flags* = Style_WithThousandsSep) [static]

Returns string representation of an integer number.

By default, the string will use thousands separators if appropriate for the current locale. This can be avoided by passing Style_None as *flags* in which case the call to the function has exactly the same effect as `wxString::Format("%ld", val)`.

Notice that calling ToString() with a value of type `int` and non-default flags results in ambiguity between this overload and the one below. To resolve it, you need to cast the value to `long`.

Parameters

val	The variable to convert to a string.
flags	Combination of values from the Style enumeration (except for Style_NoTrailingZeroes which can't be used with this overload).

static wxString wxNumberFormatter::ToString (double *val,* int *precision,* int *flags* = Style_WithThousandsSep) `[static]`

Returns string representation of a floating point number.

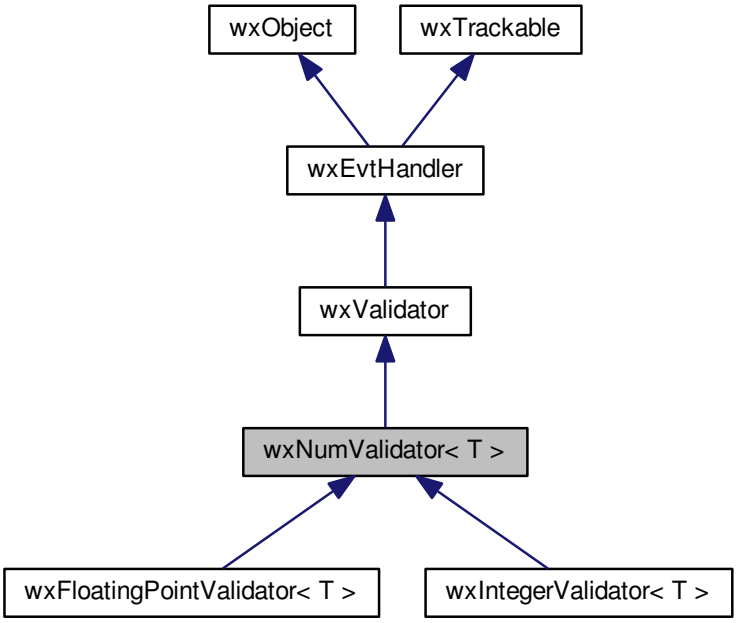
Parameters

val	The variable to convert to a string.
precision	Number of decimals to write in formatted string.
flags	Combination of values from the Style enumeration.

21.499 wxNumValidator< T > Class Template Reference

`#include <wx/valnum.h>`

Inheritance diagram for wxNumValidator< T >:



21.499.1 Detailed Description

template<typename T>class wxNumValidator< T >

wxNumValidator is the common base class for numeric validator classes.

This class is never used directly, but only as a base class for wxIntegerValidator and wxFloatingPointValidator.

Template Parameters

T	Type of the values used with this validator.

Category: Validators

Since

2.9.2

Public Types

- typedef T ValueType

  Type of the values this validator is used with.

Public Member Functions

- void SetMin (ValueType min)

  Sets the minimal value accepted by the validator.
- void SetMax (ValueType max)

  Sets the maximal value accepted by the validator.
- void SetRange (ValueType min, ValueType max)

  Sets both minimal and maximal values accepted by the validator.
- void SetStyle (int style)

  Change the validator style.
- virtual bool TransferToWindow ()

  Override base class method to format the control contents.
- virtual bool TransferFromWindow ()

  Override base class method to validate the control contents.

Additional Inherited Members

21.499.2 Member Typedef Documentation

template<typename T > typedef T wxNumValidator< T >::ValueType

Type of the values this validator is used with.

21.499.3 Member Function Documentation

template<typename T > void wxNumValidator< T >::SetMax (ValueType *max*)

Sets the maximal value accepted by the validator.

This value is inclusive, i.e. the value equal to *max* is accepted.

template<typename T > void wxNumValidator< T >::SetMin (ValueType *min*)

Sets the minimal value accepted by the validator.

This value is inclusive, i.e. the value equal to *min* is accepted.

template<typename T > void wxNumValidator< T >::SetRange (ValueType *min,* ValueType *max*)

Sets both minimal and maximal values accepted by the validator.

Calling this is equivalent to calling both SetMin() and SetMax().

template<typename T > void wxNumValidator< T >::SetStyle (int *style*)

Change the validator style.

Can be used to change the style of the validator after its creation. The *style* parameter must be a combination of the values from wxNumValidatorStyle enum.

template<typename T > virtual bool wxNumValidator< T >::TransferFromWindow ( ) [virtual]

Override base class method to validate the control contents.

This method is called to check the correctness of user input and fill the associated variable with the controls numeric value. It returns false if it is not a number in the configured range or if the control contents is empty for a validator without wxNUM_VAL_ZERO_AS_BLANK style.

It does nothing if there is no associated variable.

Reimplemented from wxValidator.

template<typename T > virtual bool wxNumValidator< T >::TransferToWindow ( ) [virtual]

Override base class method to format the control contents.

This method is called when the associated window is shown and it fills it with the contents of the associated variable, if any, formatted according to the validator style.

It does nothing if there is no associated variable.

Reimplemented from wxValidator.

21.500 wxObject Class Reference

`#include <wx/object.h>`

Inherited by wxAcceleratorTable, wxAccessible, wxAnimation, wxArchiveClassFactory, wxArchiveEntry, wxArtProvider, wxAutomationObject, wxBitmapHandler, wxClient, wxClipboard, wxColour, wxColourData, wxCommand, wxCommandProcessor, wxConfigBase, wxConnection, wxConnectionBase, wxContextHelp, wxDataViewIconText, wxDataViewRenderer, wxDC, wxDDEClient, wxDocTemplate, wxDragImage, wxEncodingConverter, wxEvent, wxEvtHandler, wxFileHistory, wxFileSystem, wxFileSystemHandler, wxFilterClassFactory, wxFindReplaceData, wxFontData, wxFSFile, wxGDIObject, wxGLContext, wxGraphicsObject, wxGraphicsRenderer, wxGridTableBase, wxHashTable, wxHelpControllerBase, wxHtmlCell, wxHtmlDCRenderer, wxHtmlEasyPrinting, wxHtmlFilter, wxHtmlHelpData, wxHtmlLinkInfo, wxHtmlTagHandler, wxImage, wxImageHandler, wxImageList, wxIndividualLayoutConstraint, wxJoystick, wxLayoutAlgorithm, wxLayoutConstraints, wxListItem, wxMask, wxMenuItem, wxMetafile, wxModule, wxPageSetupDialog, wxPageSetupDialogData, wxPGCell, wxPGEditor, wxPGProperty, wxPrintData, wxPrintDialog, wxPrintDialogData, wxPrinter, wxPrintout, wxPrintPreview, wxQuantize, wxRegionIterator, wxRichTextAction, wxRichTextDrawingContext, wxRichTextDrawingHandler, wxRichTextFieldType, wxRichTextFileHandler, wxRichTextFontTable, wxRichTextFormattingDialogFactory, wxRichTextHeaderFooterData, wxRichTextImageBlock, wxRichTextObject, wxRichTextPrinting, wxRichTextProperties, wxRichTextRenderer, wxRichTextStyleDefinition, wxRichTextStyleSheet, wxSizer, wxSizerItem, wxSockAddress, wxSocketBase, wxSound, wxStringTokenizer, wxSystemOptions, wxSystemSettings, wxTCPClient, wxTCPConnection, wxTCPServer, wxToolBarToolBase, wxToolTip, wxURI, wxVariant, wxWebViewFactory, wxXmlDocument, wxXmlResource, and wxXmlResourceHandler.

21.500.1 Detailed Description

This is the root class of many of the wxWidgets classes.

It declares a virtual destructor which ensures that destructors get called for all derived class objects where necessary.

wxObject is the hub of a dynamic object creation scheme, enabling a program to create instances of a class only knowing its string class name, and to query the class hierarchy.

The class contains optional debugging versions of **new** and **delete**, which can help trace memory allocation and deallocation problems.

wxObject can be used to implement reference counted objects, such as wxPen, wxBitmap and others (see this list). See wxRefCounter and Reference Counting for more info about reference counting.

Library: wxBase

Category: Runtime Type Information (RTTI)

See also

wxClassInfo, Debugging, Reference Counting, wxObjectDataRef, wxObjectDataPtr<T>

Public Member Functions

- wxObject ()

  Default ctor; initializes to NULL the internal reference data.
- wxObject (const wxObject &other)

  Copy ctor.
- virtual ~wxObject ()

  Destructor.
- virtual wxClassInfo * GetClassInfo () const

  This virtual function is redefined for every class that requires run-time type information, when using the wxDECLARE_CLASS macro (or similar).
- wxObjectRefData * GetRefData () const

  Returns the wxObject::m_refData pointer, i.e. the data referenced by this object.
- bool IsKindOf (const wxClassInfo *info) const

  Determines whether this class is a subclass of (or the same class as) the given class.
- bool IsSameAs (const wxObject &obj) const

  Returns true if this object has the same data pointer as obj.
- void Ref (const wxObject &clone)

  Makes this object refer to the data in clone.
- void SetRefData (wxObjectRefData *data)

  Sets the wxObject::m_refData pointer.
- void UnRef ()

  Decrements the reference count in the associated data, and if it is zero, deletes the data.
- void UnShare ()

  This is the same of AllocExclusive() but this method is public.
- void operator delete (void *buf)

  *The delete operator is defined for debugging versions of the library only, when the identifier **WXDEBUG** is defined.*
- void * operator new (size_t size, const wxString &filename=NULL, int lineNum=0)

  *The new operator is defined for debugging versions of the library only, when the identifier **WXDEBUG** is defined.*

Protected Member Functions

- void AllocExclusive ()

  Ensure that this object's data is not shared with any other object.
- virtual wxObjectRefData * CreateRefData () const

  Creates a new instance of the wxObjectRefData-derived class specific to this object and returns it.
- virtual wxObjectRefData * CloneRefData (const wxObjectRefData *data) const

  Creates a new instance of the wxObjectRefData-derived class specific to this object and initializes it copying data.

Protected Attributes

- wxObjectRefData * m_refData

  Pointer to an object which is the object's reference-counted data.

21.500.2 Constructor & Destructor Documentation

wxObject::wxObject ( )

Default ctor; initializes to NULL the internal reference data.

wxObject::wxObject (const wxObject & *other*)

Copy ctor.

Sets the internal wxObject::m_refData pointer to point to the same instance of the wxObjectRefData-derived class pointed by `other` and increments the refcount of wxObject::m_refData.

virtual wxObject::~wxObject ( ) [virtual]

Destructor.

Performs dereferencing, for those objects that use reference counting.

21.500.3 Member Function Documentation

void wxObject::AllocExclusive ( ) [protected]

Ensure that this object's data is not shared with any other object.

If we have no data, it is created using CreateRefData(); if we have shared data (i.e. data with a reference count greater than 1), it is copied using CloneRefData(); otherwise nothing is done (the data is already present and is not shared by other object instances).

If you use this function you should make sure that you override the CreateRefData() and CloneRefData() functions in your class otherwise an assertion will fail at runtime.

virtual wxObjectRefData* wxObject::CloneRefData (const wxObjectRefData * *data*) const [protected], [virtual]

Creates a new instance of the wxObjectRefData-derived class specific to this object and initializes it copying *data*.

This is usually implemented as a one-line call:

```
wxObjectRefData *MyObject::CloneRefData(const wxObjectRefData *data) const
{
    // rely on the MyObjectRefData copy ctor:
    return new MyObjectRefData(*(MyObjectRefData *)data);
}
```

virtual wxObjectRefData* wxObject::CreateRefData ( ) const [protected], [virtual]

Creates a new instance of the wxObjectRefData-derived class specific to this object and returns it.

This is usually implemented as a one-line call:

```
wxObjectRefData *MyObject::CreateRefData() const
{
    return new MyObjectRefData;
}
```

virtual wxClassInfo* wxObject::GetClassInfo ( ) const [virtual]

This virtual function is redefined for every class that requires run-time type information, when using the wxDECLARE_CLASS macro (or similar).

wxObjectRefData* wxObject::GetRefData ( ) const

Returns the wxObject::m_refData pointer, i.e. the data referenced by this object.

See also

Ref(), UnRef(), wxObject::m_refData, SetRefData(), wxObjectRefData

bool wxObject::IsKindOf (const wxClassInfo * *info*) const

Determines whether this class is a subclass of (or the same class as) the given class.

Example:

```
bool tmp = obj->IsKindOf(wxCLASSINFO(wxFrame));
```

Parameters

info	A pointer to a class information object, which may be obtained by using the wxCLASSINFO macro.

Returns

true if the class represented by info is the same class as this one or is derived from it.

bool wxObject::IsSameAs (const wxObject & *obj*) const

Returns true if this object has the same data pointer as *obj*.

Notice that true is returned if the data pointers are NULL in both objects.

This function only does a *shallow* comparison, i.e. it doesn't compare the objects pointed to by the data pointers of these objects.

See also

Reference Counting

void wxObject::operator delete (void * *buf*)

The *delete* operator is defined for debugging versions of the library only, when the identifier **WXDEBUG** is defined.

It takes over memory deallocation, allowing wxDebugContext operations.

void* wxObject::operator new (size_t *size*, const wxString & *filename* = NULL, int *lineNum* = 0)

The *new* operator is defined for debugging versions of the library only, when the identifier **WXDEBUG** is defined.

It takes over memory allocation, allowing wxDebugContext operations.

void wxObject::Ref (const wxObject & *clone*)

Makes this object refer to the data in *clone*.

Parameters

clone	The object to 'clone'.

Remarks

First this function calls UnRef() on itself to decrement (and perhaps free) the data it is currently referring to. It then sets its own wxObject::m_refData to point to that of *clone*, and increments the reference count inside the data.

See also

UnRef(), SetRefData(), GetRefData(), wxObjectRefData

void wxObject::SetRefData (wxObjectRefData * *data*)

Sets the wxObject::m_refData pointer.

See also

Ref(), UnRef(), GetRefData(), wxObjectRefData

void wxObject::UnRef ( )

Decrements the reference count in the associated data, and if it is zero, deletes the data.

The wxObject::m_refData member is set to NULL.

See also

Ref(), SetRefData(), GetRefData(), wxObjectRefData

void wxObject::UnShare ( )

This is the same of AllocExclusive() but this method is public.

21.500.4 Member Data Documentation

wxObjectRefData* wxObject::m_refData [protected]

Pointer to an object which is the object's reference-counted data.

See also

Ref(), UnRef(), SetRefData(), GetRefData(), wxObjectRefData

21.501 wxObjectDataPtr< T > Class Template Reference

`#include <wx/object.h>`

21.501.1 Detailed Description

template<class T>class wxObjectDataPtr< T >

This is an helper template class primarily written to avoid memory leaks because of missing calls to wxRefCounter::DecRef() and wxObjectRefData::DecRef().

Despite the name this template can actually be used as a smart pointer for any class implementing the reference counting interface which only consists of the two methods **T::IncRef()** and **T::DecRef()**.

The difference to wxSharedPtr<T> is that wxObjectDataPtr<T> relies on the reference counting to be in the class pointed to, where instead wxSharedPtr<T> implements the reference counting itself.

Below is an example illustrating how to implement reference counted data using wxRefCounter and wxObjectDataPtr<T> with copy-on-write semantics.

21.501.2 Example

```
class MyCarRefData: public wxRefCounter
{
public:
    MyCarRefData( int price = 0 ) : m_price(price) { }
    MyCarRefData( const MyCarRefData& data ) : m_price(data.m_price) { }

    void SetPrice( int price )  { m_price = price; }
    int GetPrice() const        { return m_price; }

protected:
    int m_price;
};

class MyCar
{
public:
    // initializes this MyCar assigning to the
    // internal data pointer a new instance of MyCarRefData
    MyCar( int price = 0 ) : m_data( new MyCarRefData(price) )
    {
    }

    MyCar& operator =( const MyCar& tocopy )
    {
        // shallow copy: this is just a fast copy of pointers; the real
        // memory-consuming data which typically is stored inside
        // MyCarRefData is not copied here!
        m_data = tocopy.m_data;
        return *this;
    }

    bool operator == ( const MyCar& other ) const
    {
        if (m_data.get() == other.m_data.get())
            return true; // this instance and the 'other' one share the
                         // same MyCarRefData data...
```

```
        return (m_data.GetPrice() == other.m_data.GetPrice());
    }

    void SetPrice( int price )
    {
        // make sure changes to this class do not affect other instances
        // currently sharing our same refcounted data:
        UnShare();

        m_data->SetPrice( price );
    }

    int GetPrice() const
    {
       return m_data->GetPrice();
    }

    wxObjectDataPtr<MyCarRefData> m_data;

protected:
    void UnShare()
    {
        if (m_data->GetRefCount() == 1)
            return;

        m_data.reset( new MyCarRefData( *m_data ) );
    }
};
```

Library: wxBase

Category: Runtime Type Information (RTTI), Smart Pointers

See also

wxObject, wxObjectRefData, Reference Counting, wxSharedPtr<T>, wxScopedPtr<T>, wxWeakRef<T>

Public Member Functions

- wxObjectDataPtr (T *ptr=NULL)

  Constructor.
- wxObjectDataPtr (const wxObjectDataPtr< T > &tocopy)

  This copy constructor increases the count of the reference counted object to which tocopy points and then this class will point to, as well.
- ~wxObjectDataPtr ()

  Decreases the reference count of the object to which this class points.
- T * get () const

  Gets a pointer to the reference counted object to which this class points.
- void reset (T *ptr)

  Reset this class to ptr which points to a reference counted object and calls T::DecRef() on the previously owned object.
- operator unspecified_bool_type () const

  Conversion to a boolean expression (in a variant which is not convertable to anything but a boolean expression).
- T & operator* () const

  Returns a reference to the object.
- T * operator-> () const

  Returns a pointer to the reference counted object to which this class points.

- wxObjectDataPtr< T > & operator= (const wxObjectDataPtr< T > &tocopy)

  Assignment operator.
- wxObjectDataPtr< T > & operator= (T *ptr)

  Assignment operator.

21.501.3 Constructor & Destructor Documentation

template<class T > wxObjectDataPtr< T >::wxObjectDataPtr (T * *ptr* = NULL)

Constructor.

ptr is a pointer to the reference counted object to which this class points. If *ptr* is not NULL **T::IncRef()** will be called on the object.

template<class T > wxObjectDataPtr< T >::wxObjectDataPtr (const wxObjectDataPtr< T > & *tocopy*)

This copy constructor increases the count of the reference counted object to which *tocopy* points and then this class will point to, as well.

template<class T > wxObjectDataPtr< T >::~wxObjectDataPtr ( )

Decreases the reference count of the object to which this class points.

21.501.4 Member Function Documentation

template<class T > T* wxObjectDataPtr< T >::get ( ) const

Gets a pointer to the reference counted object to which this class points.

template<class T > wxObjectDataPtr< T >::operator unspecified_bool_type ( ) const

Conversion to a boolean expression (in a variant which is not convertable to anything but a boolean expression).

If this class contains a valid pointer it will return true, if it contains a NULL pointer it will return false.

template<class T > T& wxObjectDataPtr< T >::operator* ( ) const

Returns a reference to the object.

If the internal pointer is NULL this method will cause an assert in debug mode.

template<class T > T* wxObjectDataPtr< T >::operator-> ( ) const

Returns a pointer to the reference counted object to which this class points.

If this the internal pointer is NULL, this method will assert in debug mode.

template<class T > wxObjectDataPtr<T>& wxObjectDataPtr< T >::operator= (const wxObjectDataPtr< T > & *tocopy*)

Assignment operator.

template<class T > wxObjectDataPtr<T>& wxObjectDataPtr< T >::operator= (T * *ptr*)

Assignment operator.

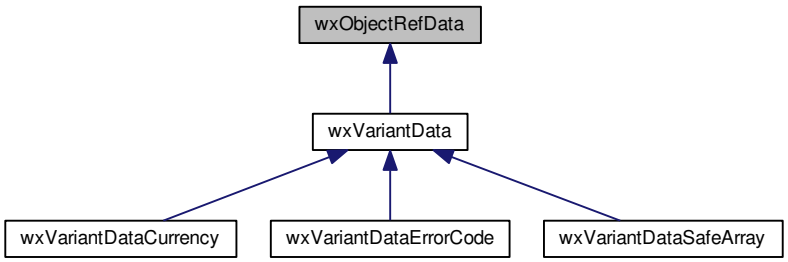
template<class T > void wxObjectDataPtr< T >::reset (T * *ptr*)

Reset this class to ptr which points to a reference counted object and calls T::DecRef() on the previously owned object.

21.502 wxObjectRefData Class Reference

`#include <wx/object.h>`

Inheritance diagram for wxObjectRefData:



21.502.1 Detailed Description

This class is just a typedef to wxRefCounter and is used by wxObject.

Derive classes from this to store your own data in wxObject-derived classes. When retrieving information from a wxObject's reference data, you will need to cast to your own derived class.

Below is an example illustrating how to store reference counted data in a class derived from wxObject including copy-on-write semantics.

21.502.2 Example

```
// include file
// ------------

class MyCar : public wxObject
{
public:
    MyCar() { }
    MyCar( int price );

    bool IsOk() const { return m_refData != NULL; }

    bool operator == ( const MyCar& car ) const;
    bool operator != (const MyCar& car) const { return !(*this == car); }

    void SetPrice( int price );
    int GetPrice() const;

protected:
    virtual wxObjectRefData *CreateRefData() const;
    virtual wxObjectRefData *CloneRefData(const wxObjectRefData *data) const;

    wxDECLARE_DYNAMIC_CLASS(MyCar)
};


// implementation
// --------------
```

```
// the reference data class is typically a private class only visible in the
// implementation source file of the refcounted class.
class MyCarRefData : public wxObjectRefData
{
public:
    MyCarRefData()
    {
        m_price = 0;
    }

    MyCarRefData( const MyCarRefData& data )
        : wxObjectRefData()
    {
        // copy refcounted data; this is usually a time- and memory-consuming operation
        // and is only done when two (or more) MyCar instances need to unshare a
        // common instance of MyCarRefData
        m_price = data.m_price;
    }

    bool operator == (const MyCarRefData& data) const
    {
        return m_price == data.m_price;
    }

private:
    // in real world, reference counting is usually used only when
    // the wxObjectRefData-derived class holds data very memory-consuming;
    // in this example the various MyCar instances may share a MyCarRefData
    // instance which however only takes 4 bytes for this integer!
    int m_price;
};


#define M_CARDATA ((MyCarRefData *)m_refData)
wxIMPLEMENT_DYNAMIC_CLASS(MyCar, wxObject);

MyCar::MyCar( int price )
{
    // here we init the MyCar internal data:
    m_refData = new MyCarRefData();
    M_CARDATA->m_price = price;
}

wxObjectRefData *MyCar::CreateRefData() const
{
    return new MyCarRefData;
}

wxObjectRefData *MyCar::CloneRefData(const wxObjectRefData *data) const
{
    return new MyCarRefData(*(MyCarRefData *)data);
}

bool MyCar::operator == ( const MyCar& car ) const
{
    if (m_refData == car.m_refData)
        return true;
    if (!m_refData || !car.m_refData)
        return false;

    // here we use the MyCarRefData::operator==() function.
    // Note however that this comparison may be very slow if the
    // reference data contains a lot of data to be compared.
    return ( *(MyCarRefData*)m_refData == *(MyCarRefData*)car.m_refData );
}

void MyCar::SetPrice( int price )
{
    // since this function modifies one of the MyCar internal property,
    // we need to be sure that the other MyCar instances which share the
    // same MyCarRefData instance are not affected by this call.
    // I.e. it's very important to call UnShare() in all setters of
    // refcounted classes!
    UnShare();

    M_CARDATA->m_price = price;
}

int MyCar::GetPrice() const
{
    wxCHECK_MSG( IsOk(), -1, "invalid car" );

    return M_CARDATA->m_price;
}
```

Library: wxBase

Category: Runtime Type Information (RTTI)

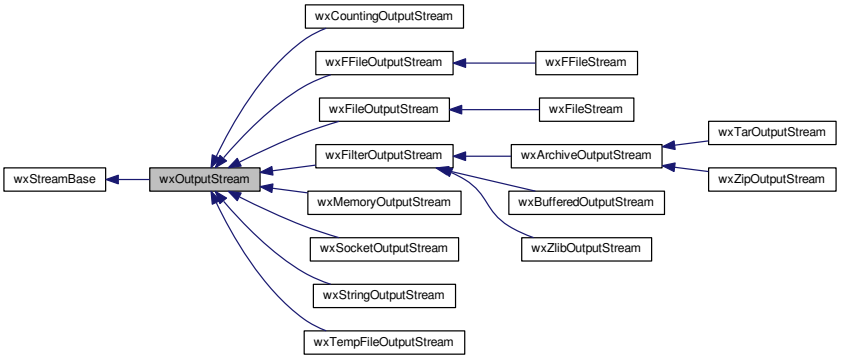
See also

wxObject, wxObjectDataPtr<T>, Reference Counting

21.503 wxOutputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxOutputStream:



21.503.1 Detailed Description

wxOutputStream is an abstract base class which may not be used directly.

It is the base class of all streams which provide a Write() function, i.e. which can be used to output data (e.g. to a file, to a socket, etc).

If you want to create your own output stream, you'll need to derive from this class and implement the protected OnSysWrite() function only.

Library: wxBase

Category: Streams

Public Member Functions

- wxOutputStream ()

  Creates a dummy wxOutputStream object.
- virtual ~wxOutputStream ()

  Destructor.
- virtual bool Close ()

  Closes the stream, returning false if an error occurs.

- virtual size_t LastWrite () const

  Returns the number of bytes written during the last Write().
- void PutC (char c)

  Puts the specified character in the output queue and increments the stream position.
- virtual wxFileOffset SeekO (wxFileOffset pos, wxSeekMode mode=wxFromStart)

  Changes the stream current position.
- virtual wxFileOffset TellO () const

  Returns the current stream position.
- virtual wxOutputStream & Write (const void ∗buffer, size_t size)

  Writes up to the specified amount of bytes using the data of buffer.
- wxOutputStream & Write (wxInputStream &stream_in)

  Reads data from the specified input stream and stores them in the current stream.
- bool WriteAll (const void ∗buffer, size_t size)

  Writes exactly the specified number of bytes from the buffer.

Protected Member Functions

- size_t OnSysWrite (const void ∗buffer, size_t bufsize)

  Internal function.

21.503.2 Constructor & Destructor Documentation

wxOutputStream::wxOutputStream ( )

Creates a dummy wxOutputStream object.

virtual wxOutputStream::∼wxOutputStream ( ) [virtual]

Destructor.

21.503.3 Member Function Documentation

virtual bool wxOutputStream::Close ( ) [virtual]

Closes the stream, returning false if an error occurs.

The stream is closed implicitly in the destructor if Close() is not called explicitly.

If this stream wraps another stream or some other resource such as a file, then the underlying resource is closed too if it is owned by this stream, or left open otherwise.

Reimplemented in wxZipOutputStream, wxTarOutputStream, and wxArchiveOutputStream.

virtual size_t wxOutputStream::LastWrite ( ) const [virtual]

Returns the number of bytes written during the last Write().

It may return 0 even if there is no error on the stream if it is only temporarily impossible to write to it.

size_t wxOutputStream::OnSysWrite (const void * *buffer*, size_t *bufsize*) `[protected]`

Internal function.

It is called when the stream wants to write data of the specified size *bufsize* into the given *buffer*.

It should return the size that was actually wrote (which maybe zero if *bufsize* is zero or if an error occurred; in this last case the internal variable `m_lasterror` should be appropriately set).

void wxOutputStream::PutC (char *c*)

Puts the specified character in the output queue and increments the stream position.

virtual wxFileOffset wxOutputStream::SeekO (wxFileOffset *pos*, wxSeekMode *mode* = wxFromStart) `[virtual]`

Changes the stream current position.

Parameters

pos	Offset to seek to.
mode	One of wxFromStart, wxFromEnd, wxFromCurrent.

Returns

The new stream position or wxInvalidOffset on error.

Reimplemented in wxBufferedOutputStream.

virtual wxFileOffset wxOutputStream::TellO ( ) const `[virtual]`

Returns the current stream position.

virtual wxOutputStream& wxOutputStream::Write (const void * *buffer*, size_t *size*) `[virtual]`

Writes up to the specified amount of bytes using the data of buffer.

Note that not all data can always be written so you must check the number of bytes really written to the stream using LastWrite() when this function returns.

In some cases (for example a write end of a pipe which is currently full) it is even possible that there is no errors and zero bytes have been written. This function returns a reference on the current object, so the user can test any states of the stream right away.

wxOutputStream& wxOutputStream::Write (wxInputStream & *stream_in*)

Reads data from the specified input stream and stores them in the current stream.

The data is read until an error is raised by one of the two streams.

bool wxOutputStream::WriteAll (const void * *buffer*, size_t *size*)

Writes exactly the specified number of bytes from the buffer.

Returns true if exactly *size* bytes were written. Otherwise, returns false and LastWrite() should be used to retrieve the exact amount of the data written if necessary.

This method uses repeated calls to Write() (which may return writing only part of the data) if necessary.

Since

2.9.5

21.504 wxOverlay Class Reference

`#include <wx/overlay.h>`

21.504.1 Detailed Description

Creates an overlay over an existing window, allowing for manipulations like rubberbanding, etc.

On wxOSX the overlay is implemented with native platform APIs, on the other platforms it is simulated using wxMemoryDC.

Library: wxCore

See also

wxDCOverlay, wxDC

Public Member Functions

- wxOverlay ()
- ~wxOverlay ()
- void Reset ()

  Clears the overlay without restoring the former state.

21.504.2 Constructor & Destructor Documentation

wxOverlay::wxOverlay ( )

wxOverlay::~wxOverlay ( )

21.504.3 Member Function Documentation

void wxOverlay::Reset ( )

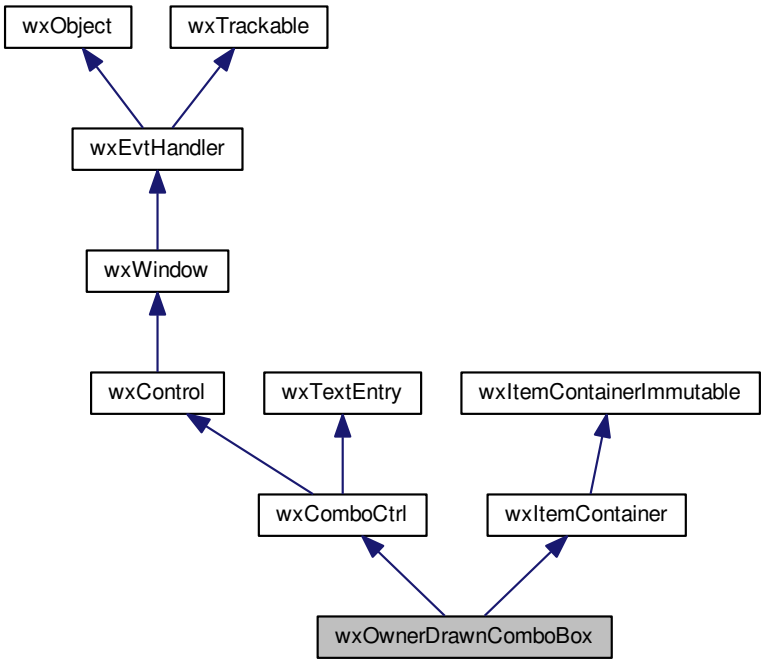
Clears the overlay without restoring the former state.

To be done, for example, when the window content has been changed and repainted.

21.505 wxOwnerDrawnComboBox Class Reference

`#include <wx/odcombo.h>`

Inheritance diagram for wxOwnerDrawnComboBox:



21.505.1 Detailed Description

wxOwnerDrawnComboBox is a combobox with owner-drawn list items.

In essence, it is a wxComboCtrl with wxVListBox popup and wxControlWithItems interface.

Implementing item drawing and measuring is similar to wxVListBox. Application needs to subclass wxOwnerDrawnComboBox and implement OnDrawItem(), OnMeasureItem() and OnMeasureItemWidth().

Styles

This class supports the following styles:

- wxODCB_DCLICK_CYCLES: Double-clicking cycles item if wxCB_READONLY is also used. Synonymous with wxCC_SPECIAL_DCLICK.
- wxODCB_STD_CONTROL_PAINT: Control itself is not custom painted using OnDrawItem. Even if this style is not used, writable wxOwnerDrawnComboBox is never custom painted unless SetCustomPaintWidth() is called.

See also

wxComboCtrl window styles and Window Styles.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_COMBOBOX(id, func): Process a wxEVT_COMBOBOX event, when an item on the list is selected. Note that calling GetValue() returns the new value of selection.

See also

Events emitted by wxComboCtrl.

Library: wxAdvanced

Category: Controls

See also

wxComboCtrl, wxComboBox, wxVListBox, wxCommandEvent

Public Member Functions

- wxOwnerDrawnComboBox ()

  Default constructor.
- wxOwnerDrawnComboBox (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name="comboBox")

  Constructor, creating and showing a owner-drawn combobox.
- wxOwnerDrawnComboBox (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name="comboBox")

  Constructor, creating and showing a owner-drawn combobox.
- virtual ~wxOwnerDrawnComboBox ()

  Destructor, destroying the owner-drawn combobox.
- bool IsEmpty () const

  IsEmpty() is not available in this class.
- bool IsListEmpty () const

  Returns true if the list of combobox choices is empty.
- bool IsTextEmpty () const

  Returns true if the text of the combobox is empty.
- virtual int GetWidestItem ()

  Returns index to the widest item in the list.
- virtual int GetWidestItemWidth ()

  Returns width of the widest item in the list.

- bool Create (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Creates the combobox for two-step construction.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, int n, const wxString choices[], long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Creates the combobox for two-step construction.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &value, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxComboBoxNameStr)

  Creates the combobox for two-step construction.

Protected Member Functions

- virtual void OnDrawBackground (wxDC &dc, const wxRect &rect, int item, int flags) const

  This method is used to draw the items background and, maybe, a border around it.
- virtual void OnDrawItem (wxDC &dc, const wxRect &rect, int item, int flags) const

  The derived class may implement this function to actually draw the item with the given index on the provided DC.
- virtual wxCoord OnMeasureItem (size_t item) const

  The derived class may implement this method to return the height of the specified item (in pixels).
- virtual wxCoord OnMeasureItemWidth (size_t item) const

  The derived class may implement this method to return the width of the specified item (in pixels).

Additional Inherited Members

21.505.2 Constructor & Destructor Documentation

wxOwnerDrawnComboBox::wxOwnerDrawnComboBox ( )

Default constructor.

wxOwnerDrawnComboBox::wxOwnerDrawnComboBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "comboBox")

Constructor, creating and showing a owner-drawn combobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial selection string. An empty string indicates no selection.
pos	Window position.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
n	Number of strings with which to initialise the control.
choices	An array of strings with which to initialise the control.
style	Window style. See wxOwnerDrawnComboBox.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

wxOwnerDrawnComboBox::wxOwnerDrawnComboBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "comboBox")

Constructor, creating and showing a owner-drawn combobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value `wxID_ANY` indicates a default value.
value	Initial selection string. An empty string indicates no selection.
pos	Window position.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
choices	An array of strings with which to initialise the control.
style	Window style. See wxOwnerDrawnComboBox.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxOwnerDrawnComboBox::~wxOwnerDrawnComboBox ( ) [virtual]

Destructor, destroying the owner-drawn combobox.

21.505.3 Member Function Documentation

bool wxOwnerDrawnComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxComboBoxNameStr)

Creates the combobox for two-step construction.

See wxOwnerDrawnComboBox() for further details.

Remarks

Derived classes should call or replace this function.

bool wxOwnerDrawnComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, int *n*, const wxString *choices[]*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxComboBoxNameStr)

Creates the combobox for two-step construction.

See wxOwnerDrawnComboBox() for further details.

Remarks

Derived classes should call or replace this function.

bool wxOwnerDrawnComboBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxComboBoxNameStr)

Creates the combobox for two-step construction.

See wxOwnerDrawnComboBox() for further details.

Remarks

Derived classes should call or replace this function.

virtual int wxOwnerDrawnComboBox::GetWidestItem ( ) [virtual]

Returns index to the widest item in the list.

virtual int wxOwnerDrawnComboBox::GetWidestItemWidth ( ) [virtual]

Returns width of the widest item in the list.

bool wxOwnerDrawnComboBox::IsEmpty ( ) const [virtual]

IsEmpty() is not available in this class.

This method is documented here only to notice that it can't be used with this class because of the ambiguity between the methods with the same name inherited from wxItemContainer and wxTextEntry base classes.

Because of this, any attempt to call it results in a compilation error and you should use either IsListEmpty() or IsTextEmpty() depending on what exactly do you want to test.

Reimplemented from wxTextEntry.

bool wxOwnerDrawnComboBox::IsListEmpty ( ) const

Returns true if the list of combobox choices is empty.

Use this method instead of (not available in this class) IsEmpty() to test if the list of items is empty.

Since

3.1.0

bool wxOwnerDrawnComboBox::IsTextEmpty ( ) const

Returns true if the text of the combobox is empty.

Use this method instead of (not available in this class) IsEmpty() to test if the text currently entered into the combobox is empty.

Since

3.1.0

virtual void wxOwnerDrawnComboBox::OnDrawBackground (wxDC & *dc,* const wxRect & *rect,* int *item,* int *flags*) const [protected], [virtual]

This method is used to draw the items background and, maybe, a border around it.

The base class version implements a reasonable default behaviour which consists in drawing the selected item with the standard background colour and drawing a border around the item if it is either selected or current.

Remarks

flags has the same meaning as with OnDrawItem().

virtual void wxOwnerDrawnComboBox::OnDrawItem (wxDC & *dc,* const wxRect & *rect,* int *item,* int *flags*) const `[protected], [virtual]`

The derived class may implement this function to actually draw the item with the given index on the provided DC.

If function is not implemented, the item text is simply drawn, as if the control was a normal combobox.

Parameters

dc	The device context to use for drawing
rect	The bounding rectangle for the item being drawn (DC clipping region is set to this rectangle before calling this function)
item	The index of the item to be drawn
flags	A combination of the wxOwnerDrawnComboBoxPaintingFlags enumeration values.

virtual wxCoord wxOwnerDrawnComboBox::OnMeasureItem (size_t *item*) const `[protected], [virtual]`

The derived class may implement this method to return the height of the specified item (in pixels).

The default implementation returns text height, as if this control was a normal combobox.

virtual wxCoord wxOwnerDrawnComboBox::OnMeasureItemWidth (size_t *item*) const `[protected], [virtual]`

The derived class may implement this method to return the width of the specified item (in pixels).

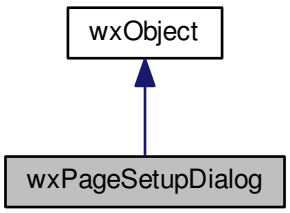
If -1 is returned, then the item text width is used.

The default implementation returns -1.

21.506 wxPageSetupDialog Class Reference

`#include <wx/printdlg.h>`

Inheritance diagram for wxPageSetupDialog:



21.506.1 Detailed Description

This class represents the page setup common dialog.

The page setup dialog contains controls for paper size (letter, A4, A5 etc.), orientation (landscape or portrait), and, only under Windows currently, controls for setting left, top, right and bottom margin sizes in millimetres.

The exact appearance of this dialog varies among the platforms as a native dialog is used when available (currently the case for all major platforms).

When the dialog has been closed, you need to query the wxPageSetupDialogData object associated with the dialog.

Note that the OK and Cancel buttons do not destroy the dialog; this must be done by the application.

Library: wxCore

Category: Printing Framework

See also

Printing Framework Overview, wxPrintDialog, wxPageSetupDialogData

Public Member Functions

- wxPageSetupDialog (wxWindow *parent, wxPageSetupDialogData *data=NULL)

  Constructor.
- virtual ~wxPageSetupDialog ()

  Destructor.
- wxPageSetupDialogData & GetPageSetupData ()

  Returns the wxPageSetupDialogData object associated with the dialog.
- int ShowModal ()

  Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise.

Additional Inherited Members

21.506.2 Constructor & Destructor Documentation

wxPageSetupDialog::wxPageSetupDialog (wxWindow * *parent*, wxPageSetupDialogData * *data* = `NULL`)

Constructor.

Pass a parent window, and optionally a pointer to a block of page setup data, which will be copied to the print dialog's internal data.

virtual wxPageSetupDialog::~wxPageSetupDialog ( ) `[virtual]`

Destructor.

21.506.3 Member Function Documentation

wxPageSetupDialogData& wxPageSetupDialog::GetPageSetupData ( )

Returns the wxPageSetupDialogData object associated with the dialog.

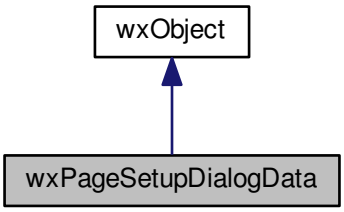
int wxPageSetupDialog::ShowModal ( )

Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise.

21.507 wxPageSetupDialogData Class Reference

```
#include <wx/cmndata.h>
```

Inheritance diagram for wxPageSetupDialogData:



21.507.1 Detailed Description

This class holds a variety of information related to wxPageSetupDialog.

It contains a wxPrintData member which is used to hold basic printer configuration data (as opposed to the user-interface configuration settings stored by wxPageSetupDialogData).

Library: wxCore

Category: Printing Framework, Data Structures

See also

Printing Framework Overview, wxPageSetupDialog

Public Member Functions

- wxPageSetupDialogData ()

  Default constructor.
- wxPageSetupDialogData (const wxPageSetupDialogData &data)

  Copy constructor.
- wxPageSetupDialogData (const wxPrintData &printData)

  Construct an object from a print data object.
- virtual ~wxPageSetupDialogData ()

  Destructor.
- void EnableHelp (bool flag)

  Enables or disables the "Help" button (Windows only).
- void EnableMargins (bool flag)

  Enables or disables the margin controls (Windows only).
- void EnableOrientation (bool flag)

  Enables or disables the orientation control (Windows only).
- void EnablePaper (bool flag)

  Enables or disables the paper size control (Windows only).
- void EnablePrinter (bool flag)

  Enables or disables the "Printer" button, which invokes a printer setup dialog.

- bool GetDefaultInfo () const

  Returns true if the dialog will simply return default printer information (such as orientation) instead of showing a dialog (Windows only).
- bool GetDefaultMinMargins () const

  Returns true if the page setup dialog will take its minimum margin values from the currently selected printer properties (Windows only).
- bool GetEnableHelp () const

  Returns true if the printer setup button is enabled.
- bool GetEnableMargins () const

  Returns true if the margin controls are enabled (Windows only).
- bool GetEnableOrientation () const

  Returns true if the orientation control is enabled (Windows only).
- bool GetEnablePaper () const

  Returns true if the paper size control is enabled (Windows only).
- bool GetEnablePrinter () const

  Returns true if the printer setup button is enabled.
- wxPoint GetMarginBottomRight () const

  Returns the right (x) and bottom (y) margins in millimetres.
- wxPoint GetMarginTopLeft () const

  Returns the left (x) and top (y) margins in millimetres.
- wxPoint GetMinMarginBottomRight () const

  Returns the right (x) and bottom (y) minimum margins the user can enter (Windows only).
- wxPoint GetMinMarginTopLeft () const

  Returns the left (x) and top (y) minimum margins the user can enter (Windows only).
- wxPaperSize GetPaperId () const

  Returns the paper id (stored in the internal wxPrintData object).
- wxSize GetPaperSize () const

  Returns the paper size in millimetres.
- wxPrintData & GetPrintData ()

  Returns a reference to the print data associated with this object.
- const wxPrintData & GetPrintData () const
- bool IsOk () const

  Returns true if the print data associated with the dialog data is valid.
- void SetDefaultInfo (bool flag)

  Pass true if the dialog will simply return default printer information (such as orientation) instead of showing a dialog (Windows only).
- void SetDefaultMinMargins (bool flag)

  Pass true if the page setup dialog will take its minimum margin values from the currently selected printer properties (Windows only).
- void SetMarginBottomRight (const wxPoint &pt)

  Sets the right (x) and bottom (y) margins in millimetres.
- void SetMarginTopLeft (const wxPoint &pt)

  Sets the left (x) and top (y) margins in millimetres.
- void SetMinMarginBottomRight (const wxPoint &pt)

  Sets the right (x) and bottom (y) minimum margins the user can enter (Windows only).
- void SetMinMarginTopLeft (const wxPoint &pt)

  Sets the left (x) and top (y) minimum margins the user can enter (Windows only).
- void SetPaperId (wxPaperSize id)

  Sets the paper size id.
- void SetPaperSize (const wxSize &size)

  Sets the paper size in millimetres.

- void SetPrintData (const wxPrintData &printData)

  Sets the print data associated with this object.
- wxPageSetupDialogData & operator= (const wxPrintData &data)

  Assigns print data to this object.
- wxPageSetupDialogData & operator= (const wxPageSetupDialogData &data)

  Assigns page setup data to this object.

Additional Inherited Members

21.507.2 Constructor & Destructor Documentation

wxPageSetupDialogData::wxPageSetupDialogData ( )

Default constructor.

wxPageSetupDialogData::wxPageSetupDialogData (const wxPageSetupDialogData & *data*)

Copy constructor.

wxPageSetupDialogData::wxPageSetupDialogData (const wxPrintData & *printData*)

Construct an object from a print data object.

virtual wxPageSetupDialogData::~wxPageSetupDialogData ( ) [virtual]

Destructor.

21.507.3 Member Function Documentation

void wxPageSetupDialogData::EnableHelp (bool *flag*)

Enables or disables the "Help" button (Windows only).

void wxPageSetupDialogData::EnableMargins (bool *flag*)

Enables or disables the margin controls (Windows only).

void wxPageSetupDialogData::EnableOrientation (bool *flag*)

Enables or disables the orientation control (Windows only).

void wxPageSetupDialogData::EnablePaper (bool *flag*)

Enables or disables the paper size control (Windows only).

void wxPageSetupDialogData::EnablePrinter (bool *flag*)

Enables or disables the "Printer" button, which invokes a printer setup dialog.

bool wxPageSetupDialogData::GetDefaultInfo ( ) const

Returns true if the dialog will simply return default printer information (such as orientation) instead of showing a dialog (Windows only).

bool wxPageSetupDialogData::GetDefaultMinMargins ( ) const

Returns true if the page setup dialog will take its minimum margin values from the currently selected printer properties (Windows only).

bool wxPageSetupDialogData::GetEnableHelp ( ) const

Returns true if the printer setup button is enabled.

bool wxPageSetupDialogData::GetEnableMargins ( ) const

Returns true if the margin controls are enabled (Windows only).

bool wxPageSetupDialogData::GetEnableOrientation ( ) const

Returns true if the orientation control is enabled (Windows only).

bool wxPageSetupDialogData::GetEnablePaper ( ) const

Returns true if the paper size control is enabled (Windows only).

bool wxPageSetupDialogData::GetEnablePrinter ( ) const

Returns true if the printer setup button is enabled.

wxPoint wxPageSetupDialogData::GetMarginBottomRight ( ) const

Returns the right (x) and bottom (y) margins in millimetres.

wxPoint wxPageSetupDialogData::GetMarginTopLeft ( ) const

Returns the left (x) and top (y) margins in millimetres.

wxPoint wxPageSetupDialogData::GetMinMarginBottomRight ( ) const

Returns the right (x) and bottom (y) minimum margins the user can enter (Windows only).

Units are in millimetres.

wxPoint wxPageSetupDialogData::GetMinMarginTopLeft ( ) const

Returns the left (x) and top (y) minimum margins the user can enter (Windows only).

Units are in millimetres.

wxPaperSize wxPageSetupDialogData::GetPaperId ( ) const

Returns the paper id (stored in the internal wxPrintData object).

See also

wxPrintData::SetPaperId()

wxSize wxPageSetupDialogData::GetPaperSize ( ) const

Returns the paper size in millimetres.

wxPrintData& wxPageSetupDialogData::GetPrintData ( )

Returns a reference to the print data associated with this object.

const wxPrintData& wxPageSetupDialogData::GetPrintData ( ) const

bool wxPageSetupDialogData::IsOk ( ) const

Returns true if the print data associated with the dialog data is valid.

This can return false on Windows if the current printer is not set, for example. On all other platforms, it returns true.

wxPageSetupDialogData& wxPageSetupDialogData::operator= (const wxPrintData & *data*)

Assigns print data to this object.

wxPageSetupDialogData& wxPageSetupDialogData::operator= (const wxPageSetupDialogData & *data*)

Assigns page setup data to this object.

void wxPageSetupDialogData::SetDefaultInfo (bool *flag*)

Pass true if the dialog will simply return default printer information (such as orientation) instead of showing a dialog (Windows only).

void wxPageSetupDialogData::SetDefaultMinMargins (bool *flag*)

Pass true if the page setup dialog will take its minimum margin values from the currently selected printer properties (Windows only).

Units are in millimetres.

void wxPageSetupDialogData::SetMarginBottomRight (const wxPoint & *pt*)

Sets the right (x) and bottom (y) margins in millimetres.

void wxPageSetupDialogData::SetMarginTopLeft (const wxPoint & *pt*)

Sets the left (x) and top (y) margins in millimetres.

void wxPageSetupDialogData::SetMinMarginBottomRight (const wxPoint & *pt*)

Sets the right (x) and bottom (y) minimum margins the user can enter (Windows only).

Units are in millimetres.

void wxPageSetupDialogData::SetMinMarginTopLeft (const wxPoint & *pt*)

Sets the left (x) and top (y) minimum margins the user can enter (Windows only).

Units are in millimetres.

void wxPageSetupDialogData::SetPaperId (wxPaperSize *id*)

Sets the paper size id.

Calling this function overrides the explicit paper dimensions passed in SetPaperSize().

See also

wxPrintData::SetPaperId()

void wxPageSetupDialogData::SetPaperSize (const wxSize & *size*)

Sets the paper size in millimetres.

If a corresponding paper id is found, it will be set in the internal wxPrintData object, otherwise the paper size overrides the paper id.

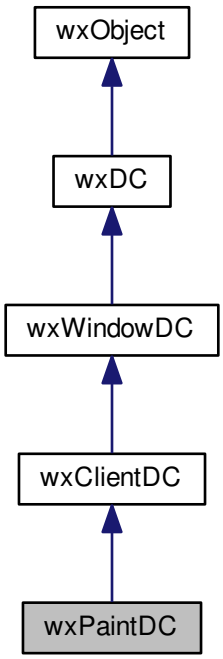
void wxPageSetupDialogData::SetPrintData (const wxPrintData & *printData*)

Sets the print data associated with this object.

21.508 wxPaintDC Class Reference

```
#include <wx/dcclient.h>
```

Inheritance diagram for wxPaintDC:



21.508.1 Detailed Description

A wxPaintDC must be constructed if an application wishes to paint on the client area of a window from within an EVT_PAINT() event handler.

This should normally be constructed as a temporary stack object; don't store a wxPaintDC object. If you have an EVT_PAINT() handler, you *must* create a wxPaintDC object within it even if you don't actually use it.

Using wxPaintDC within your EVT_PAINT() handler is important because it automatically sets the clipping area to the damaged area of the window. Attempts to draw outside this area do not appear.

To draw on a window from outside your EVT_PAINT() handler, construct a wxClientDC object.

To draw on the whole window including decorations, construct a wxWindowDC object (Windows only).

A wxPaintDC object is initialized to use the same font and colours as the window it is associated with.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxClientDC, wxMemoryDC, wxWindowDC, wxScreenDC

Public Member Functions

- wxPaintDC (wxWindow *window)

  Constructor.

Additional Inherited Members

21.508.2 Constructor & Destructor Documentation

wxPaintDC::wxPaintDC (wxWindow * *window*)

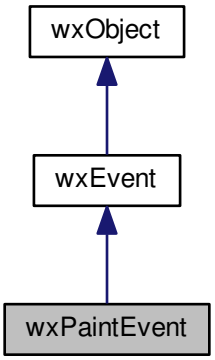
Constructor.

Pass a pointer to the window on which you wish to paint.

21.509 wxPaintEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxPaintEvent:



21.509.1 Detailed Description

A paint event is sent when a window's contents needs to be repainted.

The handler of this event must create a wxPaintDC object and use it for painting the window contents. For example:

```
void MyWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    DrawMyDocument(dc);
}
```

Notice that you must *not* create other kinds of wxDC (e.g. wxClientDC or wxWindowDC) in EVT_PAINT handlers and also don't create wxPaintDC outside of this event handlers.

You can optimize painting by retrieving the rectangles that have been damaged and only repainting these. The rectangles are in terms of the client area, and are unscrolled, so you will need to do some calculations using the current view position to obtain logical, scrolled units. Here is an example of using the wxRegionIterator class:

```
// Called when window needs to be repainted.
void MyWindow::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    // Find Out where the window is scrolled to
    int vbX,vbY;                    // Top left corner of client
    GetViewStart(&vbX,&vbY);

    int vX,vY,vW,vH;                // Dimensions of client area in pixels
    wxRegionIterator upd(GetUpdateRegion()); // get the update rect list

    while (upd)
    {
        vX = upd.GetX();
        vY = upd.GetY();
        vW = upd.GetW();
        vH = upd.GetH();

        // Alternatively we can do this:
        // wxRect rect(upd.GetRect());

        // Repaint this rectangle
        ...some code...

        upd ++ ;
    }
}
```

Remarks

Please notice that in general it is impossible to change the drawing of a standard control (such as wxButton) and so you shouldn't attempt to handle paint events for them as even if it might work on some platforms, this is inherently not portable and won't work everywhere.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxPaintEvent& event)

Event macros:

- EVT_PAINT(func): Process a `wxEVT_PAINT` event.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxPaintEvent (int id=0)

  Constructor.

Additional Inherited Members

21.509.2 Constructor & Destructor Documentation

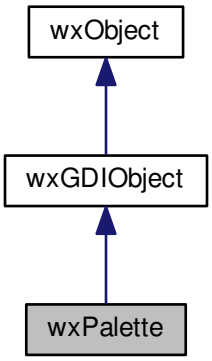
wxPaintEvent::wxPaintEvent (int *id* = 0)

Constructor.

21.510 wxPalette Class Reference

```
#include <wx/palette.h>
```

Inheritance diagram for wxPalette:



21.510.1 Detailed Description

A palette is a table that maps pixel values to RGB colours.

It allows the colours of a low-depth bitmap, for example, to be mapped to the available colours in a display. The notion of palettes is becoming more and more obsolete nowadays and only the MSW port is still using a native palette. All other ports use generic code which is basically just an array of colours.

It is likely that in the future the only use for palettes within wxWidgets will be for representing colour indices from images (such as GIF or PNG). The image handlers for these formats have been modified to create a palette if there is such information in the original image file (usually 256 or less colour images). See wxImage for more information.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullPalette

See also

wxDC::SetPalette(), wxBitmap

Public Member Functions

- wxPalette ()

  Default constructor.
- wxPalette (const wxPalette &palette)

  Copy constructor, uses Reference Counting.
- wxPalette (int n, const unsigned char *red, const unsigned char *green, const unsigned char *blue)

  Creates a palette from arrays of size n, one for each red, blue or green component.
- virtual ~wxPalette ()

  Destructor.
- bool Create (int n, const unsigned char *red, const unsigned char *green, const unsigned char *blue)

  Creates a palette from arrays of size n, one for each red, blue or green component.
- virtual int GetColoursCount () const

  Returns number of entries in palette.
- int GetPixel (unsigned char red, unsigned char green, unsigned char blue) const

  Returns a pixel value (index into the palette) for the given RGB values.
- bool GetRGB (int pixel, unsigned char *red, unsigned char *green, unsigned char *blue) const

  Returns RGB values for a given palette index.
- virtual bool IsOk () const

  Returns true if palette data is present.
- wxPalette & operator= (const wxPalette &palette)

  Assignment operator, using Reference Counting.

Additional Inherited Members

21.510.2 Constructor & Destructor Documentation

wxPalette::wxPalette ( )

Default constructor.

wxPalette::wxPalette (const wxPalette & *palette*)

Copy constructor, uses Reference Counting.

Parameters

palette	A reference to the palette to copy.

wxPalette::wxPalette (int *n*, const unsigned char * *red*, const unsigned char * *green*, const unsigned char * *blue*)

Creates a palette from arrays of size *n*, one for each red, blue or green component.

Parameters

n	The number of indices in the palette.
red	An array of red values.
green	An array of green values.
blue	An array of blue values.

wxPerl Note: In wxPerl this method takes as parameters 3 array references (they must be of the same length).

See also

Create()

virtual wxPalette::~wxPalette ( ) [virtual]

Destructor.

See also

reference-counted object destruction

21.510.3 Member Function Documentation

bool wxPalette::Create (int *n*, const unsigned char * *red*, const unsigned char * *green*, const unsigned char * *blue*)

Creates a palette from arrays of size *n*, one for each red, blue or green component.

Parameters

n	The number of indices in the palette.
red	An array of red values.
green	An array of green values.
blue	An array of blue values.

Returns

true if the creation was successful, false otherwise.

See also

wxPalette()

virtual int wxPalette::GetColoursCount ( ) const [virtual]

Returns number of entries in palette.

int wxPalette::GetPixel (unsigned char *red*, unsigned char *green*, unsigned char *blue*) const

Returns a pixel value (index into the palette) for the given RGB values.

Parameters

red	Red value.
green	Green value.
blue	Blue value.

Returns

The nearest palette index or `wxNOT_FOUND` for unexpected errors.

See also

GetRGB()

bool wxPalette::GetRGB (int *pixel*, unsigned char * *red*, unsigned char * *green*, unsigned char * *blue*) const

Returns RGB values for a given palette index.

Parameters

pixel	The palette index.
red	Receives the red value.
green	Receives the green value.
blue	Receives the blue value.

Returns

true if the operation was successful.

wxPerl Note: In wxPerl this method takes only the *pixel* parameter and returns a 3-element list (or the empty list upon failure).

See also

GetPixel()

virtual bool wxPalette::IsOk ( ) const `[virtual]`

Returns true if palette data is present.

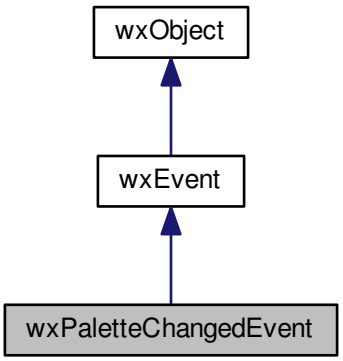
wxPalette& wxPalette::operator= (const wxPalette & *palette*)

Assignment operator, using Reference Counting.

21.511 wxPaletteChangedEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxPaletteChangedEvent:



Public Member Functions

- wxPaletteChangedEvent (wxWindowID winid=0)
- void SetChangedWindow (wxWindow *win)
- wxWindow * GetChangedWindow () const

Additional Inherited Members

21.511.1 Constructor & Destructor Documentation

wxPaletteChangedEvent::wxPaletteChangedEvent (wxWindowID *winid* = 0)

21.511.2 Member Function Documentation

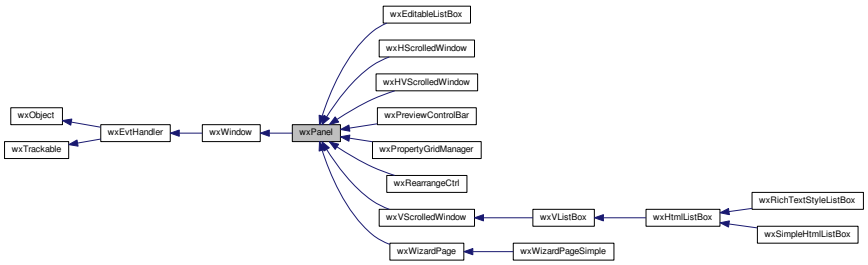
wxWindow* wxPaletteChangedEvent::GetChangedWindow ( ) const

void wxPaletteChangedEvent::SetChangedWindow (wxWindow * *win*)

21.512 wxPanel Class Reference

`#include <wx/panel.h>`

Inheritance diagram for wxPanel:



21.512.1 Detailed Description

A panel is a window on which controls are placed.

It is usually placed within a frame. Its main feature over its parent class wxWindow is code for handling child windows and TAB traversal, which is implemented natively if possible (e.g. in wxGTK) or by wxWidgets itself otherwise.

Note

Tab traversal is implemented through an otherwise undocumented intermediate wxControlContainer class from which any class can derive in addition to the normal wxWindow base class. Please see `wx/containr.h` and `wx/panel.h` to find out how this is achieved.
if not all characters are being intercepted by your OnKeyDown or OnChar handler, it may be because you are using the `wxTAB_TRAVERSAL` style, which grabs some keypresses for use by child controls.

Remarks

By default, a panel has the same colouring as a dialog.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxNavigationKeyEvent& event)

Event macros for events emitted by this class:

- EVT_NAVIGATION_KEY(func): Process a navigation key event.

Library: wxCore

Category: Miscellaneous Windows

See also

wxDialog

Public Member Functions

- wxPanel ()

  Default constructor.
- wxPanel (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTAB_TRAVERSAL, const wxString &name=wxPanelNameStr)

  Constructor.
- virtual ~wxPanel ()

  Destructor.
- bool AcceptsFocus () const

  This method is overridden from wxWindow::AcceptsFocus() and returns true only if there is no child window in the panel which can accept the focus.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTAB_TRAVERSAL, const wxString &name=wxPanelNameStr)

  Used for two-step panel construction.
- virtual void InitDialog ()

  Sends a wxInitDialogEvent, which in turn transfers data to the dialog via validators.
- virtual bool Layout ()

  See wxWindow::SetAutoLayout(): when auto layout is on, this function gets called automatically when the window is resized.
- void OnSysColourChanged (wxSysColourChangedEvent &event)

  The default handler for `wxEVT_SYS_COLOUR_CHANGED`.
- virtual void SetFocus ()

  Overrides wxWindow::SetFocus().
- void SetFocusIgnoringChildren ()

  In contrast to SetFocus() (see above) this will set the focus to the panel even if there are child windows in the panel.

Additional Inherited Members

21.512.2 Constructor & Destructor Documentation

wxPanel::wxPanel ( )

Default constructor.

wxPanel::wxPanel (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTAB_TRAVERSAL, const wxString & *name* = `wxPanelNameStr`)

Constructor.

Parameters

parent	The parent window.
id	An identifier for the panel. `wxID_ANY` is taken to mean a default.
pos	The panel position. The value wxDefaultPosition indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.
size	The panel size. The value wxDefaultSize indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.
style	The window style. See wxPanel.
name	Window name.

See also

Create()

virtual wxPanel::~wxPanel ( ) [virtual]

Destructor.

Deletes any child windows before deleting the physical window.

21.512.3 Member Function Documentation

bool wxPanel::AcceptsFocus ( ) const [virtual]

This method is overridden from wxWindow::AcceptsFocus() and returns true only if there is no child window in the panel which can accept the focus.

This is reevaluated each time a child window is added or removed from the panel.

Reimplemented from wxWindow.

bool wxPanel::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTAB_TRAVERSAL, const wxString & *name* = `wxPanelNameStr`)

Used for two-step panel construction.

See wxPanel() for details.

virtual void wxPanel::InitDialog ( ) [virtual]

Sends a wxInitDialogEvent, which in turn transfers data to the dialog via validators.

See also

wxInitDialogEvent

Reimplemented from wxWindow.

virtual bool wxPanel::Layout ( ) [virtual]

See wxWindow::SetAutoLayout(): when auto layout is on, this function gets called automatically when the window is resized.

Reimplemented from wxWindow.

void wxPanel::OnSysColourChanged (wxSysColourChangedEvent & *event*)

The default handler for `wxEVT_SYS_COLOUR_CHANGED`.

Parameters

event	The colour change event.

Remarks

Changes the panel's colour to conform to the current settings (Windows only). Add an event table entry for your panel class if you wish the behaviour to be different (such as keeping a user-defined background colour). If you do override this function, call wxEvent::Skip() to propagate the notification to child windows and controls.

See also

wxSysColourChangedEvent

virtual void wxPanel::SetFocus ( ) [virtual]

Overrides wxWindow::SetFocus().

This method uses the (undocumented) mix-in class wxControlContainer which manages the focus and TAB logic for controls which usually have child controls.

In practice, if you call this method and the control has at least one child window, the focus will be given to the child window.

See also

wxFocusEvent, wxWindow::SetFocus()

Reimplemented from wxWindow.

void wxPanel::SetFocusIgnoringChildren ( )

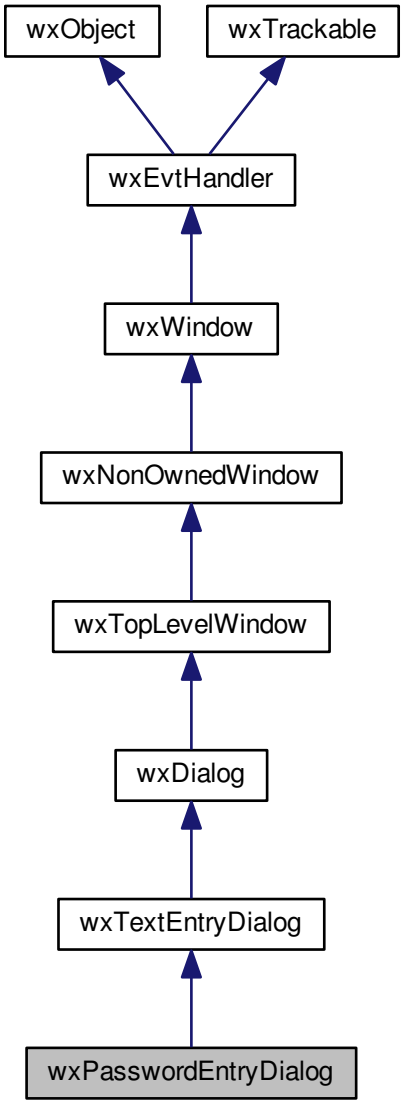
In contrast to SetFocus() (see above) this will set the focus to the panel even if there are child windows in the panel.

This is only rarely needed.

21.513 wxPasswordEntryDialog Class Reference

```
#include <wx/textdlg.h>
```

Inheritance diagram for wxPasswordEntryDialog:



21.513.1 Detailed Description

This class represents a dialog that requests a one-line password string from the user.

It is implemented as a generic wxWidgets dialog.

Library: wxCore

Category: Common Dialogs

See also

wxPasswordEntryDialog Overview

Public Member Functions

- wxPasswordEntryDialog (wxWindow *parent, const wxString &message, const wxString &caption=wxGetPasswordFromUserPromptStr, const wxString &defaultValue=wxEmptyString, long style=wxTextEntryDialogStyle, const wxPoint &pos=wxDefaultPosition)

  Constructor.

Additional Inherited Members

21.513.2 Constructor & Destructor Documentation

wxPasswordEntryDialog::wxPasswordEntryDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption* = wxGetPasswordFromUserPromptStr, const wxString & *defaultValue* = wxEmptyString, long *style* = wxTextEntryDialogStyle, const wxPoint & *pos* = wxDefaultPosition)

Constructor.

Use wxTextEntryDialog::ShowModal to show the dialog.

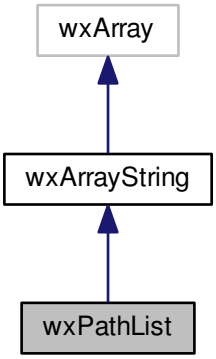
Parameters

parent	Parent window.
message	Message to show on the dialog.
caption	The caption of the dialog.
defaultValue	The default value, which may be the empty string.
style	A dialog style, specifying the buttons (wxOK, wxCANCEL) and an optional wxCENTRE style. You do not need to specify the wxTE_PASSWORD style, it is always applied.
pos	Dialog position.

21.514 wxPathList Class Reference

```
#include <wx/filefn.h>
```

Inheritance diagram for wxPathList:



21.514.1 Detailed Description

The path list is a convenient way of storing a number of directories, and when presented with a filename without a directory, searching for an existing file in those directories.

Be sure to look also at wxStandardPaths if you only want to search files in some standard paths.

Library: wxBase

Category: File Handling

See also

wxArrayString, wxStandardPaths, wxFileName

Public Member Functions

- wxPathList ()

  Standard constructor.
- wxPathList (const wxArrayString &arr)

  Constructs the object calling the Add() function.
- bool Add (const wxString &path)

  Adds the given directory to the path list, if the path is not already in the list.
- void Add (const wxArrayString &arr)

  Adds all elements of the given array as paths.
- void AddEnvList (const wxString &env_variable)

  Finds the value of the given environment variable, and adds all paths to the path list.
- bool EnsureFileAccessible (const wxString &filename)

  Given a full filename (with path), calls Add() with the path of the file.
- wxString FindAbsoluteValidPath (const wxString &file) const

  Like FindValidPath() but this function always returns an absolute path (eventually prepending the current working directory to the value returned wxPathList::FindValidPath()) or an empty string.

- wxString FindValidPath (const wxString &file) const

  Searches the given file in all paths stored in this class.

Additional Inherited Members

21.514.2 Constructor & Destructor Documentation

wxPathList::wxPathList ( )

Standard constructor.

wxPathList::wxPathList (const wxArrayString & *arr*)

Constructs the object calling the Add() function.

21.514.3 Member Function Documentation

bool wxPathList::Add (const wxString & *path*)

Adds the given directory to the path list, if the *path* is not already in the list.

If the path cannot be normalized for some reason, it returns false.

The *path* is always considered to be a directory but no existence checks will be done on it (because if it doesn't exist, it could be created later and thus result a valid path when FindValidPath() is called).

Note

if the given path is relative, it won't be made absolute before adding it (this is why FindValidPath() may return relative paths).

void wxPathList::Add (const wxArrayString & *arr*)

Adds all elements of the given array as paths.

void wxPathList::AddEnvList (const wxString & *env_variable*)

Finds the value of the given environment variable, and adds all paths to the path list.

Useful for finding files in the `PATH` variable, for example.

bool wxPathList::EnsureFileAccessible (const wxString & *filename*)

Given a full filename (with path), calls Add() with the path of the file.

wxString wxPathList::FindAbsoluteValidPath (const wxString & *file*) const

Like FindValidPath() but this function always returns an absolute path (eventually prepending the current working directory to the value returned wxPathList::FindValidPath()) or an empty string.

wxString wxPathList::FindValidPath (const wxString & *file*) const

Searches the given file in all paths stored in this class.

The first path which concatenated to the given string points to an existing file (see wxFileExists()) is returned.

If the file wasn't found in any of the stored paths, an empty string is returned.

The given string must be a file name, eventually with a path prefix (if the path prefix is absolute, only its name will be searched); i.e. it must not end with a directory separator (see wxFileName::GetPathSeparator) otherwise an assertion will fail.

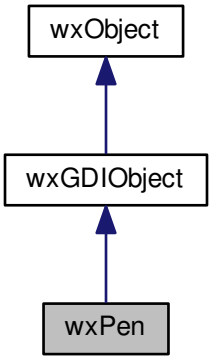
The returned path may be relative to the current working directory.

Note in fact that wxPathList can be used to store both relative and absolute paths so that if you added relative paths, then the current working directory (see wxGetCwd() and wxSetWorkingDirectory()) may affect the value returned by this function!

21.515 wxPen Class Reference

`#include <wx/pen.h>`

Inheritance diagram for wxPen:



21.515.1 Detailed Description

A pen is a drawing tool for drawing outlines.

It is used for drawing lines and painting the outline of rectangles, ellipses, etc. It has a colour, a width and a style.

Note

On a monochrome display, wxWidgets shows all non-white pens as black.

Do not initialize objects on the stack before the program commences, since other required structures may not have been set up yet. Instead, define global pointers to objects and create them in wxApp::OnInit() or when required.

An application may wish to dynamically create pens with different characteristics, and there is the consequent danger that a large number of duplicate pens will be created. Therefore an application may wish to get a pointer to a pen by using the global list of pens wxThePenList, and calling the member function wxPenList::FindOrCreatePen(). See wxPenList for more info.

This class uses reference counting and copy-on-write internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers:

- wxNullPen
- wxBLACK_DASHED_PEN
- wxBLACK_PEN
- wxBLUE_PEN
- wxCYAN_PEN
- wxGREEN_PEN
- wxYELLOW_PEN
- wxGREY_PEN
- wxLIGHT_GREY_PEN
- wxMEDIUM_GREY_PEN
- wxRED_PEN
- wxTRANSPARENT_PEN
- wxWHITE_PEN

See also

wxPenList, wxDC, wxDC::SetPen()

Public Member Functions

- wxPen ()

  Default constructor.
- wxPen (const wxColour &colour, int width=1, wxPenStyle style=wxPENSTYLE_SOLID)

  Constructs a pen from a colour object, pen width and style.
- wxPen (const wxBitmap &stipple, int width)

  Constructs a stippled pen from a stipple bitmap and a width.
- wxPen (const wxPen &pen)

  Copy constructor, uses Reference Counting.
- virtual ~wxPen ()

  Destructor.
- virtual wxPenCap GetCap () const

  Returns the pen cap style, which may be one of `wxCAP_ROUND`, `wxCAP_PROJECTING` *and* `wxCAP_BUTT`.
- virtual wxColour GetColour () const

  Returns a reference to the pen colour.
- virtual int GetDashes (wxDash **dashes) const

Gets an array of dashes (defined as `char` in X, `DWORD` under Windows).

- virtual wxPenJoin GetJoin () const

  Returns the pen join style, which may be one of `wxJOIN_BEVEL`, `wxJOIN_ROUND` and `wxJOIN_MITER`.
- virtual wxBitmap * GetStipple () const

  Gets a pointer to the stipple bitmap.
- virtual wxPenStyle GetStyle () const

  Returns the pen style.
- virtual int GetWidth () const

  Returns the pen width.
- virtual bool IsOk () const

  Returns true if the pen is initialised.
- bool IsNonTransparent () const

  Returns true if the pen is a valid non-transparent pen.
- bool IsTransparent () const

  Returns true if the pen is transparent.
- virtual void SetCap (wxPenCap capStyle)

  Sets the pen cap style, which may be one of `wxCAP_ROUND`, `wxCAP_PROJECTING` and `wxCAP_BUTT`.
- virtual void SetDashes (int n, const wxDash *dash)

  Associates an array of dash values (defined as `char` in X, `DWORD` under Windows) with the pen.
- virtual void SetJoin (wxPenJoin join_style)

  Sets the pen join style, which may be one of `wxJOIN_BEVEL`, `wxJOIN_ROUND` and `wxJOIN_MITER`.
- virtual void SetStipple (const wxBitmap &stipple)

  Sets the bitmap for stippling.
- virtual void SetStyle (wxPenStyle style)

  Set the pen style.
- virtual void SetWidth (int width)

  Sets the pen width.
- bool operator!= (const wxPen &pen) const

  Inequality operator.
- wxPen & operator= (const wxPen &pen)

  Assignment operator, using Reference Counting.
- bool operator== (const wxPen &pen) const

  Equality operator.

- virtual void SetColour (wxColour &colour)

  The pen's colour is changed to the given colour.
- virtual void SetColour (unsigned char red, unsigned char green, unsigned char blue)

  The pen's colour is changed to the given colour.

Additional Inherited Members

21.515.2 Constructor & Destructor Documentation

wxPen::wxPen ( )

Default constructor.

The pen will be uninitialised, and IsOk() will return false.

wxPen::wxPen (const wxColour & *colour*, int *width* = 1, wxPenStyle *style* = wxPENSTYLE_SOLID)

Constructs a pen from a colour object, pen width and style.

Parameters

colour	A colour object.
width	Pen width. Under Windows, the pen width cannot be greater than 1 if the style is `wxPENS↩TYLE_DOT`, `wxPENSTYLE_LONG_DASH`, `wxPENSTYLE_SHORT_DASH`, `wxPENSTY↩LE_DOT_DASH`, or `wxPENSTYLE_USER_DASH`.
style	The style may be one of the wxPenStyle values.

Remarks

Different versions of Windows and different versions of other platforms support very different subsets of the styles above

- there is no similarity even between Windows95 and Windows98 - so handle with care.

See also

SetStyle(), SetColour(), SetWidth()

wxPen::wxPen (const wxBitmap & *stipple,* int *width*)

Constructs a stippled pen from a stipple bitmap and a width.

Parameters

width	Pen width. Under Windows, the pen width cannot be greater than 1 if the style is `wxPENS↩TYLE_DOT`, `wxPENSTYLE_LONG_DASH`, `wxPENSTYLE_SHORT_DASH`, `wxPENSTY↩LE_DOT_DASH`, or `wxPENSTYLE_USER_DASH`.
stipple	A stipple bitmap.

Availability: only available for the wxMSW, wxOSX ports.

See also

SetWidth(), SetStipple()

wxPen::wxPen (const wxPen & *pen*)

Copy constructor, uses Reference Counting.

Parameters

pen	A pointer or reference to a pen to copy.

virtual wxPen::~wxPen ( ) [virtual]

Destructor.

See also

reference-counted object destruction

Remarks

Although all remaining pens are deleted when the application exits, the application should try to clean up all pens itself. This is because wxWidgets cannot know if a pointer to the pen object is stored in an application data structure, and there is a risk of double deletion.

21.515.3 Member Function Documentation

virtual wxPenCap wxPen::GetCap ( ) const `[virtual]`

Returns the pen cap style, which may be one of `wxCAP_ROUND`, `wxCAP_PROJECTING` and `wxCAP_BUTT`.

The default is `wxCAP_ROUND`.

See also

SetCap()

virtual wxColour wxPen::GetColour ( ) const `[virtual]`

Returns a reference to the pen colour.

See also

SetColour()

virtual int wxPen::GetDashes (wxDash ** *dashes*) const `[virtual]`

Gets an array of dashes (defined as `char` in X, `DWORD` under Windows).

dashes is a pointer to the internal array. Do not deallocate or store this pointer.

Returns

The number of dashes associated with this pen.

See also

SetDashes()

virtual wxPenJoin wxPen::GetJoin ( ) const `[virtual]`

Returns the pen join style, which may be one of `wxJOIN_BEVEL`, `wxJOIN_ROUND` and `wxJOIN_MITER`.

The default is `wxJOIN_ROUND`.

See also

SetJoin()

virtual wxBitmap* wxPen::GetStipple ( ) const `[virtual]`

Gets a pointer to the stipple bitmap.

See also

SetStipple()

virtual wxPenStyle wxPen::GetStyle ( ) const `[virtual]`

Returns the pen style.

See also

wxPen(), SetStyle()

virtual int wxPen::GetWidth ( ) const `[virtual]`

Returns the pen width.

See also

SetWidth()

bool wxPen::IsNonTransparent ( ) const

Returns true if the pen is a valid non-transparent pen.

This method returns true if the pen object is initialized and has a non-transparent style. Notice that this should be used instead of simply testing whether GetStyle() returns a style different from wxPENSTYLE_TRANSPARENT if the pen may be invalid as GetStyle() would assert in this case.

See also

IsTransparent()

Since

2.9.2.

virtual bool wxPen::IsOk ( ) const `[virtual]`

Returns true if the pen is initialised.

Notice that an uninitialized pen object can't be queried for any pen properties and all calls to the accessor methods on it will result in an assert failure.

bool wxPen::IsTransparent ( ) const

Returns true if the pen is transparent.

A transparent pen is simply a pen with wxPENSTYLE_TRANSPARENT style.

Notice that this function works even for non-initialized pens (for which it returns false) unlike tests of the form `GetStyle() == wxPENSTYLE_TRANSPARENT` which would assert if the pen is invalid.

See also

IsNonTransparent()

Since

2.9.2.

bool wxPen::operator!= (const wxPen & *pen*) const

Inequality operator.

See reference-counted object comparison for more info.

wxPen& wxPen::operator= (const wxPen & *pen*)

Assignment operator, using Reference Counting.

bool wxPen::operator== (const wxPen & *pen*) const

Equality operator.

See reference-counted object comparison for more info.

virtual void wxPen::SetCap (wxPenCap *capStyle*) [virtual]

Sets the pen cap style, which may be one of `wxCAP_ROUND`, `wxCAP_PROJECTING` and `wxCAP_BUTT`.

The default is `wxCAP_ROUND`.

See also

GetCap()

virtual void wxPen::SetColour (wxColour & *colour*) [virtual]

The pen's colour is changed to the given colour.

See also

GetColour()

virtual void wxPen::SetColour (unsigned char *red,* unsigned char *green,* unsigned char *blue*) [virtual]

The pen's colour is changed to the given colour.

See also

GetColour()

virtual void wxPen::SetDashes (int *n,* const wxDash * *dash*) [virtual]

Associates an array of dash values (defined as `char` in X, `DWORD` under Windows) with the pen.

The array is not deallocated by wxPen, but neither must it be deallocated by the calling application until the pen is deleted or this function is called with a NULL array.

See also

GetDashes()

virtual void wxPen::SetJoin (wxPenJoin *join_style*) `[virtual]`

Sets the pen join style, which may be one of `wxJOIN_BEVEL`, `wxJOIN_ROUND` and `wxJOIN_MITER`.

The default is `wxJOIN_ROUND`.

See also

GetJoin()

virtual void wxPen::SetStipple (const wxBitmap & *stipple*) `[virtual]`

Sets the bitmap for stippling.

See also

GetStipple()

virtual void wxPen::SetStyle (wxPenStyle *style*) `[virtual]`

Set the pen style.

See also

wxPen()

virtual void wxPen::SetWidth (int *width*) `[virtual]`

Sets the pen width.

See also

GetWidth()

21.516 wxPenList Class Reference

`#include <wx/pen.h>`

21.516.1 Detailed Description

There is only one instance of this class: wxThePenList.

Use this object to search for a previously created pen of the desired type and create it if not already found. In some windowing systems, the pen may be a scarce resource, so it can pay to reuse old resources if possible. When an application finishes, all pens will be deleted and their resources freed, eliminating the possibility of 'memory leaks'. However, it is best not to rely on this automatic cleanup because it can lead to double deletion in some circumstances.

There are two mechanisms in recent versions of wxWidgets which make the pen list less useful than it once was. Under Windows, scarce resources are cleaned up internally if they are not being used. Also, a referencing counting mechanism applied to all GDI objects means that some sharing of underlying resources is possible. You don't have to keep track of pointers, working out when it is safe delete a pen, because the referencing counting does it for you. For example, you can set a pen in a device context, and then immediately delete the pen you passed, because the pen is 'copied'.

So you may find it easier to ignore the pen list, and instead create and copy pens as you see fit. If your Windows resource meter suggests your application is using too many resources, you can resort to using GDI lists to share objects explicitly.

The only compelling use for the pen list is for wxWidgets to keep track of pens in order to clean them up on exit. It is also kept for backward compatibility with earlier versions of wxWidgets.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxThePenList

See also

wxPen

Public Member Functions

- wxPenList ()

  Constructor.
- wxPen * FindOrCreatePen (const wxColour &colour, int width=1, wxPenStyle style=wxPENSTYLE_SOLID)

  Finds a pen with the specified attributes and returns it, else creates a new pen, adds it to the pen list, and returns it.

21.516.2 Constructor & Destructor Documentation

wxPenList::wxPenList ( )

Constructor.

The application should not construct its own pen list: use the object pointer wxThePenList.

21.516.3 Member Function Documentation

wxPen* wxPenList::FindOrCreatePen (const wxColour & *colour*, int *width* = 1, wxPenStyle *style* = wxPENSTYLE_SOLID)

Finds a pen with the specified attributes and returns it, else creates a new pen, adds it to the pen list, and returns it.

Parameters

colour	Colour object.
width	Width of pen.
style	Pen style. See wxPenStyle for a list of styles.

21.517 wxPersistenceManager Class Reference

```
#include <wx/persist.h>
```

21.517.1 Detailed Description

Provides support for automatically saving and restoring object properties to persistent storage.

This class is the central element of wxWidgets persistence framework, see Persistent Objects Overview for its overview.

This is a singleton class and its unique instance can be retrieved using Get() method.

Since

2.9.0

Library: wxCore

Public Member Functions

- void DisableSaving ()

  Globally disable saving the persistence object properties.
- void DisableRestoring ()

  Globally disable restoring the persistence object properties.
- template<class T >
  wxPersistentObject * Register (T *obj)

  Register an object with the manager automatically creating a persistence adapter for it.
- wxPersistentObject * Register (void *obj, wxPersistentObject *po)

  Register an object with the manager.
- wxPersistentObject * Find (void *obj) const

  Check if the object is registered and return the associated wxPersistentObject if it is or NULL otherwise.
- void Unregister (void *obj)

  Unregister the object and delete the associated wxPersistentObject.
- void Save (void *obj)

  Save the object properties to persistent storage.
- bool Restore (void *obj)

  Restore the object properties previously saved by Save().
- void SaveAndUnregister (void *obj)

  Combines both Save() and Unregister() calls.
- template<class T >
  bool RegisterAndRestore (T *obj)

  Combines both Register() and Restore() calls.
- bool RegisterAndRestore (void *obj, wxPersistentObject *po)

  Combines both Register() and Restore() calls.

Static Public Member Functions

- static void Set (wxPersistenceManager &manager)

  Set the global persistence manager to use.
- static wxPersistenceManager & Get ()

  Returns the unique persistence manager object.

Protected Member Functions

- wxPersistenceManager ()

  Protected default constructor.
- virtual wxConfigBase * GetConfig () const

  Return the config object to use.
- virtual wxString GetKey (const wxPersistentObject &who, const wxString &name) const

  Return the path to use for saving the setting with the given name for the specified object.

21.517.2 Constructor & Destructor Documentation

wxPersistenceManager::wxPersistenceManager ( ) [protected]

Protected default constructor.

This constructor is only provided for the derived classes, to use an object of this class static Get() method should be called.

21.517.3 Member Function Documentation

void wxPersistenceManager::DisableRestoring ( )

Globally disable restoring the persistence object properties.

By default, restoring properties in Restore() is enabled but this function allows to disable it. This is mostly useful for testing.

See also

DisableSaving()

void wxPersistenceManager::DisableSaving ( )

Globally disable saving the persistence object properties.

By default, saving properties in Save() is enabled but the program may wish to disable if, for example, it detects that it is running on a system which shouldn't be modified in any way and so configuration file (or Windows registry) shouldn't be written to.

See also

DisableRestoring()

wxPersistentObject* wxPersistenceManager::Find (void * *obj*) const

Check if the object is registered and return the associated wxPersistentObject if it is or NULL otherwise.

static wxPersistenceManager& wxPersistenceManager::Get ( ) [static]

Returns the unique persistence manager object.

If Set() hadn't been called before, a default persistence manager implementation is returned.

virtual **wxConfigBase*** **wxPersistenceManager::GetConfig** ( ) **const** [protected], [virtual]

Return the config object to use.

By default the global wxConfig, returned by wxConfigBase::Get(), is used but a derived class could override this function to return a different one if necessary.

Since

2.9.3

virtual **wxString** **wxPersistenceManager::GetKey** (**const wxPersistentObject &** *who,* **const wxString &** *name*) **const** [protected], [virtual]

Return the path to use for saving the setting with the given name for the specified object.

Notice that the *name* argument is the name of the setting, not the name of the object itself which can be retrieved with its GetName() method.

This method can be overridden by a derived class to change where in wxConfig the different options are stored. By default, all settings of the persistent controls are stored under "Persistent_Options" group and grouped by control type (e.g. "Window" for top level windows or "Splitter") and name, so that the position of a splitter called "sep" could be stored under "Persistent_Options/Splitter/sep/Position" key.

Since

2.9.3

template<class T> **wxPersistentObject*** **wxPersistenceManager::Register** (**T** * *obj*)

Register an object with the manager automatically creating a persistence adapter for it.

This is equivalent to calling Register(void *, wxPersistentObject *) with wxCreatePersistentObject(obj) as the second argument.

Parameters

obj	The object to register. wxCreatePersistentObject() overload must be defined for the objects of this class.

wxPersistentObject* **wxPersistenceManager::Register** (**void** * *obj,* **wxPersistentObject** * *po*)

Register an object with the manager.

Note that registering the object doesn't do anything except allowing to call Restore() for it later. If you want to register the object and restore its properties, use RegisterAndRestore().

The manager takes ownership of *po* and will delete it when it is unregistered.

Parameters

obj	The object to register.
po	The wxPersistentObject to use for saving and restoring this object properties.

template<class T> **bool** **wxPersistenceManager::RegisterAndRestore** (**T** * *obj*)

Combines both Register() and Restore() calls.

bool wxPersistenceManager::RegisterAndRestore (void * *obj*, wxPersistentObject * *po*)

Combines both Register() and Restore() calls.

bool wxPersistenceManager::Restore (void * *obj*)

Restore the object properties previously saved by Save().

This method does nothing if DisableRestoring() had been called.

Parameters

obj	An object previously registered with Register().

Returns

true if the object properties were restored or false if nothing was found to restore or the saved settings were invalid.

See also

RegisterAndRestore()

void wxPersistenceManager::Save (void * *obj*)

Save the object properties to persistent storage.

This method does nothing if DisableSaving() had been called.

Parameters

obj	An object previously registered with Register().

See also

SaveAndUnregister()

void wxPersistenceManager::SaveAndUnregister (void * *obj*)

Combines both Save() and Unregister() calls.

static void wxPersistenceManager::Set (wxPersistenceManager & *manager*) `[static]`

Set the global persistence manager to use.

Call this method to specify a non-default persistence manager to use. It should usually be called very early (e.g. in wxApp-derived class constructor or in the beginning of overridden wxApp::OnInit()) to affect creation of all persistent controls and the object passed to it must have a lifetime long enough to be still alive when the persistent controls are destroyed and need it to save their state so typically this would be a global or a wxApp member.

Since

2.9.3

void wxPersistenceManager::Unregister (void * *obj*)

Unregister the object and delete the associated wxPersistentObject.

For the persistent windows this is done automatically (via SaveAndUnregister()) when the window is destroyed so you only need to call this function explicitly if you are using custom persistent objects or if you want to prevent the object properties from being saved.

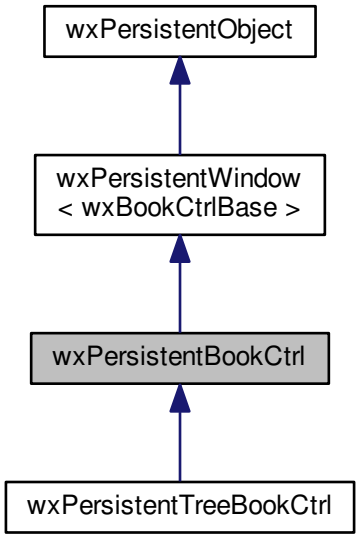
Parameters

obj	An object previously registered with Register().

21.518 wxPersistentBookCtrl Class Reference

`#include <wx/persist/bookctrl.h>`

Inheritance diagram for wxPersistentBookCtrl:



21.518.1 Detailed Description

Persistence adapter for wxBookCtrlBase.

This adapter handles the selected page of wxBookCtrlBase, i.e. it saves its value when the associated book control is destroyed and restores it when it is recreated.

See also

wxPersistentTreeBookCtrl

Public Member Functions

- wxPersistentBookCtrl (wxBookCtrlBase *book)

Constructor.

- virtual void Save () const

  Save the currently selected page index.

- virtual bool Restore ()

  Restore the selected page index.

Additional Inherited Members

21.518.2 Constructor & Destructor Documentation

wxPersistentBookCtrl::wxPersistentBookCtrl (wxBookCtrlBase * *book*)

Constructor.

Parameters

book	The associated book control.

21.518.3 Member Function Documentation

virtual bool wxPersistentBookCtrl::Restore ( ) [virtual]

Restore the selected page index.

The book control must be initialized before calling this function, i.e. all of its pages should be already added to it – otherwise restoring the selection has no effect.

Implements wxPersistentObject.

Reimplemented in wxPersistentTreeBookCtrl.

virtual void wxPersistentBookCtrl::Save ( ) const [virtual]

Save the currently selected page index.

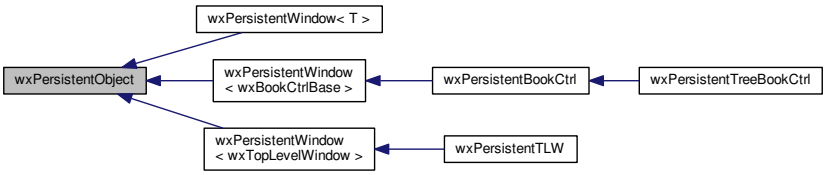
Implements wxPersistentObject.

Reimplemented in wxPersistentTreeBookCtrl.

21.519 wxPersistentObject Class Reference

```
#include <wx/persist.h>
```

Inheritance diagram for wxPersistentObject:



21.519.1 Detailed Description

Base class for persistent object adapters.

wxWidgets persistence framework is non-intrusive, i.e. can work with the classes which have no relationship to nor knowledge of it. To allow this, an intermediate persistence adapter is used: this is just a simple object which provides the methods used by wxPersistenceManager to save and restore the object properties and implements them using the concrete class methods.

You may derive your own classes from wxPersistentObject to implement persistence support for your common classes, see Defining Custom Persistent Windows.

See also

wxPersistentWindow<>

Public Member Functions

- wxPersistentObject (void *obj)

  Constructor takes the object which we're associated with.
- virtual ~wxPersistentObject ()

  Trivial but virtual destructor.
- void * GetObject () const

  Return the associated object.

Methods to be implemented in the derived classes.

Notice that these methods are only used by wxPersistenceManager normally and shouldn't be called directly.

- virtual void Save () const =0

  Save the object properties.
- virtual bool Restore ()=0

  Restore the object properties.
- virtual wxString GetKind () const =0

  Returns the string uniquely identifying the objects supported by this adapter.
- virtual wxString GetName () const =0

  Returns the string uniquely identifying the object we're associated with among all the other objects of the same type.

Protected Member Functions

- template<typename T >
  bool SaveValue (const wxString &name, T value) const

  Save the specified value using the given name.
- template<typename T >
  bool RestoreValue (const wxString &name, T *value)

  Restore the value saved by Save().

21.519.2 Constructor & Destructor Documentation

wxPersistentObject::wxPersistentObject (void * *obj*)

Constructor takes the object which we're associated with.

This object must have life-time greater than ours as we keep a pointer to it.

virtual wxPersistentObject::~wxPersistentObject ( ) [virtual]

Trivial but virtual destructor.

21.519.3 Member Function Documentation

virtual wxString wxPersistentObject::GetKind ( ) const [pure virtual]

Returns the string uniquely identifying the objects supported by this adapter.

This method is called from SaveValue() and RestoreValue() and normally returns some short (but not too cryptic) strings, e.g. `"Checkbox"`.

virtual wxString wxPersistentObject::GetName ( ) const [pure virtual]

Returns the string uniquely identifying the object we're associated with among all the other objects of the same type.

This method is used together with GetKind() to construct the unique full name of the object in e.g. a configuration file.

Implemented in wxPersistentWindow< T >, wxPersistentWindow< wxBookCtrlBase >, and wxPersistentWindow< wxTopLevelWindow >.

void* wxPersistentObject::GetObject ( ) const

Return the associated object.

virtual bool wxPersistentObject::Restore ( ) [pure virtual]

Restore the object properties.

The implementation of this method should use RestoreValue().

Implemented in wxPersistentBookCtrl, wxPersistentTreeBookCtrl, and wxPersistentTLW.

template<typename T > bool wxPersistentObject::RestoreValue (const wxString & *name*, T * *value*) [protected]

Restore the value saved by Save().

Parameters

name	The same name as was used by Save().
value	Non-NULL pointer which will be filled with the value if it was read successfully or not modified if it wasn't.

Returns

true if the value was successfully read or false if it was not found or an error occurred.

virtual void wxPersistentObject::Save ( ) const [pure virtual]

Save the object properties.

The implementation of this method should use SaveValue().

Implemented in wxPersistentBookCtrl, wxPersistentTLW, and wxPersistentTreeBookCtrl.

template<typename T > bool wxPersistentObject::SaveValue (const wxString & *name,* T *value*) const [protected]

Save the specified value using the given name.

Parameters

name	The name of the value in the configuration file.
value	The value to save, currently must be a type supported by wxConfig.

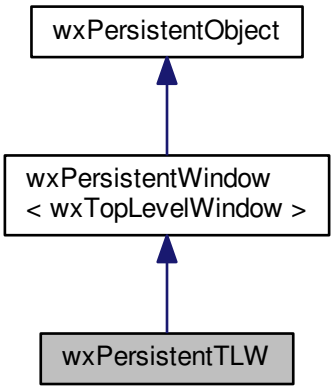
Returns

true if the value was saved or false if an error occurred.

21.520 wxPersistentTLW Class Reference

`#include <wx/persist/toplevel.h>`

Inheritance diagram for wxPersistentTLW:



21.520.1 Detailed Description

Persistence adapter for wxTopLevelWindow.

This adapter saves and restores the geometry (i.e. position and size) and the state (iconized, maximized or normal) of top level windows. It can be used with both wxFrame and wxDialog.

Note that it does *not* save nor restore the window visibility.

Public Member Functions

- wxPersistentTLW (wxTopLevelWindow *book)

  Constructor.

- virtual void Save () const

  Save the current window geometry.

- virtual bool Restore ()

  Restore the window geometry.

Additional Inherited Members

21.520.2 Constructor & Destructor Documentation

wxPersistentTLW::wxPersistentTLW (wxTopLevelWindow * *book*)

Constructor.

Parameters

book	The associated window.

21.520.3 Member Function Documentation

virtual bool wxPersistentTLW::Restore ( ) [virtual]

Restore the window geometry.

Implements wxPersistentObject.

virtual void wxPersistentTLW::Save ( ) const [virtual]

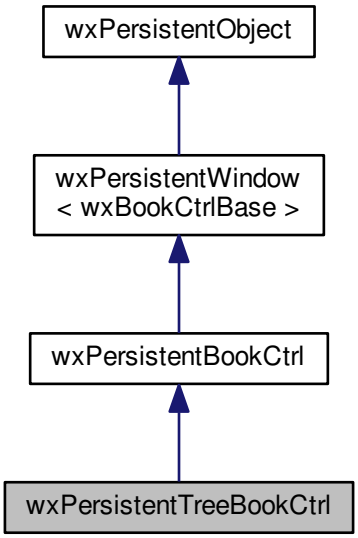
Save the current window geometry.

Implements wxPersistentObject.

21.521 wxPersistentTreeBookCtrl Class Reference

`#include <wx/persist/treebook.h>`

Inheritance diagram for wxPersistentTreeBookCtrl:



21.521.1 Detailed Description

Persistence adapter for wxTreebook.

This adapter saves and restores the expanded branches of the wxTreeCtrl used by wxTreebook, in addition to saving and restoring the selection as implemented by the base wxPersistentBookCtrl class.

Public Member Functions

- wxPersistentTreeBookCtrl (wxTreebook *book)

  Constructor.

- virtual void Save () const

  Save the currently opened branches.

- virtual bool Restore ()

  Restore the opened branches.

Additional Inherited Members

21.521.2 Constructor & Destructor Documentation

wxPersistentTreeBookCtrl::wxPersistentTreeBookCtrl (wxTreebook * *book*)

Constructor.

Parameters

book	The associated tree book control.

21.521.3 Member Function Documentation

virtual bool wxPersistentTreeBookCtrl::Restore ( ) [virtual]

Restore the opened branches.

The book control must be initialized before calling this function, i.e. all of its pages should be already added to it.

Reimplemented from wxPersistentBookCtrl.

virtual void wxPersistentTreeBookCtrl::Save ( ) const [virtual]

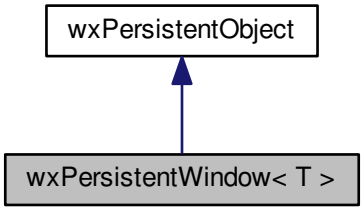
Save the currently opened branches.

Reimplemented from wxPersistentBookCtrl.

21.522 wxPersistentWindow< T > Class Template Reference

`#include <wx/persist/window.h>`

Inheritance diagram for wxPersistentWindow< T >:



21.522.1 Detailed Description

template<class T>class wxPersistentWindow< T >

Base class for persistent windows.

Compared to wxPersistentObject this class does three things:

- Most importantly, wxPersistentWindow catches wxWindowDestroyEvent generated when the window is destroyed and saves its properties automatically when it happens.
- It implements GetName() using wxWindow::GetName() so that the derived classes don't need to do it.
- It adds a convenient wxPersistentWindow::Get() accessor returning the window object of the correct type.

Public Types

- typedef T WindowType

  The type of the associated window.

Public Member Functions

- wxPersistentWindow (WindowType *win)

  Constructor for a persistent window object.
- WindowType * Get () const
- virtual wxString GetName () const

  Implements the base class pure virtual method using wxWindow::GetName().

Additional Inherited Members

21.522.2 Member Typedef Documentation

template<class T> typedef T wxPersistentWindow< T >::WindowType

The type of the associated window.

21.522.3 Constructor & Destructor Documentation

template<class T> wxPersistentWindow< T >::wxPersistentWindow (WindowType * *win*)

Constructor for a persistent window object.

The constructor uses wxEvtHandler::Connect() to catch wxWindowDestroyEvent generated when the window is destroyed and call wxPersistenceManager::SaveAndUnregister() when this happens. This ensures that the window properties are saved and that this object itself is deleted when the window is.

21.522.4 Member Function Documentation

template<class T> WindowType* wxPersistentWindow< T >::Get ( ) const `[inline]`

template<class T> virtual wxString wxPersistentWindow< T >::GetName ( ) const `[virtual]`

Implements the base class pure virtual method using wxWindow::GetName().

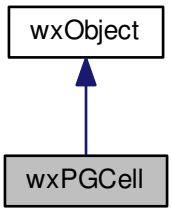
Notice that window names are usually not unique while this function must return a unique (at least among the objects of this type) string. Because of this you need to specify a non-default window name in its constructor when creating it or explicitly call wxWindow::SetName() before saving or restoring persistent properties.

Implements wxPersistentObject.

21.523 wxPGCell Class Reference

`#include <wx/propgrid/property.h>`

Inheritance diagram for wxPGCell:



21.523.1 Detailed Description

Base class for wxPropertyGrid cell information.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPGCell ()
- wxPGCell (const wxPGCell &other)
- wxPGCell (const wxString &text, const wxBitmap &bitmap=wxNullBitmap, const wxColour &fgCol=wxNull↩ Colour, const wxColour &bgCol=wxNullColour)
- virtual ∼wxPGCell ()
- const wxPGCellData ∗ GetData () const
- bool HasText () const

  Returns true if this cell has custom text stored within.
- void MergeFrom (const wxPGCell &srcCell)

  Merges valid data from srcCell into this.
- void SetText (const wxString &text)
- void SetBitmap (const wxBitmap &bitmap)
- void SetFgCol (const wxColour &col)
- void SetFont (const wxFont &font)

  Sets font of the cell.
- void SetBgCol (const wxColour &col)
- const wxString & GetText () const
- const wxBitmap & GetBitmap () const
- const wxColour & GetFgCol () const
- const wxFont & GetFont () const

  Returns font of the cell.
- const wxColour & GetBgCol () const
- wxPGCell & operator= (const wxPGCell &other)

Additional Inherited Members

21.523.2 Constructor & Destructor Documentation

wxPGCell::wxPGCell ( )

wxPGCell::wxPGCell (const wxPGCell & *other*)

wxPGCell::wxPGCell (const wxString & *text*, const wxBitmap & *bitmap* = wxNullBitmap, const wxColour & *fgCol* = wxNullColour, const wxColour & *bgCol* = wxNullColour)

virtual wxPGCell::∼wxPGCell ( ) `[virtual]`

21.523.3 Member Function Documentation

const wxColour& wxPGCell::GetBgCol ( ) const

const wxBitmap& wxPGCell::GetBitmap ( ) const

const wxPGCellData∗ wxPGCell::GetData ( ) const

const wxColour& wxPGCell::GetFgCol ( ) const

const wxFont& wxPGCell::GetFont ( ) const

Returns font of the cell.

If no specific font is set for this cell, then the font will be invalid.

const wxString& wxPGCell::GetText ( ) const

bool wxPGCell::HasText ( ) const

Returns true if this cell has custom text stored within.

void wxPGCell::MergeFrom (const wxPGCell & *srcCell*)

Merges valid data from srcCell into this.

wxPGCell& wxPGCell::operator= (const wxPGCell & *other*)

void wxPGCell::SetBgCol (const wxColour & *col*)

void wxPGCell::SetBitmap (const wxBitmap & *bitmap*)

void wxPGCell::SetFgCol (const wxColour & *col*)

void wxPGCell::SetFont (const wxFont & *font*)

Sets font of the cell.

Remarks

Because wxPropertyGrid does not support rows of different height, it makes little sense to change size of the font. Therefore it is recommended to use return value of wxPropertyGrid::GetFont() or wxPropertyGrid::GetCaptionFont() as a basis for the font that, after modifications, is passed to this member function.

void wxPGCell::SetText (const wxString & *text*)

21.524 wxPGChoices Class Reference

`#include <wx/propgrid/property.h>`

21.524.1 Detailed Description

Helper class for managing choices of wxPropertyGrid properties.

Each entry can have label, value, bitmap, text colour, and background colour.

wxPGChoices uses reference counting, similar to other wxWidgets classes. This means that assignment operator and copy constructor only copy the reference and not the actual data. Use Copy() member function to create a real copy.

Remarks

If you do not specify value for entry, index is used.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Types

- typedef long ValArrItem

Public Member Functions

- wxPGChoices ()

  Default constructor.
- wxPGChoices (const wxPGChoices &a)

  Copy constructor, uses reference counting.
- wxPGChoices (const wxChar **labels, const long *values=NULL)

  Constructor.
- wxPGChoices (const wxArrayString &labels, const wxArrayInt &values=wxArrayInt())

  Constructor.
- wxPGChoices (wxPGChoicesData *data)

  Constructor.
- ~wxPGChoices ()

  Destructor.
- void Add (const wxChar **labels, const ValArrItem *values=NULL)

  Adds to current.
- void Add (const wxArrayString &arr, const wxArrayInt &arrint)

  Version that works with wxArrayString and wxArrayInt.
- wxPGChoiceEntry & Add (const wxString &label, int value=wxPG_INVALID_VALUE)

  Adds single item.
- wxPGChoiceEntry & Add (const wxString &label, const wxBitmap &bitmap, int value=wxPG_INVALID_VA↩ LUE)

  Adds a single item, with bitmap.
- wxPGChoiceEntry & Add (const wxPGChoiceEntry &entry)

  Adds a single item with full entry information.
- wxPGChoiceEntry & AddAsSorted (const wxString &label, int value=wxPG_INVALID_VALUE)

  Adds single item, sorted.
- void Assign (const wxPGChoices &a)

  Assigns choices data, using reference counting.
- void AssignData (wxPGChoicesData *data)

  Assigns data from another set of choices.
- void Clear ()

  Deletes all items.
- wxPGChoices Copy () const

  Returns a real copy of the choices.
- const wxString & GetLabel (unsigned int ind) const

  Returns label of item.
- unsigned int GetCount () const

  Returns number of items.
- int GetValue (unsigned int ind) const

  Returns value of item;.
- wxArrayInt GetValuesForStrings (const wxArrayString &strings) const

  Returns array of values matching the given strings.
- wxArrayInt GetIndicesForStrings (const wxArrayString &strings, wxArrayString *unmatched=NULL) const

  Returns array of indices matching given strings.
- int Index (const wxString &label) const

  Returns index of item with given label.

- int Index (int val) const

  Returns index of item with given value.
- wxPGChoiceEntry & Insert (const wxString &label, int index, int value=wxPG_INVALID_VALUE)

  Inserts single item.
- wxPGChoiceEntry & Insert (const wxPGChoiceEntry &entry, int index)

  Inserts a single item with full entry information.
- bool IsOk () const

  Returns false if this is a constant empty set of choices, which should not be modified.
- const wxPGChoiceEntry & Item (unsigned int i) const

  Returns item at given index.
- wxPGChoiceEntry & Item (unsigned int i)

  Returns item at given index.
- void RemoveAt (size_t nIndex, size_t count=1)

  Removes count items starting at position nIndex.
- void Set (const wxChar **labels, const long *values=NULL)

  Sets contents from lists of strings and values.
- void Set (const wxArrayString &labels, const wxArrayInt &values=wxArrayInt())

  Sets contents from lists of strings and values.
- void AllocExclusive ()

  Creates exclusive copy of current choices.
- wxArrayString GetLabels () const

  Returns array of choice labels.
- void operator= (const wxPGChoices &a)
- wxPGChoiceEntry & operator[] (unsigned int i)
- const wxPGChoiceEntry & operator[] (unsigned int i) const

21.524.2 Member Typedef Documentation

typedef long wxPGChoices::ValArrItem

21.524.3 Constructor & Destructor Documentation

wxPGChoices::wxPGChoices ( )

Default constructor.

wxPGChoices::wxPGChoices (const wxPGChoices & *a*)

Copy constructor, uses reference counting.

To create a real copy, use Copy() member function instead.

wxPGChoices::wxPGChoices (const wxChar ** *labels*, const long * *values* = NULL)

Constructor.

wxPGChoices::wxPGChoices (const wxArrayString & *labels*, const wxArrayInt & *values* = wxArrayInt())

Constructor.

wxPGChoices::wxPGChoices (wxPGChoicesData * *data*)

Constructor.

wxPGChoices::~wxPGChoices ( )

Destructor.

21.524.4 Member Function Documentation

void wxPGChoices::Add (const wxChar ** *labels*, const wxArrItem * *values* = NULL)

Adds to current.

If did not have own copies, creates them now. If was empty, identical to set except that creates copies.

void wxPGChoices::Add (const wxArrayString & *arr*, const wxArrayInt & *arrint*)

Version that works with wxArrayString and wxArrayInt.

wxPGChoiceEntry& wxPGChoices::Add (const wxString & *label*, int *value* = wxPG_INVALID_VALUE)

Adds single item.

wxPGChoiceEntry& wxPGChoices::Add (const wxString & *label*, const wxBitmap & *bitmap*, int *value* = wxPG_INVALID_VALUE)

Adds a single item, with bitmap.

wxPGChoiceEntry& wxPGChoices::Add (const wxPGChoiceEntry & *entry*)

Adds a single item with full entry information.

wxPGChoiceEntry& wxPGChoices::AddAsSorted (const wxString & *label*, int *value* = wxPG_INVALID_VALUE)

Adds single item, sorted.

void wxPGChoices::AllocExclusive ( )

Creates exclusive copy of current choices.

void wxPGChoices::Assign (const wxPGChoices & *a*)

Assigns choices data, using reference counting.

To create a real copy, use Copy() member function instead.

void wxPGChoices::AssignData (wxPGChoicesData * *data*)

Assigns data from another set of choices.

void wxPGChoices::Clear ( )

Deletes all items.

wxPGChoices wxPGChoices::Copy ( ) const

Returns a real copy of the choices.

unsigned int wxPGChoices::GetCount ( ) const

Returns number of items.

wxArrayInt wxPGChoices::GetIndicesForStrings (const wxArrayString & *strings,* wxArrayString * *unmatched* = NULL) const

Returns array of indices matching given strings.

Unmatching strings are added to 'unmatched', if not NULL.

const wxString& wxPGChoices::GetLabel (unsigned int *ind*) const

Returns label of item.

wxArrayString wxPGChoices::GetLabels ( ) const

Returns array of choice labels.

int wxPGChoices::GetValue (unsigned int *ind*) const

Returns value of item;.

wxArrayInt wxPGChoices::GetValuesForStrings (const wxArrayString & *strings*) const

Returns array of values matching the given strings.

Unmatching strings result in wxPG_INVALID_VALUE entry in array.

int wxPGChoices::Index (const wxString & *label*) const

Returns index of item with given label.

int wxPGChoices::Index (int *val*) const

Returns index of item with given value.

wxPGChoiceEntry& wxPGChoices::Insert (const wxString & *label,* int *index,* int *value* = wxPG_INVALID_VALUE)

Inserts single item.

wxPGChoiceEntry& wxPGChoices::Insert (const wxPGChoiceEntry & *entry,* int *index*)

Inserts a single item with full entry information.

bool wxPGChoices::IsOk ( ) const

Returns false if this is a constant empty set of choices, which should not be modified.

const wxPGChoiceEntry& wxPGChoices::Item (unsigned int *i*) const

Returns item at given index.

wxPGChoiceEntry& wxPGChoices::Item (unsigned int *i*)

Returns item at given index.

void wxPGChoices::operator= (const wxPGChoices & *a*)

wxPGChoiceEntry& wxPGChoices::operator[] (unsigned int *i*)

const wxPGChoiceEntry& wxPGChoices::operator[] (unsigned int *i*) const

void wxPGChoices::RemoveAt (size_t *nIndex,* size_t *count* = `1`)

Removes count items starting at position nIndex.

void wxPGChoices::Set (const wxChar ** *labels,* const long * *values* = `NULL`)**

Sets contents from lists of strings and values.

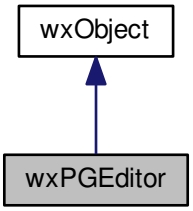
void wxPGChoices::Set (const wxArrayString & *labels,* const wxArrayInt & *values* = wxArrayInt `()`)

Sets contents from lists of strings and values.

21.525 wxPGEditor Class Reference

`#include <wx/propgrid/editors.h>`

Inheritance diagram for wxPGEditor:



21.525.1 Detailed Description

Base class for custom wxPropertyGrid editors.

Remarks

- Names of built-in property editors are: TextCtrl, Choice, ComboBox, CheckBox, TextCtrlAndButton, and ChoiceAndButton. Additional editors include SpinCtrl and DatePickerCtrl, but using them requires calling wxPropertyGrid::RegisterAdditionalEditors() prior use.
- Pointer to built-in editor is available as wxPGEditor_EditorName (eg. wxPGEditor_TextCtrl).
- Before you start using new editor you just created, you need to register it using static function wxPropertyGrid::RegisterEditorClass(), with code like this:

```
wxPGEditor* editorPointer = wxPropertyGrid::RegisterEditorClass
      (new MyEditorClass(), "MyEditor");
```

  After that, wxPropertyGrid will take ownership of the given object, but you should still store editorPointer somewhere, so you can pass it to wxPGProperty::SetEditor(), or return it from wxPGEditor::DoGetEditorClass().

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPGEditor ()

  Constructor.
- virtual ~wxPGEditor ()

  Destructor.
- virtual wxString GetName () const

  Returns pointer to the name of the editor.
- virtual wxPGWindowList CreateControls (wxPropertyGrid *propgrid, wxPGProperty *property, const wxPoint &pos, const wxSize &size) const =0

  Instantiates editor controls.
- virtual void UpdateControl (wxPGProperty *property, wxWindow *ctrl) const =0

  Loads value from property to the control.

- virtual void DrawValue (wxDC &dc, const wxRect &rect, wxPGProperty *property, const wxString &text) const

  Draws value for given property.
- virtual bool OnEvent (wxPropertyGrid *propgrid, wxPGProperty *property, wxWindow *wnd_primary, wxEvent &event) const =0

  Handles events.
- virtual bool GetValueFromControl (wxVariant &variant, wxPGProperty *property, wxWindow *ctrl) const

  Returns value from control, via parameter 'variant'.
- virtual void SetValueToUnspecified (wxPGProperty *property, wxWindow *ctrl) const =0

  Sets value in control to unspecified.
- virtual void SetControlAppearance (wxPropertyGrid *pg, wxPGProperty *property, wxWindow *ctrl, const wxPGCell &appearance, const wxPGCell &oldAppearance, bool unspecified) const

  Called by property grid to set new appearance for the control.
- virtual void SetControlStringValue (wxPGProperty *property, wxWindow *ctrl, const wxString &txt) const

  Sets control's value specifically from string.
- virtual void SetControlIntValue (wxPGProperty *property, wxWindow *ctrl, int value) const

  Sets control's value specifically from int (applies to choice etc.).
- virtual int InsertItem (wxWindow *ctrl, const wxString &label, int index) const

  Inserts item to existing control.
- virtual void DeleteItem (wxWindow *ctrl, int index) const

  Deletes item from existing control.
- virtual void OnFocus (wxPGProperty *property, wxWindow *wnd) const

  Extra processing when control gains focus.
- virtual bool CanContainCustomImage () const

  Returns true if control itself can contain the custom image.

Additional Inherited Members

21.525.2 Constructor & Destructor Documentation

wxPGEditor::wxPGEditor ( )

Constructor.

virtual wxPGEditor::~wxPGEditor ( ) `[virtual]`

Destructor.

21.525.3 Member Function Documentation

virtual bool wxPGEditor::CanContainCustomImage ( ) const `[virtual]`

Returns true if control itself can contain the custom image.

Default implementation returns false.

virtual wxPGWindowList wxPGEditor::CreateControls (wxPropertyGrid * *propgrid*, wxPGProperty * *property*, const wxPoint & *pos*, const wxSize & *size*) const `[pure virtual]`

Instantiates editor controls.

Parameters

propgrid	wxPropertyGrid to which the property belongs (use as parent for control).
property	Property for which this method is called.
pos	Position, inside wxPropertyGrid, to create control(s) to.
size	Initial size for control(s).

Remarks

- Primary control shall use id wxPG_SUBID1, and secondary (button) control shall use wxPG_SUBID2.
- Unlike in previous version of wxPropertyGrid, it is no longer necessary to call wxEvtHandler::Connect() for interesting editor events. Instead, all events from control are now automatically forwarded to wxPGEditor::OnEvent() and wxPGProperty::OnEvent().

virtual void wxPGEditor::DeleteItem (wxWindow * *ctrl,* int *index*) const `[virtual]`

Deletes item from existing control.

Default implementation does nothing.

virtual void wxPGEditor::DrawValue (wxDC & *dc,* const wxRect & *rect,* wxPGProperty * *property,* const wxString & *text*) const `[virtual]`

Draws value for given property.

virtual wxString wxPGEditor::GetName ( ) const `[virtual]`

Returns pointer to the name of the editor.

For example, wxPGEditor_TextCtrl has name "TextCtrl". If you dont' need to access your custom editor by string name, then you do not need to implement this function.

virtual bool wxPGEditor::GetValueFromControl (wxVariant & *variant,* wxPGProperty * *property,* wxWindow * *ctrl*) const `[virtual]`

Returns value from control, via parameter 'variant'.

Usually ends up calling property's StringToValue() or IntToValue(). Returns true if value was different.

virtual int wxPGEditor::InsertItem (wxWindow * *ctrl,* const wxString & *label,* int *index*) const `[virtual]`

Inserts item to existing control.

Index -1 means end of list. Default implementation does nothing. Returns index of item added.

virtual bool wxPGEditor::OnEvent (wxPropertyGrid * *propgrid,* wxPGProperty * *property,* wxWindow * *wnd_primary,* wxEvent & *event*) const `[pure virtual]`

Handles events.

Returns true if value in control was modified (see wxPGProperty::OnEvent() for more information).

Remarks

wxPropertyGrid will automatically unfocus the editor when `wxEVT_TEXT_ENTER` is received and when it results in property value being modified. This happens regardless of editor type (ie. behaviour is same for any wxTextCtrl and wxComboBox based editor).

virtual void wxPGEditor::OnFocus (wxPGProperty * *property,* wxWindow * *wnd*) const `[virtual]`

Extra processing when control gains focus.

For example, wxTextCtrl based controls should select all text.

virtual void wxPGEditor::SetControlAppearance (wxPropertyGrid * *pg,* wxPGProperty * *property,* wxWindow * *ctrl,* const wxPGCell & *appearance,* const wxPGCell & *oldAppearance,* bool *unspecified*) const `[virtual]`

Called by property grid to set new appearance for the control.

Default implementation sets foreground colour, background colour, font, plus text for wxTextCtrl and wxComboCtrl.

The parameter *appearance* represents the new appearance to be applied.

The parameter *oldAppearance* is the previously applied appearance. Used to detect which control attributes need to be changed (e.g. so we only change background colour if really needed).

Finally, the parameter *unspecified* if true tells this function that the new appearance represents an unspecified property value.

virtual void wxPGEditor::SetControlIntValue (wxPGProperty * *property,* wxWindow * *ctrl,* int *value*) const `[virtual]`

Sets control's value specifically from int (applies to choice etc.).

virtual void wxPGEditor::SetControlStringValue (wxPGProperty * *property,* wxWindow * *ctrl,* const wxString & *txt*) const `[virtual]`

Sets control's value specifically from string.

virtual void wxPGEditor::SetValueToUnspecified (wxPGProperty * *property,* wxWindow * *ctrl*) const `[pure virtual]`

Sets value in control to unspecified.

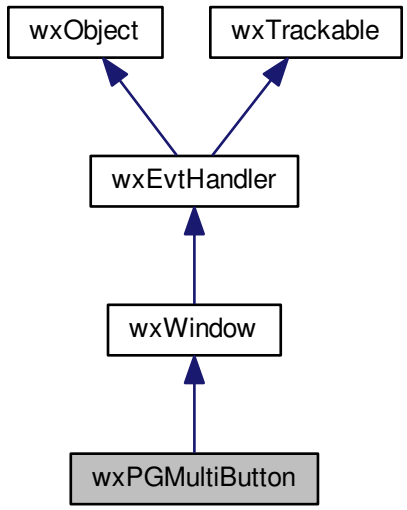
virtual void wxPGEditor::UpdateControl (wxPGProperty * *property,* wxWindow * *ctrl*) const `[pure virtual]`

Loads value from property to the control.

21.526 wxPGMultiButton Class Reference

`#include <wx/propgrid/editors.h>`

Inheritance diagram for wxPGMultiButton:



21.526.1 Detailed Description

This class can be used to have multiple buttons in a property editor.

You will need to create a new property editor class, override CreateControls, and have it return wxPGMultiButton instance in wxPGWindowList::SetSecondary().

For instance, here we add three buttons to a TextCtrl editor:

```
#include <wx/propgrid/editors.h>

class wxSampleMultiButtonEditor : public wxPGTextCtrlEditor
{
    wxDECLARE_DYNAMIC_CLASS(wxSampleMultiButtonEditor);

public:
    wxSampleMultiButtonEditor() {}
    virtual ~wxSampleMultiButtonEditor() {}

    virtual wxString GetName() const { return "SampleMultiButtonEditor"; }

    virtual wxPGWindowList CreateControls( wxPropertyGrid* propGrid,
                                           wxPGProperty* property,
                                           const wxPoint& pos,
                                           const wxSize& sz ) const;
    virtual bool OnEvent( wxPropertyGrid* propGrid,
                          wxPGProperty* property,
                          wxWindow* ctrl,
                          wxEvent& event ) const;
};

wxIMPLEMENT_DYNAMIC_CLASS(wxSampleMultiButtonEditor, wxPGTextCtrlEditor);

wxPGWindowList wxSampleMultiButtonEditor::CreateControls( wxPropertyGrid* propGrid,
                                                          wxPGProperty* property,
                                                          const wxPoint& pos,
                                                          const wxSize& sz ) const
{
    // Create and populate buttons-subwindow
    wxPGMultiButton* buttons = new wxPGMultiButton( propGrid, sz );

    // Add two regular buttons
    buttons->Add( "..." );
    buttons->Add( "A" );
```

```
    // Add a bitmap button
    buttons->Add( wxArtProvider::GetBitmap(
      wxART_FOLDER) );

    // Create the 'primary' editor control (textctrl in this case)
    wxPGWindowList wndList = wxPGTextCtrlEditor::CreateControls
                             ( propGrid, property, pos,
                               buttons->GetPrimarySize() );

    // Finally, move buttons-subwindow to correct position and make sure
    // returned wxPGWindowList contains our custom button list.
    buttons->Finalize(propGrid, pos);

    wndList.SetSecondary( buttons );
    return wndList;
}

bool wxSampleMultiButtonEditor::OnEvent( wxPropertyGrid* propGrid,
                                         wxPGProperty* property,
                                         wxWindow* ctrl,
                                         wxEvent& event ) const
{
    if ( event.GetEventType() == wxEVT_BUTTON )
    {
        wxPGMultiButton* buttons = (wxPGMultiButton*) propGrid->
      GetEditorControlSecondary();

        if ( event.GetId() == buttons->GetButtonId(0) )
        {
            // Do something when the first button is pressed
            // Return true if the action modified the value in editor.
            ...
        }
        if ( event.GetId() == buttons->GetButtonId(1) )
        {
            // Do something when the second button is pressed
            ...
        }
        if ( event.GetId() == buttons->GetButtonId(2) )
        {
            // Do something when the third button is pressed
            ...
        }
    }
    return wxPGTextCtrlEditor::OnEvent(propGrid, property, ctrl, event);
}
```

Further to use this editor, code like this can be used:

```
// Register editor class - needs only to be called once
wxPGEditor* multiButtonEditor = new wxSampleMultiButtonEditor();
wxPropertyGrid::RegisterEditorClass( multiButtonEditor );

// Insert the property that will have multiple buttons
propGrid->Append( new wxLongStringProperty("MultipleButtons", wxPG_LABEL) );

// Change property to use editor created in the previous code segment
propGrid->SetPropertyEditor( "MultipleButtons", multiButtonEditor );
```

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPGMultiButton (wxPropertyGrid *pg, const wxSize &sz)

  Constructor.
- virtual ~wxPGMultiButton ()

  Destructor.
- void Add (const wxString &label, int id=-2)

  Adds new button, with given label.
- void Add (const wxBitmap &bitmap, int id=-2)

Adds new bitmap button.

- void Finalize (wxPropertyGrid *propGrid, const wxPoint &pos)

  Call this in CreateControls() of your custom editor class after all buttons have been added.
- wxWindow * GetButton (unsigned int i)

  Returns pointer to one of the buttons.
- int GetButtonId (unsigned int i) const

  Returns Id of one of the buttons.
- unsigned int GetCount ()

  Returns number of buttons.
- wxSize GetPrimarySize () const

  Returns size of primary editor control, as appropriately reduced by number of buttons present.

Additional Inherited Members

21.526.2 Constructor & Destructor Documentation

wxPGMultiButton::wxPGMultiButton (wxPropertyGrid * *pg*, const wxSize & *sz*)

Constructor.

virtual wxPGMultiButton::~wxPGMultiButton ( ) [inline], [virtual]

Destructor.

21.526.3 Member Function Documentation

void wxPGMultiButton::Add (const wxString & *label*, int *id* = -2)

Adds new button, with given label.

void wxPGMultiButton::Add (const wxBitmap & *bitmap*, int *id* = -2)

Adds new bitmap button.

void wxPGMultiButton::Finalize (wxPropertyGrid * *propGrid*, const wxPoint & *pos*)

Call this in CreateControls() of your custom editor class after all buttons have been added.

Parameters

propGrid	wxPropertyGrid given in CreateControls().
pos	wxPoint given in CreateControls().

wxWindow* wxPGMultiButton::GetButton (unsigned int *i*)

Returns pointer to one of the buttons.

int wxPGMultiButton::GetButtonId (unsigned int *i*) const

Returns Id of one of the buttons.

This is utility function to be used in event handlers.

unsigned int wxPGMultiButton::GetCount ( )

Returns number of buttons.

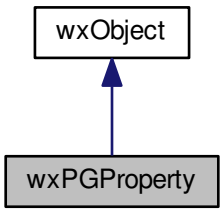
wxSize wxPGMultiButton::GetPrimarySize ( ) const

Returns size of primary editor control, as appropriately reduced by number of buttons present.

21.527 wxPGProperty Class Reference

`#include <wx/propgrid/property.h>`

Inheritance diagram for wxPGProperty:



21.527.1 Detailed Description

wxPGProperty is base class for all wxPropertyGrid properties.

In sections below we cover few related topics.

- Supplied Ready-to-use Property Classes
- Creating Custom Properties

21.527.2 Supplied Ready-to-use Property Classes

Here is a list and short description of supplied fully-functional property classes. They are located in either props.h or advprops.h.

- wxArrayStringProperty
- wxBoolProperty
- wxColourProperty
- wxCursorProperty
- wxDateProperty
- wxDirProperty
- wxEditEnumProperty

- wxEnumProperty
- wxFileProperty
- wxFlagsProperty
- wxFloatProperty
- wxFontProperty
- wxImageFileProperty
- wxIntProperty
- wxLongStringProperty
- wxMultiChoiceProperty
- wxPropertyCategory
- wxStringProperty
- wxSystemColourProperty
- wxUIntProperty

wxPropertyCategory

Not an actual property per se, but a header for a group of properties. Regardless inherits from wxPGProperty, and supports displaying 'labels' for columns other than the first one. Easiest way to set category's label for second column is to call wxPGProperty::SetValue() with string argument.

wxStringProperty

Simple string property. wxPG_STRING_PASSWORD attribute may be used to echo value as asterisks and use wxTE_PASSWORD for wxTextCtrl. wxPG_ATTR_AUTOCOMPLETE attribute may be used to enable auto-completion (use a wxArrayString value), and is also supported by any property that happens to use a wxTextCtrl-based editor.

Remarks

wxStringProperty has a special trait: if it has value of "<composed>", and also has child properties, then its displayed value becomes composition of child property values, similar as with wxFontProperty, for instance.

wxIntProperty

Like wxStringProperty, but converts text to a signed long integer. wxIntProperty seamlessly supports 64-bit integers (ie. wxLongLong). To safely convert variant to integer, use code like this:

```
wxLongLong ll;
ll << property->GetValue();

// or
wxLongLong ll = propertyGrid->GetPropertyValueAsLong(property);
```

wxUIntProperty

Like wxIntProperty, but displays value as unsigned int. To set the prefix used globally, manipulate wxPG_UINT_PREFIX string attribute. To set the globally used base, manipulate wxPG_UINT_BASE int attribute. Regardless of current prefix, understands (hex) values starting with both "0x" and "$". Like wxIntProperty, wxUIntProperty seamlessly supports 64-bit unsigned integers (ie. wxULongLong). Same wxVariant safety rules apply.

wxFloatProperty

Like wxStringProperty, but converts text to a double-precision floating point. Default float-to-text precision is 6 decimals, but this can be changed by modifying wxPG_FLOAT_PRECISION attribute.

Note that when displaying the value, sign is omitted if the resulting textual representation is effectively zero (for example, -0.0001 with precision of 3 will become 0.0 instead of -0.0). This behaviour is unlike what C standard library does, but should result in better end-user experience in almost all cases.

wxBoolProperty

Represents a boolean value. wxChoice is used as editor control, by the default. wxPG_BOOL_USE_CHECKBOX attribute can be set to true in order to use check box instead.

wxLongStringProperty

Like wxStringProperty, but has a button that triggers a small text editor dialog. Note that in long string values, tabs are represented by "\t" and line break by "\n".

To display custom dialog on button press, you can subclass wxLongStringProperty and implement OnButtonClick, like this:

```
virtual bool OnButtonClick( wxPropertyGrid* propGrid, wxString& value )
{
    wxSize dialogSize(...size of your dialog...);

    wxPoint dlgPos = propGrid->GetGoodEditorDialogPosition(this,
                                                           dialogSize)

    // Create dialog dlg at dlgPos. Use value as initial string
    // value.
    ...

    if ( dlg.ShowModal() == wxID_OK )
    {
        value = dlg.GetStringValue);
        return true;
    }
    return false;
}
```

Also, if you wish not to have line breaks and tabs translated to escape sequences, then do following in constructor of your subclass:

```
m_flags |= wxPG_PROP_NO_ESCAPE;
```

wxDirProperty

Like wxLongStringProperty, but the button triggers dir selector instead. Supported properties (all with string value): wxPG_DIR_DIALOG_MESSAGE.

wxFileProperty

Like wxLongStringProperty, but the button triggers file selector instead. Default wildcard is "All files..." but this can be changed by setting wxPG_FILE_WILDCARD attribute (see wxFileDialog for format details). Attribute wxPG_F↩ ILE_SHOW_FULL_PATH can be set to false in order to show only the filename, not the entire path.

wxEnumProperty

Represents a single selection from a list of choices - wxOwnerDrawnComboBox is used to edit the value.

wxFlagsProperty

Represents a bit set that fits in a long integer. wxBoolProperty sub- properties are created for editing individual bits. Textctrl is created to manually edit the flags as a text; a continuous sequence of spaces, commas and semicolons are considered as a flag id separator.

Note: When changing "choices" (ie. flag labels) of wxFlagsProperty, you will need to use wxPGProperty::SetChoices() - otherwise they will not get updated properly.

wxFlagsProperty supports the same attributes as wxBoolProperty.

wxArrayStringProperty

Allows editing of a list of strings in wxTextCtrl and in a separate dialog. Supports "Delimiter" attribute, which defaults to comma (','').

wxDateProperty

wxDateTime property. Default editor is DatePickerCtrl, although TextCtrl should work as well. wxPG_DATE_FORMAT attribute can be used to change string wxDateTime::Format uses (although default is recommended as it is locale-dependent), and wxPG_DATE_PICKER_STYLE allows changing window style given to DatePickerCtrl (default is wxDP_DEFAULT|wxDP_SHOWCENTURY). Using wxDP_ALLOWNONE will enable better unspecified value support.

wxEditEnumProperty

Represents a string that can be freely edited or selected from list of choices - custom combobox control is used to edit the value.

wxMultiChoiceProperty

Allows editing a multiple selection from a list of strings. This is property is pretty much built around concept of wxMultiChoiceDialog. It uses wxArrayString value.

wxImageFileProperty

Like wxFileProperty, but has thumbnail of the image in front of the filename and autogenerates wildcard from available image handlers.

wxColourProperty

Useful alternate editor: Choice.

Represents wxColour. wxButton is used to trigger a colour picker dialog. There are various sub-classing opportunities with this class. See below in wxSystemColourProperty section for details.

Setting "HasAlpha" attribute to true for this property allows user to edit the alpha colour component.

wxFontProperty

Represents wxFont. Various sub-properties are used to edit individual subvalues.

wxSystemColourProperty

Represents wxColour and a system colour index. wxChoice is used to edit the value. Drop-down list has color images. Note that value type is wxColourPropertyValue instead of wxColour (which wxColourProperty uses).

```
class wxColourPropertyValue : public wxObject
{
public:
    //  An integer value relating to the colour, and which exact
    //  meaning depends on the property with which it is used.
    //
    //  For wxSystemColourProperty:
    //  Any of wxSYS_COLOUR_XXX, or any web-colour ( use wxPG_TO_WEB_COLOUR
    //  macro - (currently unsupported) ), or wxPG_COLOUR_CUSTOM.
    wxUint32    m_type;

    // Resulting colour. Should be correct regardless of type.
    wxColour    m_colour;
};
```

in wxSystemColourProperty, and its derived class wxColourProperty, there are various sub-classing features. To set a basic list of colour names, call wxPGProperty::SetChoices().

```
// Override in derived class to customize how colours are translated
// to strings.
virtual wxString ColourToString( const wxColour& col, int index ) const;

// Returns index of entry that triggers colour picker dialog
// (default is last).
virtual int GetCustomColourIndex() const;

// Helper function to show the colour dialog
bool QueryColourFromUser( wxVariant& variant ) const;

// Returns colour for given choice.
// Default function returns wxSystemSettings::GetColour(index).
virtual wxColour GetColour( int index ) const;
```

wxCursorProperty

Represents a wxCursor. wxChoice is used to edit the value. Drop-down list has cursor images under some (wxM↩ SW) platforms.

21.527.3 Creating Custom Properties

New properties can be created by subclassing wxPGProperty or one of the provided property classes, and (re)implementing necessary member functions. Below, each virtual member function has ample documentation about its purpose and any odd details which to keep in mind.

Here is a very simple 'template' code:

```
class MyProperty : public wxPGProperty
{
public:
    // Default constructor
    MyProperty() { }

    // All arguments of this ctor must have a default value -
    // use wxPG_LABEL for label and name
    MyProperty( const wxString& label = wxPG_LABEL,
                const wxString& name = wxPG_LABEL,
                const wxString& value = wxEmptyString )
        : wxPGProperty(label, name)
    {
        // m_value is wxVariant
        m_value = value;
    }

    virtual ~MyProperty() { }

    const wxPGEditor* DoGetEditorClass() const
    {
        // Determines editor used by property.
```

```
        // You can replace 'TextCtrl' below with any of these
        // builtin-in property editor identifiers: Choice, ComboBox,
        // TextCtrlAndButton, ChoiceAndButton, CheckBox, SpinCtrl,
        // DatePickerCtrl.
        return wxPGEditor_TextCtrl;
    }

    virtual wxString ValueToString( wxVariant& value,
                                    int argFlags ) const
    {
        // TODO: Convert given property value to a string
    }

    virtual bool StringToValue( wxVariant& variant, const
      wxString& text, int argFlags )
    {
        // TODO: Adapt string to property value.
    }

protected:
};
```

Since wxPGProperty derives from wxObject, you can use standard wxDECLARE_DYNAMIC_CLASS and wxI↩MPLEMENT_DYNAMIC_CLASS macros. From the above example they were omitted for sake of simplicity, and besides, they are only really needed if you need to use wxRTTI with your property class.

You can change the 'value type' of a property by simply assigning different type of variant with SetValue. **It is mandatory to implement wxVariantData class for all data types used as property values.** You can use macros declared in wxPropertyGrid headers. For instance:

```
// In header file:
// (If you need to have export declaration, use version of macros
// with _EXPORTED postfix)
WX_PG_DECLARE_VARIANT_DATA(MyDataClass)

// In sources file:
WX_PG_IMPLEMENT_VARIANT_DATA(MyDataClass)

// Or, if you don't have valid == operator:
WX_PG_IMPLEMENT_VARIANT_DATA_DUMMY_EQ(MyDataClass)
```

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Types

- typedef wxUint32 FlagType

Public Member Functions

- wxPGProperty ()

  Default constructor.
- wxPGProperty (const wxString &label, const wxString &name)

  Constructor.
- virtual ~wxPGProperty ()

  Virtual destructor.
- virtual void OnSetValue ()

  This virtual function is called after m_value has been set.
- virtual wxVariant DoGetValue () const

  Override this to return something else than m_value as the value.
- virtual bool ValidateValue (wxVariant &value, wxPGValidationInfo &validationInfo) const

  Implement this function in derived class to check the value.

- virtual bool StringToValue (wxVariant &variant, const wxString &text, int argFlags=0) const

  Converts text into wxVariant value appropriate for this property.
- virtual bool IntToValue (wxVariant &variant, int number, int argFlags=0) const

  Converts integer (possibly a choice selection) into wxVariant value appropriate for this property.
- virtual wxString ValueToString (wxVariant &value, int argFlags=0) const

  Converts property value into a text representation.
- bool SetValueFromString (const wxString &text, int flags=0)

  Converts string to a value, and if successful, calls SetValue() on it.
- bool SetValueFromInt (long value, int flags=0)

  Converts integer to a value, and if successful, calls SetValue() on it.
- virtual wxSize OnMeasureImage (int item=-1) const

  Returns size of the custom painted image in front of property.
- virtual bool OnEvent (wxPropertyGrid *propgrid, wxWindow *wnd_primary, wxEvent &event)

  Events received by editor widgets are processed here.
- virtual wxVariant ChildChanged (wxVariant &thisValue, int childIndex, wxVariant &childValue) const

  Called after value of a child property has been altered.
- virtual const wxPGEditor * DoGetEditorClass () const

  Returns pointer to an instance of used editor.
- virtual wxValidator * DoGetValidator () const

  Returns pointer to the wxValidator that should be used with the editor of this property (NULL for no validator).
- virtual void OnCustomPaint (wxDC &dc, const wxRect &rect, wxPGPaintData &paintdata)

  Override to paint an image in front of the property value text or drop-down list item (but only if wxPGProperty::OnMeasureImage is overridden as well).
- virtual wxPGCellRenderer * GetCellRenderer (int column) const

  Returns used wxPGCellRenderer instance for given property column (label=0, value=1).
- virtual int GetChoiceSelection () const

  Returns which choice is currently selected.
- virtual void RefreshChildren ()

  Refresh values of child properties.
- virtual bool DoSetAttribute (const wxString &name, wxVariant &value)

  Reimplement this member function to add special handling for attributes of this property.
- virtual wxVariant DoGetAttribute (const wxString &name) const

  Returns value of an attribute.
- virtual wxPGEditorDialogAdapter * GetEditorDialog () const

  Returns instance of a new wxPGEditorDialogAdapter instance, which is used when user presses the (optional) button next to the editor control;.
- virtual void OnValidationFailure (wxVariant &pendingValue)

  Called whenever validation has failed with given pending value.
- int AddChoice (const wxString &label, int value=wxPG_INVALID_VALUE)

  Append a new choice to property's list of choices.
- wxDEPRECATED (void AddChild(wxPGProperty *prop))

  Adds a private child property.
- void AddPrivateChild (wxPGProperty *prop)

  Adds a private child property.
- void AdaptListToValue (wxVariant &list, wxVariant *value) const

  Adapts list variant into proper value using consecutive ChildChanged() calls.
- wxPGProperty * AppendChild (wxPGProperty *childProperty)

  Use this member function to add independent (ie.
- bool AreAllChildrenSpecified (wxVariant *pendingList=NULL) const

  Determines, recursively, if all children are not unspecified.
- bool AreChildrenComponents () const

Returns true if children of this property are component values (for instance, points size, face name, and is_underlined are component values of a font).

- void ChangeFlag (wxPGPropertyFlags flag, bool set)

  Sets or clears given property flag.
- void DeleteChildren ()

  Deletes children of the property.
- void DeleteChoice (int index)

  Removes entry from property's wxPGChoices and editor control (if it is active).
- void Enable (bool enable=true)

  Enables or disables the property.
- wxString GenerateComposedValue () const

  Composes text from values of child properties.
- wxVariant GetAttribute (const wxString &name) const

  Returns property attribute value, null variant if not found.
- wxString GetAttribute (const wxString &name, const wxString &defVal) const

  Returns named attribute, as string, if found.
- long GetAttributeAsLong (const wxString &name, long defVal) const

  Returns named attribute, as long, if found.
- double GetAttributeAsDouble (const wxString &name, double defVal) const

  Returns named attribute, as double, if found.
- wxVariant GetAttributesAsList () const

  Returns attributes as list wxVariant.
- const wxPGEditor * GetColumnEditor (int column) const

  Returns editor used for given column.
- const wxString & GetBaseName () const

  Returns property's base name (ie.
- const wxPGCell & GetCell (unsigned int column) const

  Returns wxPGCell of given column.
- wxPGCell & GetCell (unsigned int column)

  Returns wxPGCell of given column, creating one if necessary.
- wxPGCell & GetOrCreateCell (unsigned int column)

  Returns wxPGCell of given column, creating one if necessary.
- unsigned int GetChildCount () const

  Returns number of child properties.
- int GetChildrenHeight (int lh, int iMax=-1) const

  Returns height of children, recursively, and by taking expanded/collapsed status into account.
- const wxPGChoices & GetChoices () const

  Returns read-only reference to property's list of choices.
- void * GetClientData () const

  Returns client data (void) of a property.*
- wxClientData * GetClientObject () const

  Sets managed client object of a property.
- wxVariant GetDefaultValue () const

  Returns property's default value.
- wxString GetDisplayedString () const

  Returns property's displayed text.
- const wxPGEditor * GetEditorClass () const

  Returns wxPGEditor that will be used and created when property becomes selected.
- FlagType GetFlags () const

  Returns property flags.
- wxPropertyGrid * GetGrid () const

Returns property grid where property lies.

- wxPropertyGrid * GetGridIfDisplayed () const

  Returns owner wxPropertyGrid, but only if one is currently on a page displaying this property.
- const wxString & GetHelpString () const

  Returns property's help or description text.
- unsigned int GetIndexInParent () const

  Returns position in parent's array.
- const wxString & GetLabel () const

  Returns property's label.
- const wxPGProperty * GetLastVisibleSubItem () const

  Returns last visible child property, recursively.
- wxPGProperty * GetMainParent () const

  Returns highest level non-category, non-root parent.
- int GetMaxLength () const

  Returns maximum allowed length of property's text value.
- wxString GetName () const

  Returns property's name with all (non-category, non-root) parents.
- wxPGProperty * GetParent () const

  Return parent of property.
- wxPGProperty * GetPropertyByName (const wxString &name) const

  Returns (direct) child property with given name (or NULL if not found).
- wxValidator * GetValidator () const

  Gets assignable version of property's validator.
- wxVariant GetValue () const

  Returns property's value.
- wxBitmap * GetValueImage () const

  Returns bitmap that appears next to value text.
- virtual wxString GetValueAsString (int argFlags=0) const

  Returns text representation of property's value.
- wxDEPRECATED (wxString GetValueString(int argFlags=0) const)

  Synonymous to GetValueAsString().
- wxString GetValueType () const

  Returns value type used by this property.
- int GetY () const

  Returns coordinate to the top y of the property.
- FlagType HasFlag (wxPGPropertyFlags flag) const

  Returns non-zero if property has given flag set.
- bool HasVisibleChildren () const

  Returns true if property has even one visible child.
- bool Hide (bool hide, int flags=wxPG_RECURSE)

  Hides or reveals the property.
- int Index (const wxPGProperty *p) const

  Returns index of given child property.
- wxPGProperty * InsertChild (int index, wxPGProperty *childProperty)

  Use this member function to add independent (ie.
- int InsertChoice (const wxString &label, int index, int value=wxPG_INVALID_VALUE)

  Inserts a new choice to property's list of choices.
- bool IsCategory () const

  Returns true if this property is actually a wxPropertyCategory.
- bool IsEnabled () const

  Returns true if property is enabled.

- bool IsExpanded () const

  Returns true if property has visible children.
- bool IsRoot () const

  Returns true if this property is actually a wxRootProperty.
- bool IsSomeParent (wxPGProperty *candidateParent) const

  Returns true if candidateParent is some parent of this property.
- bool IsTextEditable () const

  Returns true if property has editable wxTextCtrl when selected.
- bool IsValueUnspecified () const

  Returns true if property's value is considered unspecified.
- bool IsVisible () const

  Returns true if all parents expanded.
- wxPGProperty * Item (unsigned int i) const

  Returns child property at index i.
- void RefreshEditor ()

  If property's editor is active, then update it's value.
- void SetAttribute (const wxString &name, wxVariant value)

  Sets an attribute for this property.
- void SetAutoUnspecified (bool enable=true)

  Set if user can change the property's value to unspecified by modifying the value of the editor control (usually by clearing it).
- void SetBackgroundColour (const wxColour &colour, int flags=wxPG_RECURSE)

  Sets property's background colour.
- void SetEditor (const wxPGEditor *editor)

  Sets editor for a property.
- void SetEditor (const wxString &editorName)

  Sets editor for a property, by editor name.
- void SetCell (int column, const wxPGCell &cell)

  Sets cell information for given column.
- bool SetChoices (wxPGChoices &choices)

  Sets new set of choices for the property.
- void SetClientData (void *clientData)

  Sets client data (void) of a property.*
- void SetClientObject (wxClientData *clientObject)

  Returns client object of a property.
- void SetChoiceSelection (int newValue)

  Sets selected choice and changes property value.
- void SetDefaultValue (wxVariant &value)

  Set default value of a property.
- void SetFlagRecursively (wxPGPropertyFlags flag, bool set)

  Sets or clears given property flag, recursively.
- void SetHelpString (const wxString &helpString)

  Sets property's help string, which is shown, for example, in wxPropertyGridManager's description text box.
- void SetLabel (const wxString &label)

  Sets property's label.
- bool SetMaxLength (int maxLen)

  Set max length of text in text editor.
- void SetModifiedStatus (bool modified)

  Sets property's "is it modified?" flag.
- void SetName (const wxString &newName)

  Sets new (base) name for property.

- void SetParentalType (int flag)

  Changes what sort of parent this property is for its children.
- void SetTextColour (const wxColour &colour, int flags=wxPG_RECURSE)

  Sets property's text colour.
- void SetValidator (const wxValidator &validator)

  Sets wxValidator for a property.
- void SetValue (wxVariant value, wxVariant *pList=NULL, int flags=wxPG_SETVAL_REFRESH_EDITOR)

  Call this to set value of the property.
- void SetValueImage (wxBitmap &bmp)

  Set wxBitmap in front of the value.
- void SetValueInEvent (wxVariant value) const

  Call this function in OnEvent(), OnButtonClick() etc.
- void SetValueToUnspecified ()

  Sets property's value to unspecified (ie.
- void SetWasModified (bool set=true)

  Call with false in OnSetValue() to cancel value changes after all (ie.
- wxPGProperty * UpdateParentValues ()

  Updates composed values of parent non-category properties, recursively.
- bool UsesAutoUnspecified () const

  Returns true if containing grid uses wxPG_EX_AUTO_UNSPECIFIED_VALUES.

Protected Member Functions

- void Empty ()

  Deletes all child properties.

Additional Inherited Members

21.527.4 Member Typedef Documentation

typedef wxUint32 wxPGProperty::FlagType

21.527.5 Constructor & Destructor Documentation

wxPGProperty::wxPGProperty ( )

Default constructor.

wxPGProperty::wxPGProperty (const wxString & *label*, const wxString & *name*)

Constructor.

Non-abstract property classes should have constructor of this style:

```
MyProperty( const wxString& label, const wxString& name, const T& value )
    : wxPGProperty(label, name)
{
    // Generally recommended way to set the initial value
    // (as it should work in pretty much 100% of cases).
    wxVariant variant;
    variant << value;
    SetValue(variant);

    // If has private child properties then create them here.
    // For example:
    //     AddPrivateChild( new wxStringProperty("Subprop 1",
    //                                           wxPG_LABEL,
    //                                           value.GetSubProp1()));
}
```

virtual wxPGProperty::~wxPGProperty ( ) [virtual]

Virtual destructor.

It is customary for derived properties to implement this.

21.527.6 Member Function Documentation

void wxPGProperty::AdaptListToValue (wxVariant & *list*, wxVariant * *value*) const

Adapts list variant into proper value using consecutive ChildChanged() calls.

int wxPGProperty::AddChoice (const wxString & *label*, int *value* = wxPG_INVALID_VALUE)

Append a new choice to property's list of choices.

Parameters

label	Label for added choice.
value	Value for new choice. Do not specify if you wish this to equal choice index.

Returns

Index to added choice.

void wxPGProperty::AddPrivateChild (wxPGProperty * *prop*)

Adds a private child property.

If you use this instead of wxPropertyGridInterface::Insert() or wxPropertyGridInterface::AppendIn(), then property's parental type will automatically be set up to wxPG_PROP_AGGREGATE. In other words, all properties of this property will become private.

wxPGProperty* wxPGProperty::AppendChild (wxPGProperty * *childProperty*)

Use this member function to add independent (ie.

regular) children to a property.

Returns

Appended childProperty.

Remarks

wxPropertyGrid is not automatically refreshed by this function.

See also

InsertChild(), AddPrivateChild()

bool wxPGProperty::AreAllChildrenSpecified (wxVariant * *pendingList* = NULL) const

Determines, recursively, if all children are not unspecified.

Parameters

pendingList	Assumes members in this wxVariant list as pending replacement values.

bool wxPGProperty::AreChildrenComponents ( ) const

Returns true if children of this property are component values (for instance, points size, face name, and is_↩ underlined are component values of a font).

void wxPGProperty::ChangeFlag (wxPGPropertyFlags *flag,* bool *set*)

Sets or clears given property flag.

Mainly for internal use.

Remarks

Setting a property flag never has any side-effect, and is intended almost exclusively for internal use. So, for example, if you want to disable a property, call Enable(false) instead of setting wxPG_PROP_DISABLED flag.

See also

HasFlag(), GetFlags()

virtual wxVariant wxPGProperty::ChildChanged (wxVariant & *thisValue,* int *childIndex,* wxVariant & *childValue*) const [virtual]

Called after value of a child property has been altered.

Must return new value of the whole property (after any alterations warranted by child's new value).

Note that this function is usually called at the time that value of this property, or given child property, is still pending for change, and as such, result of GetValue() or m_value should not be relied on.

Sample pseudo-code implementation:

```
wxVariant MyProperty::ChildChanged( wxVariant& thisValue,
                                    int childIndex,
                                    wxVariant& childValue ) const
{
    // Acquire reference to actual type of data stored in variant
    // (TFromVariant only exists if wxPropertyGrid's wxVariant-macros
    // were used to create the variant class).
    T& data = TFromVariant(thisValue);

    // Copy childValue into data.
    switch ( childIndex )
    {
        case 0:
            data.SetSubProp1( childvalue.GetLong() );
            break;
        case 1:
            data.SetSubProp2( childvalue.GetString() );
            break;
        ...
    }

    // Return altered data
    return data;
}
```

Parameters

thisValue	Value of this property. Changed value should be returned (in previous versions of wxPropertyGrid it was only necessary to write value back to this argument).
childIndex	Index of child changed (you can use Item(childIndex) to get child property).
childValue	(Pending) value of the child property.

Returns

Modified value of the whole property.

void wxPGProperty::DeleteChildren ( )

Deletes children of the property.

void wxPGProperty::DeleteChoice (int *index*)

Removes entry from property's wxPGChoices and editor control (if it is active).

If selected item is deleted, then the value is set to unspecified.

virtual wxVariant wxPGProperty::DoGetAttribute (const wxString & *name*) const [virtual]

Returns value of an attribute.

Override if custom handling of attributes is needed.

Default implementation simply return NULL variant.

virtual const wxPGEditor* wxPGProperty::DoGetEditorClass ( ) const [virtual]

Returns pointer to an instance of used editor.

virtual wxValidator* wxPGProperty::DoGetValidator ( ) const [virtual]

Returns pointer to the wxValidator that should be used with the editor of this property (NULL for no validator).

Setting validator explicitly via SetPropertyValidator will override this.

In most situations, code like this should work well (macros are used to maintain one actual validator instance, so on the second call the function exits within the first macro):

```
wxValidator* wxMyPropertyClass::DoGetValidator () const
{
    WX_PG_DOGETVALIDATOR_ENTRY()

    wxMyValidator* validator = new wxMyValidator(...);

    ... prepare validator...

    WX_PG_DOGETVALIDATOR_EXIT(validator)
}
```

Remarks

You can get common filename validator by returning wxFileProperty::GetClassValidator(). wxDirProperty, for example, uses it.

virtual wxVariant wxPGProperty::DoGetValue ( ) const `[virtual]`

Override this to return something else than m_value as the value.

virtual bool wxPGProperty::DoSetAttribute (const wxString & *name*, wxVariant & *value*) `[virtual]`

Reimplement this member function to add special handling for attributes of this property.

Returns

Return false to have the attribute automatically stored in m_attributes. Default implementation simply does that and nothing else.

Remarks

To actually set property attribute values from the application, use wxPGProperty::SetAttribute() instead.

void wxPGProperty::Empty ( ) `[protected]`

Deletes all child properties.

void wxPGProperty::Enable (bool *enable* = `true`)

Enables or disables the property.

Disabled property usually appears as having grey text.

Parameters

enable	If false, property is disabled instead.

See also

wxPropertyGridInterface::EnableProperty()

wxString wxPGProperty::GenerateComposedValue ( ) const

Composes text from values of child properties.

wxVariant wxPGProperty::GetAttribute (const wxString & *name*) const

Returns property attribute value, null variant if not found.

wxString wxPGProperty::GetAttribute (const wxString & *name*, const wxString & *defVal*) const

Returns named attribute, as string, if found.

Otherwise defVal is returned.

double wxPGProperty::GetAttributeAsDouble (const wxString & *name*, double *defVal*) const

Returns named attribute, as double, if found.

Otherwise defVal is returned.

long wxPGProperty::GetAttributeAsLong (const wxString & *name,* long *defVal*) const

Returns named attribute, as long, if found.

Otherwise defVal is returned.

wxVariant wxPGProperty::GetAttributesAsList ( ) const

Returns attributes as list wxVariant.

const wxString& wxPGProperty::GetBaseName ( ) const

Returns property's base name (ie.

parent's name is not added in any case)

const wxPGCell& wxPGProperty::GetCell (unsigned int *column*) const

Returns wxPGCell of given column.

Remarks

const version of this member function returns 'default' wxPGCell object if the property itself didn't hold cell data.

wxPGCell& wxPGProperty::GetCell (unsigned int *column*)

Returns wxPGCell of given column, creating one if necessary.

virtual wxPGCellRenderer* wxPGProperty::GetCellRenderer (int *column*) const `[virtual]`

Returns used wxPGCellRenderer instance for given property column (label=0, value=1).

Default implementation returns editor's renderer for all columns.

unsigned int wxPGProperty::GetChildCount ( ) const

Returns number of child properties.

int wxPGProperty::GetChildrenHeight (int *lh,* int *iMax* = `-1`) const

Returns height of children, recursively, and by taking expanded/collapsed status into account.

Parameters

lh	Line height. Pass result of GetGrid()->GetRowHeight() here.
iMax	Only used (internally) when finding property y-positions.

const wxPGChoices& wxPGProperty::GetChoices ( ) const

Returns read-only reference to property's list of choices.

virtual int wxPGProperty::GetChoiceSelection ( ) const `[virtual]`

Returns which choice is currently selected.

Only applies to properties which have choices.

Needs to reimplemented in derived class if property value does not map directly to a choice. Integer as index, bool, and string usually do.

void* wxPGProperty::GetClientData ( ) const

Returns client data (void*) of a property.

wxClientData* wxPGProperty::GetClientObject ( ) const

Sets managed client object of a property.

const wxPGEditor* wxPGProperty::GetColumnEditor (int *column*) const

Returns editor used for given column.

NULL for no editor.

wxVariant wxPGProperty::GetDefaultValue ( ) const

Returns property's default value.

If property's value type is not a built-in one, and "DefaultValue" attribute is not defined, then this function usually returns Null variant.

wxString wxPGProperty::GetDisplayedString ( ) const

Returns property's displayed text.

const wxPGEditor* wxPGProperty::GetEditorClass ( ) const

Returns wxPGEditor that will be used and created when property becomes selected.

Returns more accurate value than DoGetEditorClass().

virtual wxPGEditorDialogAdapter* wxPGProperty::GetEditorDialog ( ) const `[virtual]`

Returns instance of a new wxPGEditorDialogAdapter instance, which is used when user presses the (optional) button next to the editor control;.

Default implementation returns NULL (ie. no action is generated when button is pressed).

FlagType wxPGProperty::GetFlags ( ) const

Returns property flags.

wxPropertyGrid* wxPGProperty::GetGrid ( ) const

Returns property grid where property lies.

wxPropertyGrid* wxPGProperty::GetGridIfDisplayed ( ) const

Returns owner wxPropertyGrid, but only if one is currently on a page displaying this property.

const wxString& wxPGProperty::GetHelpString ( ) const

Returns property's help or description text.

See also

SetHelpString()

unsigned int wxPGProperty::GetIndexInParent ( ) const

Returns position in parent's array.

const wxString& wxPGProperty::GetLabel ( ) const

Returns property's label.

const wxPGProperty* wxPGProperty::GetLastVisibleSubItem ( ) const

Returns last visible child property, recursively.

wxPGProperty* wxPGProperty::GetMainParent ( ) const

Returns highest level non-category, non-root parent.

Useful when you have nested properties with children.

Remarks

If immediate parent is root or category, this will return the property itself.

int wxPGProperty::GetMaxLength ( ) const

Returns maximum allowed length of property's text value.

wxString wxPGProperty::GetName ( ) const

Returns property's name with all (non-category, non-root) parents.

wxPGCell& wxPGProperty::GetOrCreateCell (unsigned int *column*)

Returns wxPGCell of given column, creating one if necessary.

wxPGProperty* wxPGProperty::GetParent ( ) const

Return parent of property.

wxPGProperty* wxPGProperty::GetPropertyByName (const wxString & *name*) const

Returns (direct) child property with given name (or NULL if not found).

wxValidator* wxPGProperty::GetValidator ( ) const

Gets assignable version of property's validator.

wxVariant wxPGProperty::GetValue ( ) const

Returns property's value.

virtual wxString wxPGProperty::GetValueAsString (int *argFlags* = 0) const [virtual]

Returns text representation of property's value.

Parameters

argFlags	If 0 (default value), then displayed string is returned. If wxPG_FULL_VALUE is set, returns complete, storable string value instead of displayable. If wxPG_EDITABLE_VALUE is set, returns string value that must be editable in textctrl. If wxPG_COMPOSITE_FRAGMENT is set, returns text that is appropriate to display as a part of string property's composite text representation.

Remarks

In older versions, this function used to be overridden to convert property's value into a string representation. This function is now handled by ValueToString(), and overriding this function now will result in run-time assertion failure.

wxBitmap* wxPGProperty::GetValueImage ( ) const

Returns bitmap that appears next to value text.

Only returns non-NULL bitmap if one was set with SetValueImage().

wxString wxPGProperty::GetValueType ( ) const

Returns value type used by this property.

int wxPGProperty::GetY ( ) const

Returns coordinate to the top y of the property.

Note that the position of scrollbars is not taken into account.

FlagType wxPGProperty::HasFlag (wxPGPropertyFlags *flag*) const

Returns non-zero if property has given flag set.

See also

propgrid_propflags

bool wxPGProperty::HasVisibleChildren ( ) const

Returns true if property has even one visible child.

bool wxPGProperty::Hide (bool *hide*, int *flags* = wxPG_RECURSE)

Hides or reveals the property.

Parameters

hide	true for hide, false for reveal.
flags	By default changes are applied recursively. Set this parameter wxPG_DONT_RECURSE to prevent this.

int wxPGProperty::Index (const wxPGProperty * *p*) const

Returns index of given child property.

wxNOT_FOUND if given property is not child of this.

wxPGProperty* wxPGProperty::InsertChild (int *index*, wxPGProperty * *childProperty*)

Use this member function to add independent (ie.

regular) children to a property.

Returns

Inserted childProperty.

Remarks

wxPropertyGrid is not automatically refreshed by this function.

See also

AppendChild(), AddPrivateChild()

int wxPGProperty::InsertChoice (const wxString & *label*, int *index*, int *value* = wxPG_INVALID_VALUE)

Inserts a new choice to property's list of choices.

Parameters

label	Text for new choice
index	Insertion position. Use wxNOT_FOUND to append.
value	Value for new choice. Do not specify if you wish this to equal choice index.

virtual bool wxPGProperty::IntToValue (wxVariant & *variant*, int *number*, int *argFlags* = 0) const [virtual]

Converts integer (possibly a choice selection) into wxVariant value appropriate for this property.

Parameters

variant	On function entry this is the old value (should not be wxNullVariant in normal cases). Translated value must be assigned back to it.
number	Integer to be translated into variant.
argFlags	If wxPG_FULL_VALUE is set, returns complete, storable value instead of displayable one.

Returns

Returns true if resulting wxVariant value was different.

Remarks

- If property is not supposed to use choice or spinctrl or other editor with int-based value, it is not necessary to implement this method.
- Default implementation simply assign given int to m_value.
- If property uses choice control, and displays a dialog on some choice items, then it is preferred to display that dialog in IntToValue instead of OnEvent.
- You might want to take into account that m_value is Mull variant if property value is unspecified (which is usually only case if you explicitly enabled that sort behaviour).

bool wxPGProperty::IsCategory ( ) const

Returns true if this property is actually a wxPropertyCategory.

bool wxPGProperty::IsEnabled ( ) const

Returns true if property is enabled.

bool wxPGProperty::IsExpanded ( ) const

Returns true if property has visible children.

bool wxPGProperty::IsRoot ( ) const

Returns true if this property is actually a wxRootProperty.

bool wxPGProperty::IsSomeParent (wxPGProperty * *candidateParent*) const

Returns true if candidateParent is some parent of this property.

bool wxPGProperty::IsTextEditable ( ) const

Returns true if property has editable wxTextCtrl when selected.

Remarks

Although disabled properties do not displayed editor, they still return true here as being disabled is considered a temporary condition (unlike being read-only or having limited editing enabled).

bool wxPGProperty::IsValueUnspecified ( ) const

Returns true if property's value is considered unspecified.

This usually means that value is Null variant.

bool wxPGProperty::IsVisible ( ) const

Returns true if all parents expanded.

wxPGProperty* wxPGProperty::Item (unsigned int *i*) const

Returns child property at index i.

virtual void wxPGProperty::OnCustomPaint (wxDC & *dc*, const wxRect & *rect*, wxPGPaintData & *paintdata*) [virtual]

Override to paint an image in front of the property value text or drop-down list item (but only if wxPGProperty::OnMeasureImage is overridden as well).

If property's OnMeasureImage() returns size that has height != 0 but less than row height (< 0 has special meanings), wxPropertyGrid calls this method to draw a custom image in a limited area in front of the editor control or value text/graphics, and if control has drop-down list, then the image is drawn there as well (even in the case OnMeasureImage() returned higher height than row height).

NOTE: Following applies when OnMeasureImage() returns a "flexible" height (using wxPG_FLEXIBLE_SIZE(W,H) macro), which implies variable height items: If rect.x is < 0, then this is a measure item call, which means that dc is invalid and only thing that should be done is to set paintdata.m_drawnHeight to the height of the image of item at index paintdata.m_choiceItem. This call may be done even as often as once every drop-down popup show.

Parameters

dc	wxDC to paint on.
rect	Box reserved for custom graphics. Includes surrounding rectangle, if any. If x is < 0, then this is a measure item call (see above).

paintdata	wxPGPaintData structure with much useful data about painted item.
	
	`struct wxPGPaintData`
	`{`
	`  // wxPropertyGrid.`
	`  const wxPropertyGrid*  m_parent;`
	
	`  // Normally -1, otherwise index to drop-down list item that has to be drawn.`
	`  int  m_choiceItem;`
	
	`  // Set to drawn width in OnCustomPaint (optional).`
	`  int  m_drawnWidth;`
	
	`  // In a measure item call, set this to the height of item at m_choiceItem index`
	`  int  m_drawnHeight;`
	`};`

Remarks

- You can actually exceed rect width, but if you do so then paintdata.m_drawnWidth must be set to the full width drawn in pixels.
- Due to technical reasons, rect's height will be default even if custom height was reported during measure call.
- Brush is guaranteed to be default background colour. It has been already used to clear the background of area being painted. It can be modified.
- Pen is guaranteed to be 1-wide 'black' (or whatever is the proper colour) pen for drawing framing rectangle. It can be changed as well.

See also

ValueToString()

virtual bool wxPGProperty::OnEvent (wxPropertyGrid * *propgrid*, wxWindow * *wnd_primary*, wxEvent & *event*) [virtual]

Events received by editor widgets are processed here.

Note that editor class usually processes most events. Some, such as button press events of TextCtrlAndButton class, can be handled here. Also, if custom handling for regular events is desired, then that can also be done (for example, wxSystemColourProperty custom handles `wxEVT_CHOICE` to display colour picker dialog when 'custom' selection is made).

If the event causes value to be changed, SetValueInEvent() should be called to set the new value.

The parameter *event* is the associated wxEvent.

Return values

Should	return true if any changes in value should be reported.

Remarks

- If property uses choice control, and displays a dialog on some choice items, then it is preferred to display that dialog in IntToValue instead of OnEvent.

virtual wxSize wxPGProperty::OnMeasureImage (int *item* = -1) const [virtual]

Returns size of the custom painted image in front of property.

This method must be overridden to return non-default value if OnCustomPaint is to be called.

Parameters

item	Normally -1, but can be an index to the property's list of items.

Remarks

- Default behaviour is to return wxSize(0,0), which means no image.
- Default image width or height is indicated with dimension -1.
- You can also return wxPG_DEFAULT_IMAGE_SIZE which equals wxSize(-1, -1).

virtual void wxPGProperty::OnSetValue ( ) [virtual]

This virtual function is called after m_value has been set.

Remarks

- If m_value was set to Null variant (ie. unspecified value), OnSetValue() will not be called.
- m_value may be of any variant type. Typically properties internally support only one variant type, and as such OnSetValue() provides a good opportunity to convert supported values into internal type.
- Default implementation does nothing.

virtual void wxPGProperty::OnValidationFailure (wxVariant & *pendingValue*) `[virtual]`

Called whenever validation has failed with given pending value.

Remarks

If you implement this in your custom property class, please remember to call the baser implementation as well, since they may use it to revert property into pre-change state.

virtual void wxPGProperty::RefreshChildren ( ) `[virtual]`

Refresh values of child properties.

Automatically called after value is set.

void wxPGProperty::RefreshEditor ( )

If property's editor is active, then update it's value.

void wxPGProperty::SetAttribute (const wxString & *name,* wxVariant *value*)

Sets an attribute for this property.

Parameters

name	Text identifier of attribute. See wxPropertyGrid Property Attribute Identifiers.
value	Value of attribute.

Remarks

Setting attribute's value to Null variant will simply remove it from property's set of attributes.

void wxPGProperty::SetAutoUnspecified (bool *enable* = `true`)

Set if user can change the property's value to unspecified by modifying the value of the editor control (usually by clearing it).

Currently, this can work with following properties: wxIntProperty, wxUIntProperty, wxFloatProperty, wxEditEnum↩ Property.

Parameters

enable	Whether to enable or disable this behaviour (it is disabled by default).

void wxPGProperty::SetBackgroundColour (const wxColour & *colour,* int *flags* = `wxPG_RECURSE`)

Sets property's background colour.

Parameters

colour	Background colour to use.
flags	Default is wxPG_RECURSE which causes colour to be set recursively. Omit this flag to only set colour for the property in question and not any of its children.

void wxPGProperty::SetCell (int *column,* const wxPGCell & *cell*)

Sets cell information for given column.

bool wxPGProperty::SetChoices (wxPGChoices & *choices*)

Sets new set of choices for the property.

Remarks

This operation deselects the property and clears its value.

void wxPGProperty::SetChoiceSelection (int *newValue*)

Sets selected choice and changes property value.

Tries to retain value type, although currently if it is not string, then it is forced to integer.

void wxPGProperty::SetClientData (void * *clientData*)

Sets client data (void*) of a property.

Remarks

This untyped client data has to be deleted manually.

void wxPGProperty::SetClientObject (wxClientData * *clientObject*)

Returns client object of a property.

void wxPGProperty::SetDefaultValue (wxVariant & *value*)

Set default value of a property.

Synonymous to

```
SetAttribute("DefaultValue", value);
```

void wxPGProperty::SetEditor (const wxPGEditor * *editor*)

Sets editor for a property.

Parameters

editor	For builtin editors, use wxPGEditor_X, where X is builtin editor's name (TextCtrl, Choice, etc. see wxPGEditor documentation for full list).

For custom editors, use pointer you received from wxPropertyGrid::RegisterEditorClass().

void wxPGProperty::SetEditor (const wxString & *editorName*)

Sets editor for a property, by editor name.

void wxPGProperty::SetFlagRecursively (wxPGPropertyFlags *flag,* bool *set*)

Sets or clears given property flag, recursively.

This function is primarily intended for internal use.

See also

ChangeFlag()

void wxPGProperty::SetHelpString (const wxString & *helpString*)

Sets property's help string, which is shown, for example, in wxPropertyGridManager's description text box.

void wxPGProperty::SetLabel (const wxString & *label*)

Sets property's label.

Remarks

Properties under same parent may have same labels. However, property names must still remain unique.

bool wxPGProperty::SetMaxLength (int *maxLen*)

Set max length of text in text editor.

void wxPGProperty::SetModifiedStatus (bool *modified*)

Sets property's "is it modified?" flag.

Affects children recursively.

void wxPGProperty::SetName (const wxString & *newName*)

Sets new (base) name for property.

void wxPGProperty::SetParentalType (int *flag*)

Changes what sort of parent this property is for its children.

Parameters

flag	Use one of the following values: wxPG_PROP_MISC_PARENT (for generic parents), wxPG_PROP_CATEGORY (for categories), or wxPG_PROP_AGGREGATE (for derived property classes with private children).

Remarks

You generally do not need to call this function.

void wxPGProperty::SetTextColour (const wxColour & *colour,* int *flags* = `wxPG_RECURSE`)

Sets property's text colour.

Parameters

colour	Text colour to use.
flags	Default is wxPG_RECURSE which causes colour to be set recursively. Omit this flag to only set colour for the property in question and not any of its children.

void wxPGProperty::SetValidator (const wxValidator & *validator*)

Sets wxValidator for a property.

void wxPGProperty::SetValue (wxVariant *value*, wxVariant * *pList* = NULL, int *flags* = wxPG_SETVAL_REFRESH_EDITOR)

Call this to set value of the property.

Unlike methods in wxPropertyGrid, this does not automatically update the display.

Remarks

Use wxPropertyGrid::ChangePropertyValue() instead if you need to run through validation process and send property change event.

If you need to change property value in event, based on user input, use SetValueInEvent() instead.

Parameters

value	The value to set.
pList	Pointer to list variant that contains child values. Used to indicate which children should be marked as modified. Usually you just use NULL.
flags	wxPG_SETVAL_REFRESH_EDITOR is set by default, to refresh editor and redraw properties.

bool wxPGProperty::SetValueFromInt (long *value*, int *flags* = 0)

Converts integer to a value, and if successful, calls SetValue() on it.

Default behaviour is to do nothing.

Parameters

value	Int to get the value from.
flags	If has wxPG_FULL_VALUE, then the value given is a actual value and not an index.

Returns

true if value was changed.

bool wxPGProperty::SetValueFromString (const wxString & *text*, int *flags* = 0)

Converts string to a value, and if successful, calls SetValue() on it.

Default behaviour is to do nothing.

Parameters

text	String to get the value from.
flags	

Todo docme

Returns

true if value was changed.

void wxPGProperty::SetValueImage (wxBitmap & *bmp*)

Set wxBitmap in front of the value.

This bitmap may be ignored by custom cell renderers.

void wxPGProperty::SetValueInEvent (wxVariant *value*) const

Call this function in OnEvent(), OnButtonClick() etc.

to change the property value based on user input.

Remarks

This method is const since it doesn't actually modify value, but posts given variant as pending value, stored in wxPropertyGrid.

void wxPGProperty::SetValueToUnspecified ( )

Sets property's value to unspecified (ie.

Null variant).

void wxPGProperty::SetWasModified (bool *set* = `true`)

Call with false in OnSetValue() to cancel value changes after all (ie.

cancel true returned by StringToValue() or IntToValue()).

virtual bool wxPGProperty::StringToValue (wxVariant & *variant*, const wxString & *text*, int *argFlags* = `0`) const `[virtual]`

Converts text into wxVariant value appropriate for this property.

Parameters

variant	On function entry this is the old value (should not be wxNullVariant in normal cases). Translated value must be assigned back to it.
text	Text to be translated into variant.
argFlags	If wxPG_FULL_VALUE is set, returns complete, storable value instead of displayable one (they may be different). If wxPG_COMPOSITE_FRAGMENT is set, text is interpreted as a part of composite property string value (as generated by ValueToString() called with this same flag).

Returns

Returns true if resulting wxVariant value was different.

Remarks

Default implementation converts semicolon delimited tokens into child values. Only works for properties with children.

You might want to take into account that m_value is Null variant if property value is unspecified (which is usually only case if you explicitly enabled that sort behaviour).

wxPGProperty* wxPGProperty::UpdateParentValues ( )

Updates composed values of parent non-category properties, recursively.

Returns topmost property updated.

bool wxPGProperty::UsesAutoUnspecified ( ) const

Returns true if containing grid uses wxPG_EX_AUTO_UNSPECIFIED_VALUES.

virtual bool wxPGProperty::ValidateValue (wxVariant & *value*, wxPGValidationInfo & *validationInfo*) const [virtual]

Implement this function in derived class to check the value.

Return true if it is ok. Returning false prevents property change events from occurring.

Remarks

- Default implementation always returns true.

virtual wxString wxPGProperty::ValueToString (wxVariant & *value*, int *argFlags* = 0) const [virtual]

Converts property value into a text representation.

Parameters

value	Value to be converted.
argFlags	If 0 (default value), then displayed string is returned. If wxPG_FULL_VALUE is set, returns complete, storable string value instead of displayable. If wxPG_EDITABLE_VALUE is set, returns string value that must be editable in textctrl. If wxPG_COMPOSITE_FRAGMENT is set, returns text that is appropriate to display as a part of string property's composite text representation.

Remarks

Default implementation calls GenerateComposedValue().

wxPGProperty::wxDEPRECATED (void *AddChildwxPGProperty* *prop)

Adds a private child property.

Deprecated Use AddPrivateChild() instead.

See also

AddPrivateChild()

wxPGProperty::wxDEPRECATED (wxString GetValueString(int argFlags=0) *const*)

Synonymous to GetValueAsString().

Deprecated Use GetValueAsString() instead.

See also

GetValueAsString()

21.528 wxPGValidationInfo Class Reference

`#include <wx/propgrid/propgrid.h>`

21.528.1 Detailed Description

wxPGValidationInfo

Used to convey validation information to and from functions that actually perform validation. Mostly used in custom property classes.

Public Member Functions

- wxPGVFBFlags GetFailureBehavior ()
- const wxString & GetFailureMessage () const

  Returns current failure message.
- wxVariant & GetValue ()

  Returns reference to pending value.
- void SetFailureBehavior (wxPGVFBFlags failureBehavior)

  Set validation failure behaviour.
- void SetFailureMessage (const wxString &message)

  Set current failure message.

21.528.2 Member Function Documentation

wxPGVFBFlags wxPGValidationInfo::GetFailureBehavior ( )

Returns

Returns failure behaviour which is a combination of wxPropertyGrid Validation Failure behaviour Flags.

const wxString& wxPGValidationInfo::GetFailureMessage ( ) const

Returns current failure message.

wxVariant& wxPGValidationInfo::GetValue ( )

Returns reference to pending value.

void wxPGValidationInfo::SetFailureBehavior (wxPGVFBFlags *failureBehavior*)

Set validation failure behaviour.

Parameters

failureBehavior	Mixture of wxPropertyGrid Validation Failure behaviour Flags.

void wxPGValidationInfo::SetFailureMessage (const wxString & *message*)

Set current failure message.

21.529 wxPGVIterator Class Reference

`#include <wx/propgrid/propgridpagestate.h>`

21.529.1 Detailed Description

21.529.2 wxPGVIterator

Abstract implementation of a simple iterator. Can only be used to iterate in forward order, and only through the entire container. Used to have functions dealing with all properties work with both wxPropertyGrid and wxPropertyGrid↩ Manager.

Public Member Functions

- wxPGVIterator ()
- wxPGVIterator (wxPGVIteratorBase ∗obj)
- ∼wxPGVIterator ()
- void UnRef ()
- wxPGVIterator (const wxPGVIterator &it)
- const wxPGVIterator & operator= (const wxPGVIterator &it)
- void Next ()
- bool AtEnd () const
- wxPGProperty ∗ GetProperty () const

Protected Attributes

- wxPGVIteratorBase ∗ m_pIt

21.529.3 Constructor & Destructor Documentation

wxPGVIterator::wxPGVIterator ( ) [inline]

wxPGVIterator::wxPGVIterator (wxPGVIteratorBase ∗ *obj*) [inline]

wxPGVIterator::~wxPGVIterator ( ) [inline]

wxPGVIterator::wxPGVIterator (const wxPGVIterator & *it*) [inline]

21.529.4 Member Function Documentation

bool wxPGVIterator::AtEnd ( ) const [inline]

wxPGProperty* wxPGVIterator::GetProperty ( ) const [inline]

void wxPGVIterator::Next ( ) [inline]

const wxPGVIterator& wxPGVIterator::operator= (const wxPGVIterator & *it*) [inline]

void wxPGVIterator::UnRef ( ) [inline]

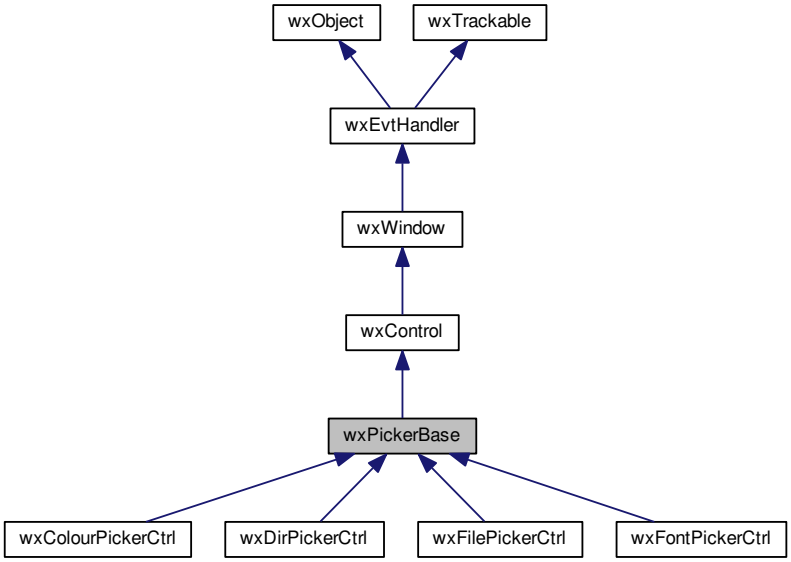
21.529.5 Member Data Documentation

wxPGVIteratorBase* wxPGVIterator::m_plt [protected]

21.530 wxPickerBase Class Reference

`#include <wx/pickerbase.h>`

Inheritance diagram for wxPickerBase:



21.530.1 Detailed Description

Base abstract class for all pickers which support an auxiliary text control.

This class handles all positioning and sizing of the text control like a an horizontal wxBoxSizer would do, with the text control on the left of the picker button.

The proportion (see wxSizer documentation for more info about proportion values) of the picker control defaults to 1 when there isn't a text control associated (see `wxPB_USE_TEXTCTRL` style) and to 0 otherwise.

Styles

This class supports the following styles:

- wxPB_USE_TEXTCTRL: Creates a text control to the left of the picker which is completely managed by this wxPickerBase class.

Library: wxCore

Category: Picker Controls

See also

wxColourPickerCtrl

Public Member Functions

- wxPickerBase ()
- virtual ~wxPickerBase ()
- bool CreateBase (wxWindow *parent, wxWindowID id, const wxString &text=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxButtonNameStr)
- int GetInternalMargin () const

  Returns the margin (in pixel) between the picker and the text control.
- int GetPickerCtrlProportion () const

  Returns the proportion value of the picker.
- wxTextCtrl * GetTextCtrl ()

  Returns a pointer to the text control handled by this window or NULL if the `wxPB_USE_TEXTCTRL` style was not specified when this control was created.
- wxControl * GetPickerCtrl ()

  Returns the native implementation of the real picker control.
- int GetTextCtrlProportion () const

  Returns the proportion value of the text control.
- bool HasTextCtrl () const

  Returns true if this window has a valid text control (i.e. if the `wxPB_USE_TEXTCTRL` style was given when creating this control).
- bool IsPickerCtrlGrowable () const

  Returns true if the picker control is growable.
- bool IsTextCtrlGrowable () const

  Returns true if the text control is growable.
- void SetInternalMargin (int margin)

  Sets the margin (in pixel) between the picker and the text control.
- void SetPickerCtrlGrowable (bool grow=true)

  Sets the picker control as growable when `grow` is true.
- void SetPickerCtrlProportion (int prop)

  Sets the proportion value of the picker.
- void SetTextCtrlGrowable (bool grow=true)

  Sets the text control as growable when `grow` is true.

- void SetTextCtrlProportion (int prop)

  Sets the proportion value of the text control.
- void SetTextCtrl (wxTextCtrl *text)
- void SetPickerCtrl (wxControl *picker)
- virtual void UpdatePickerFromTextCtrl ()=0
- virtual void UpdateTextCtrlFromPicker ()=0

Protected Member Functions

- virtual long GetTextCtrlStyle (long style) const
- virtual long GetPickerStyle (long style) const
- void PostCreation ()

Additional Inherited Members

21.530.2 Constructor & Destructor Documentation

wxPickerBase::wxPickerBase ( )

virtual wxPickerBase::~wxPickerBase ( ) `[virtual]`

21.530.3 Member Function Documentation

bool wxPickerBase::CreateBase (wxWindow * *parent*, wxWindowID *id*, const wxString & *text* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxButtonNameStr)

int wxPickerBase::GetInternalMargin ( ) const

Returns the margin (in pixel) between the picker and the text control.

This function can be used only when HasTextCtrl() returns true.

wxControl* wxPickerBase::GetPickerCtrl ( )

Returns the native implementation of the real picker control.

Note

The returned control in the generic implementation of wxFilePickerCtrl, wxDirPickerCtrl, wxFontPickerCtrl and wxColourPickerCtrl is a specialized wxButton class so that you can change its label doing, e.g.:

```
#ifdef __WXMSW__
    // wxMSW is one of the platforms where the generic implementation
    // of wxFilePickerCtrl is used...

    wxButton *pButt = wx_static_cast(wxButton*, myFilePickerCtrl->
      GetPickerCtrl());
    if (pButt)
        pButt->SetLabel("Custom browse string");
#endif
```

int wxPickerBase::GetPickerCtrlProportion ( ) const

Returns the proportion value of the picker.

virtual long wxPickerBase::GetPickerStyle (long *style*) const `[protected], [virtual]`

wxTextCtrl* wxPickerBase::GetTextCtrl ( )

Returns a pointer to the text control handled by this window or NULL if the `wxPB_USE_TEXTCTRL` style was not specified when this control was created.

Remarks

The contents of the text control could be an invalid representation of the entity which can be chosen through the picker (e.g. when the user enters an invalid colour syntax because of a typo). Thus you should never parse the content of the textctrl to get the user's input; rather use the derived-class getter (e.g. wxColourPickerCtrl::GetColour(), wxFilePickerCtrl::GetPath(), etc).

int wxPickerBase::GetTextCtrlProportion ( ) const

Returns the proportion value of the text control.

This function can be used only when HasTextCtrl() returns true.

virtual long wxPickerBase::GetTextCtrlStyle (long *style*) const `[protected], [virtual]`

bool wxPickerBase::HasTextCtrl ( ) const

Returns true if this window has a valid text control (i.e. if the `wxPB_USE_TEXTCTRL` style was given when creating this control).

bool wxPickerBase::IsPickerCtrlGrowable ( ) const

Returns true if the picker control is growable.

bool wxPickerBase::IsTextCtrlGrowable ( ) const

Returns true if the text control is growable.

This function can be used only when HasTextCtrl() returns true.

void wxPickerBase::PostCreation ( ) `[protected]`

void wxPickerBase::SetInternalMargin (int *margin*)

Sets the margin (in pixel) between the picker and the text control.

This function can be used only when HasTextCtrl() returns true.

void wxPickerBase::SetPickerCtrl (wxControl * *picker*)

void wxPickerBase::SetPickerCtrlGrowable (bool *grow* = `true`)

Sets the picker control as growable when `grow` is true.

void wxPickerBase::SetPickerCtrlProportion (int *prop*)

Sets the proportion value of the picker.

Look at the detailed description of wxPickerBase for more info.

void wxPickerBase::SetTextCtrl (wxTextCtrl * *text*)

void wxPickerBase::SetTextCtrlGrowable (bool *grow* = `true`)

Sets the text control as growable when `grow` is true.

This function can be used only when HasTextCtrl() returns true.

void wxPickerBase::SetTextCtrlProportion (int *prop*)

Sets the proportion value of the text control.

Look at the detailed description of wxPickerBase for more info.

This function can be used only when HasTextCtrl() returns true.

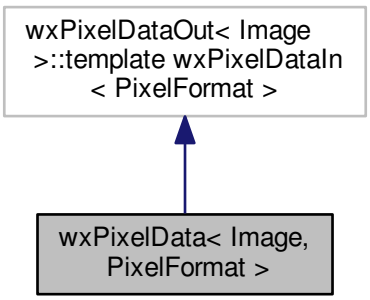
virtual void wxPickerBase::UpdatePickerFromTextCtrl ( ) [pure virtual]

virtual void wxPickerBase::UpdateTextCtrlFromPicker ( ) [pure virtual]

21.531 wxPixelData< Image, PixelFormat > Class Template Reference

`#include <wx/rawbmp.h>`

Inheritance diagram for wxPixelData< Image, PixelFormat >:



21.531.1 Detailed Description

template<class Image, class PixelFormat = wxPixelFormatFor<Image>>class wxPixelData< Image, PixelFormat >

A class template with ready to use implementations for getting direct and efficient access to wxBitmap's internal data and wxImage's internal data through a standard interface.

It is possible to extend this class (interface) to other types of image content.

Implemented on Windows, GTK+ and OS X:

- wxNativePixelData: Class to access to wxBitmap's internal data without alpha channel (RGB).
- wxAlphaPixelData: Class to access to wxBitmap's internal data with alpha channel (RGBA).

Implemented everywhere:

- wxImagePixelData: Class to access to wxImage's internal data with alpha channel (RGBA).

wxMSW note: efficient access is only possible to the bits of the so called device independent bitmaps (DIB) under MSW. To ensure that wxBitmap uses a DIB internally and not a device dependent bitmap (DDB), you need to pass an explicit depth to its ctor, i.e. either 24 or 32, as by default wxBitmap creates a DDB of the screen depth.

Example:

```
wxBitmap bmp(width, size, 24); // explicit depth important under MSW
wxNativePixelData data(bmp);
if ( !data )
{
    // ... raw access to bitmap data unavailable, do something else ...
    return;
}

if ( data.GetWidth() < 20 || data.GetHeight() < 20 )
{
    // ... complain: the bitmap it too small ...
    return;
}

wxNativePixelData::Iterator p(data);

// we draw a (10, 10)-(20, 20) rect manually using the given r, g, b
p.Offset(data, 10, 10);

for ( int y = 0; y < 10; ++y )
{
    wxNativePixelData::Iterator rowStart = p;

    for ( int x = 0; x < 10; ++x, ++p )
    {
        p.Red() = r;
        p.Green() = g;
        p.Blue() = b;
    }

    p = rowStart;
    p.OffsetY(data, 1);
}
```

Library: wxCore

Category: Graphics Device Interface (GDI)

See also

wxBitmap, wxImage

Classes

- class Iterator

  The iterator of class wxPixelData.

Public Types

- typedef Image ImageType

  The type of the class we're working with.

Public Member Functions

- wxPixelData (Image &image)

  Create pixel data object representing the entire image.
- wxPixelData (Image &i, const wxRect &rect)

  Create pixel data object representing the area of the image defined by rect.
- wxPixelData (Image &i, const wxPoint &pt, const wxSize &sz)

  Create pixel data object representing the area of the image defined by pt and sz.
- operator bool () const

  Return true of if we could get access to bitmap data successfully.
- Iterator GetPixels () const

  Return the iterator pointing to the origin of the image.
- wxPoint GetOrigin () const

  Returns origin of the rectangular region this wxPixelData represents.
- int GetWidth () const

  Return width of the region this wxPixelData represents.
- int GetHeight () const

  Return height of the region this wxPixelData represents.
- wxSize GetSize () const

  Return the area which this wxPixelData represents in the image.
- int GetRowStride () const

  Return the distance between two rows.

21.531.2 Member Typedef Documentation

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> typedef Image wxPixelData< Image, PixelFormat >::ImageType

The type of the class we're working with.

21.531.3 Constructor & Destructor Documentation

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::wxPixelData (Image & *image*)

Create pixel data object representing the entire image.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::wxPixelData (Image & *i,* const wxRect & *rect*)

Create pixel data object representing the area of the image defined by *rect*.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::wxPixelData (Image & *i,* const wxPoint & *pt,* const wxSize & *sz*)

Create pixel data object representing the area of the image defined by *pt* and *sz*.

21.531.4 Member Function Documentation

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> int wxPixelData< Image, PixelFormat >::GetHeight ( ) const

Return height of the region this wxPixelData represents.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPoint wxPixelData< Image, PixelFormat >::GetOrigin ( ) const

Returns origin of the rectangular region this wxPixelData represents.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> Iterator wxPixelData< Image, PixelFormat >::GetPixels ( ) const

Return the iterator pointing to the origin of the image.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> int wxPixelData< Image, PixelFormat >::GetRowStride ( ) const

Return the distance between two rows.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxSize wxPixelData< Image, PixelFormat >::GetSize ( ) const

Return the area which this wxPixelData represents in the image.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> int wxPixelData< Image, PixelFormat >::GetWidth ( ) const

Return width of the region this wxPixelData represents.

template<class Image , class PixelFormat = wxPixelFormatFor<Image>> wxPixelData< Image, PixelFormat >::operator bool ( ) const

Return true of if we could get access to bitmap data successfully.

21.532 wxPlatformInfo Class Reference

`#include <wx/platinfo.h>`

21.532.1 Detailed Description

This class holds information about the operating system, the toolkit and the basic architecture of the machine where the application is currently running.

This class does not only have *getters* for the information above, it also has *setters*. This allows you to e.g. save the current platform information in a data file (maybe in string form) so that when you later load it, you can easily retrieve (see the static getters for string->enum conversion functions) and store inside a wxPlatformInfo instance (using its setters) the signature of the system which generated it.

In general however you only need to use the static Get() function and then access the various information for the current platform:

```
wxLogMessage("This application is running under %s.",
             wxPlatformInfo::Get().GetOperatingSystemIdName());
```

Library: wxBase

Category: Application and System configuration

See also

wxGetOsVersion(), wxIsPlatformLittleEndian(), wxIsPlatform64Bit(), wxAppTraits, Network, User and OS

Public Member Functions

- wxPlatformInfo ()

  Initializes the instance with the values corresponding to the currently running platform.
- wxPlatformInfo (wxPortId pid, int tkMajor=-1, int tkMinor=-1, wxOperatingSystemId id=wxOS_UNKNOWN, int osMajor=-1, int osMinor=-1, wxArchitecture arch=wxARCH_INVALID, wxEndianness endian=wxENDIAN_↩ INVALID)

  Initializes the object using given values.
- bool CheckOSVersion (int major, int minor) const

  Returns true if the OS version is at least `major.minor`.
- bool CheckToolkitVersion (int major, int minor) const

  Returns true if the toolkit version is at least `major.minor`.
- bool IsOk () const

  Returns true if this instance is fully initialized with valid values.
- bool IsUsingUniversalWidgets () const

  Returns true if this wxPlatformInfo describes wxUniversal build.
- bool operator!= (const wxPlatformInfo &t) const

  Inequality operator.
- bool operator== (const wxPlatformInfo &t) const

  Equality operator.

Getters

- wxArchitecture GetArchitecture () const

  Returns the architecture ID of this wxPlatformInfo instance.
- wxEndianness GetEndianness () const

  Returns the endianness ID of this wxPlatformInfo instance.
- int GetOSMajorVersion () const

  Returns the run-time major version of the OS associated with this wxPlatformInfo instance.
- int GetOSMinorVersion () const

  Returns the run-time minor version of the OS associated with this wxPlatformInfo instance.
- wxOperatingSystemId GetOperatingSystemId () const

  Returns the operating system ID of this wxPlatformInfo instance.
- wxString GetOperatingSystemDescription () const

  Returns the description of the operating system of this wxPlatformInfo instance.
- wxPortId GetPortId () const

  Returns the wxWidgets port ID associated with this wxPlatformInfo instance.
- wxLinuxDistributionInfo GetLinuxDistributionInfo () const

  Returns the Linux distribution info associated with this wxPlatformInfo instance.
- wxString GetDesktopEnvironment () const

Returns the desktop environment associated with this wxPlatformInfo instance.

- int GetToolkitMajorVersion () const

  Returns the run-time major version of the toolkit associated with this wxPlatformInfo instance.
- int GetToolkitMinorVersion () const

  Returns the run-time minor version of the toolkit associated with this wxPlatformInfo instance.

String-form getters

- wxString GetArchName () const

  Returns the name for the architecture of this wxPlatformInfo instance.
- wxString GetEndiannessName () const

  Returns the name for the endianness of this wxPlatformInfo instance.
- wxString GetOperatingSystemFamilyName () const

  Returns the operating system family name of the OS associated with this wxPlatformInfo instance.
- wxString GetOperatingSystemIdName () const

  Returns the operating system name of the OS associated with this wxPlatformInfo instance.
- wxString GetPortIdName () const

  Returns the name of the wxWidgets port ID associated with this wxPlatformInfo instance.
- wxString GetPortIdShortName () const

  Returns the short name of the wxWidgets port ID associated with this wxPlatformInfo instance.

Setters

- void SetArchitecture (wxArchitecture n)

  Sets the architecture enum value associated with this wxPlatformInfo instance.
- void SetEndianness (wxEndianness n)

  Sets the endianness enum value associated with this wxPlatformInfo instance.
- void SetOSVersion (int major, int minor)

  Sets the version of the operating system associated with this wxPlatformInfo instance.
- void SetOperatingSystemId (wxOperatingSystemId n)

  Sets the operating system associated with this wxPlatformInfo instance.
- void SetPortId (wxPortId n)

  Sets the wxWidgets port ID associated with this wxPlatformInfo instance.
- void SetToolkitVersion (int major, int minor)

  Sets the version of the toolkit associated with this wxPlatformInfo instance.
- void SetOperatingSystemDescription (const wxString &desc)

  Sets the operating system description associated with this wxPlatformInfo instance.
- void SetDesktopEnvironment (const wxString &de)

  Sets the desktop environment associated with this wxPlatformInfo instance.
- void SetLinuxDistributionInfo (const wxLinuxDistributionInfo &di)

  Sets the linux distribution info associated with this wxPlatformInfo instance.

Static Public Member Functions

- static const wxPlatformInfo & Get ()

  Returns the global wxPlatformInfo object, initialized with the values for the currently running platform.

Static enum getters

These getters allow for easy string-to-enumeration-value conversion.

- static wxArchitecture GetArch (const wxString &arch)

  Converts the given string to a wxArchitecture enum value or to `wxARCH_INVALID` if the given string is not a valid architecture string (i.e.
- static wxEndianness GetEndianness (const wxString &end)

  Converts the given string to a wxEndianness enum value or to `wxENDIAN_INVALID` if the given string is not a valid endianness string (i.e.
- static wxOperatingSystemId GetOperatingSystemId (const wxString &name)

Converts the given string to a wxOperatingSystemId enum value or to `wxOS_UNKNOWN` if the given string is not a valid operating system name.

- static wxPortId GetPortId (const wxString &portname)

  Converts the given string to a wxWidgets port ID value or to `wxPORT_UNKNOWN` if the given string does not match any of the wxWidgets canonical name ports ("wxGTK", "wxMSW", etc) nor any of the short wxWidgets name ports ("gtk", "msw", etc).

Static string-form getters

These getters allow for easy enumeration-value-to-string conversion.

- static wxString GetArchName (wxArchitecture arch)

  Returns the name for the given wxArchitecture enumeration value.
- static wxString GetEndiannessName (wxEndianness end)

  Returns name for the given wxEndianness enumeration value.
- static wxString GetOperatingSystemFamilyName (wxOperatingSystemId os)

  Returns the operating system family name for the given wxOperatingSystemId enumeration value: `Unix` for `wxOS_UNIX`, `Macintosh` for `wxOS_MAC`, `Windows` for `wxOS_WINDOWS`, `DOS` for `wxOS_DOS`, `OS/2` for `wxOS_OS2`.
- static wxString GetOperatingSystemIdName (wxOperatingSystemId os)

  Returns the name for the given operating system ID value.
- static wxString GetPortIdName (wxPortId port, bool usingUniversal)

  Returns the name of the given wxWidgets port ID value.
- static wxString GetPortIdShortName (wxPortId port, bool usingUniversal)

  Returns the short name of the given wxWidgets port ID value.
- static wxString GetOperatingSystemDirectory ()

  Returns the operating system directory.

21.532.2 Constructor & Destructor Documentation

wxPlatformInfo::wxPlatformInfo ( )

Initializes the instance with the values corresponding to the currently running platform.

This is a fast operation because it only requires to copy the values internally cached for the currently running platform.

See also

Get()

wxPlatformInfo::wxPlatformInfo (wxPortId *pid*, int *tkMajor* = −1, int *tkMinor* = −1, wxOperatingSystemId *id* = wxOS_UNKNOWN, int *osMajor* = −1, int *osMinor* = −1, wxArchitecture *arch* = wxARCH_INVALID, wxEndianness *endian* = wxENDIAN_INVALID)

Initializes the object using given values.

21.532.3 Member Function Documentation

bool wxPlatformInfo::CheckOSVersion (int *major*, int *minor*) const

Returns true if the OS version is at least `major.minor`.

See also

GetOSMajorVersion(), GetOSMinorVersion(), CheckToolkitVersion()

bool wxPlatformInfo::CheckToolkitVersion (int *major,* int *minor*) const

Returns true if the toolkit version is at least `major.minor`.

See also

GetToolkitMajorVersion(), GetToolkitMinorVersion(), CheckOSVersion()

static const wxPlatformInfo& wxPlatformInfo::Get ( ) [static]

Returns the global wxPlatformInfo object, initialized with the values for the currently running platform.

static wxArchitecture wxPlatformInfo::GetArch (const wxString & *arch*) [static]

Converts the given string to a wxArchitecture enum value or to `wxARCH_INVALID` if the given string is not a valid architecture string (i.e.

does not contain nor `32` nor `64` strings).

wxArchitecture wxPlatformInfo::GetArchitecture ( ) const

Returns the architecture ID of this wxPlatformInfo instance.

static wxString wxPlatformInfo::GetArchName (wxArchitecture *arch*) [static]

Returns the name for the given wxArchitecture enumeration value.

wxString wxPlatformInfo::GetArchName ( ) const

Returns the name for the architecture of this wxPlatformInfo instance.

wxString wxPlatformInfo::GetDesktopEnvironment ( ) const

Returns the desktop environment associated with this wxPlatformInfo instance.

See wxAppTraits::GetDesktopEnvironment() for more info.

static wxEndianness wxPlatformInfo::GetEndianness (const wxString & *end*) [static]

Converts the given string to a wxEndianness enum value or to `wxENDIAN_INVALID` if the given string is not a valid endianness string (i.e.

does not contain nor little nor big strings).

wxEndianness wxPlatformInfo::GetEndianness ( ) const

Returns the endianness ID of this wxPlatformInfo instance.

static wxString wxPlatformInfo::GetEndiannessName (wxEndianness *end*) [static]

Returns name for the given wxEndianness enumeration value.

wxString wxPlatformInfo::GetEndiannessName ( ) const

Returns the name for the endianness of this wxPlatformInfo instance.

wxLinuxDistributionInfo wxPlatformInfo::GetLinuxDistributionInfo ( ) const

Returns the Linux distribution info associated with this wxPlatformInfo instance.

See wxGetLinuxDistributionInfo() for more info.

wxString wxPlatformInfo::GetOperatingSystemDescription ( ) const

Returns the description of the operating system of this wxPlatformInfo instance.

See wxGetOSDescription() for more info.

static wxString wxPlatformInfo::GetOperatingSystemDirectory ( ) [static]

Returns the operating system directory.

See wxGetOSDirectory() for more info.

static wxString wxPlatformInfo::GetOperatingSystemFamilyName (wxOperatingSystemId *os*) [static]

Returns the operating system family name for the given wxOperatingSystemId enumeration value: `Unix` for `wxOS_UNIX`, `Macintosh` for `wxOS_MAC`, `Windows` for `wxOS_WINDOWS`, `DOS` for `wxOS_DOS`, `OS/2` for `wxOS_OS2`.

wxString wxPlatformInfo::GetOperatingSystemFamilyName ( ) const

Returns the operating system family name of the OS associated with this wxPlatformInfo instance.

static wxOperatingSystemId wxPlatformInfo::GetOperatingSystemId (const wxString & *name*) [static]

Converts the given string to a wxOperatingSystemId enum value or to `wxOS_UNKNOWN` if the given string is not a valid operating system name.

wxOperatingSystemId wxPlatformInfo::GetOperatingSystemId ( ) const

Returns the operating system ID of this wxPlatformInfo instance.

See wxGetOsVersion() for more info.

static wxString wxPlatformInfo::GetOperatingSystemIdName (wxOperatingSystemId *os*) [static]

Returns the name for the given operating system ID value.

This can be a long name (e.g. `Microsoft Windows NT`); use GetOperatingSystemFamilyName() to retrieve a short, generic name.

wxString wxPlatformInfo::GetOperatingSystemIdName ( ) const

Returns the operating system name of the OS associated with this wxPlatformInfo instance.

int wxPlatformInfo::GetOSMajorVersion ( ) const

Returns the run-time major version of the OS associated with this wxPlatformInfo instance.

See also

wxGetOsVersion(), CheckOSVersion()

int wxPlatformInfo::GetOSMinorVersion ( ) const

Returns the run-time minor version of the OS associated with this wxPlatformInfo instance.

See also

wxGetOsVersion(), CheckOSVersion()

static wxPortId wxPlatformInfo::GetPortId (const wxString & *portname*) [static]

Converts the given string to a wxWidgets port ID value or to `wxPORT_UNKNOWN` if the given string does not match any of the wxWidgets canonical name ports ("wxGTK", "wxMSW", etc) nor any of the short wxWidgets name ports ("gtk", "msw", etc).

wxPortId wxPlatformInfo::GetPortId ( ) const

Returns the wxWidgets port ID associated with this wxPlatformInfo instance.

static wxString wxPlatformInfo::GetPortIdName (wxPortId *port*, bool *usingUniversal*) [static]

Returns the name of the given wxWidgets port ID value.

The *usingUniversal* argument specifies whether the port is in its native or wxUniversal variant.

The returned string always starts with the "wx" prefix and is a mixed-case string.

wxString wxPlatformInfo::GetPortIdName ( ) const

Returns the name of the wxWidgets port ID associated with this wxPlatformInfo instance.

static wxString wxPlatformInfo::GetPortIdShortName (wxPortId *port*, bool *usingUniversal*) [static]

Returns the short name of the given wxWidgets port ID value.

The *usingUniversal* argument specifies whether the port is in its native or wxUniversal variant.

The returned string does not start with the "wx" prefix and is always lower case.

wxString wxPlatformInfo::GetPortIdShortName ( ) const

Returns the short name of the wxWidgets port ID associated with this wxPlatformInfo instance.

int wxPlatformInfo::GetToolkitMajorVersion ( ) const

Returns the run-time major version of the toolkit associated with this wxPlatformInfo instance.

Note that if GetPortId() returns `wxPORT_BASE`, then this value is zero (unless externally modified with SetToolkitVersion()); that is, no native toolkit is in use. See wxAppTraits::GetToolkitVersion() for more info.

See also

CheckToolkitVersion()

int wxPlatformInfo::GetToolkitMinorVersion ( ) const

Returns the run-time minor version of the toolkit associated with this wxPlatformInfo instance.

Note that if GetPortId() returns `wxPORT_BASE`, then this value is zero (unless externally modified with SetToolkitVersion()); that is, no native toolkit is in use. See wxAppTraits::GetToolkitVersion() for more info.

See also

CheckToolkitVersion()

bool wxPlatformInfo::IsOk ( ) const

Returns true if this instance is fully initialized with valid values.

bool wxPlatformInfo::IsUsingUniversalWidgets ( ) const

Returns true if this wxPlatformInfo describes wxUniversal build.

bool wxPlatformInfo::operator!= (const wxPlatformInfo & *t*) const

Inequality operator.

Tests all class' internal variables.

bool wxPlatformInfo::operator== (const wxPlatformInfo & *t*) const

Equality operator.

Tests all class' internal variables.

void wxPlatformInfo::SetArchitecture (wxArchitecture *n*)

Sets the architecture enum value associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetDesktopEnvironment (const wxString & *de*)

Sets the desktop environment associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetEndianness (wxEndianness *n*)

Sets the endianness enum value associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetLinuxDistributionInfo (const wxLinuxDistributionInfo & *di*)

Sets the linux distribution info associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetOperatingSystemDescription (const wxString & *desc*)

Sets the operating system description associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetOperatingSystemId (wxOperatingSystemId *n*)

Sets the operating system associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetOSVersion (int *major,* int *minor*)

Sets the version of the operating system associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetPortId (wxPortId *n*)

Sets the wxWidgets port ID associated with this wxPlatformInfo instance.

void wxPlatformInfo::SetToolkitVersion (int *major,* int *minor*)

Sets the version of the toolkit associated with this wxPlatformInfo instance.

21.533 wxPoint Class Reference

```
#include <wx/gdicmn.h>
```

21.533.1 Detailed Description

A wxPoint is a useful data structure for graphics operations.

It contains integer *x* and *y* members. See wxRealPoint for a floating point version.

Note that the width and height stored inside a wxPoint object may be negative and that wxPoint functions do not perform any check against negative values (this is used to e.g. store the special -1 value in wxDefaultPosition instance).

Library: wxCore

Category: Data Structures

Predefined objects/pointers: wxDefaultPosition

See also

wxRealPoint

Public Member Functions

- wxPoint ()

  Constructs a point.
- wxPoint (int x, int y)

  Initializes the point object with the given x and y coordinates.
- wxPoint (const wxRealPoint &pt)

  Converts the given wxRealPoint (with floating point coordinates) to a wxPoint instance.

Miscellaneous operators

Note that these operators are documented as class members (to make them easier to find) but, as their prototype shows, they are implemented as global operators; note that this is transparent to the user but it helps to understand why the following functions are documented to take the wxPoint they operate on as an explicit argument.

- wxPoint & operator= (const wxPoint &pt)
- bool operator== (const wxPoint &p1, const wxPoint &p2)
- bool operator!= (const wxPoint &p1, const wxPoint &p2)
- wxPoint operator+ (const wxPoint &p1, const wxPoint &p2)
- wxPoint operator- (const wxPoint &p1, const wxPoint &p2)
- wxPoint & operator+= (const wxPoint &pt)
- wxPoint & operator-= (const wxPoint &pt)
- wxPoint operator+ (const wxPoint &pt, const wxSize &sz)
- wxPoint operator- (const wxPoint &pt, const wxSize &sz)
- wxPoint operator+ (const wxSize &sz, const wxPoint &pt)
- wxPoint operator- (const wxSize &sz, const wxPoint &pt)
- wxPoint & operator+= (const wxSize &sz)
- wxPoint & operator-= (const wxSize &sz)
- wxSize operator/ (const wxPoint &sz, int factor)
- wxSize operator* (const wxPoint &sz, int factor)
- wxSize operator* (int factor, const wxSize &sz)
- wxSize & operator/= (int factor)
- wxSize & operator*= (int factor)

Defaults handling.

Test for and set non-specified wxPoint components.

Although a wxPoint is always initialized to (0, 0), wxWidgets commonly uses wxDefaultCoord (defined as `-1`) to indicate that a point hasn't been initialized or specified. In particular, wxDefaultPosition is used in many places with this meaning.

- bool IsFullySpecified () const

  Returns true if neither of the point components is equal to wxDefaultCoord.
- void SetDefaults (const wxPoint &pt)

  Combine this object with another one replacing the uninitialized values.

Public Attributes

- int x

  x member.
- int y

  y member.

21.533.2 Constructor & Destructor Documentation

wxPoint::wxPoint ( )

Constructs a point.

Initializes the internal x and y coordinates to zero.

wxPoint::wxPoint (int *x*, int *y*)

Initializes the point object with the given *x* and *y* coordinates.

wxPoint::wxPoint (const wxRealPoint & *pt*)

Converts the given wxRealPoint (with floating point coordinates) to a wxPoint instance.

Notice that this truncates the floating point values of *pt* components, if you want to round them instead you need to do it manually, e.g.

```
#include <wx/math.h>    // for wxRound()

wxRealPoint rp = ...;
wxPoint p(wxRound(rp.x), wxRound(rp.y));
```

21.533.3 Member Function Documentation

bool wxPoint::IsFullySpecified ( ) const

Returns true if neither of the point components is equal to wxDefaultCoord.

This method is typically used before calling SetDefaults().

Since

2.9.2

bool wxPoint::operator!= (const wxPoint & *p1*, const wxPoint & *p2*)

wxSize wxPoint::operator* (const wxPoint & *sz*, int *factor*)

wxSize wxPoint::operator* (int *factor*, const wxSize & *sz*)

wxSize& wxPoint::operator*= (int *factor*)

wxPoint wxPoint::operator+ (const wxPoint & *p1*, const wxPoint & *p2*)

wxPoint wxPoint::operator+ (const wxPoint & *pt*, const wxSize & *sz*)

wxPoint wxPoint::operator+ (const wxSize & *sz*, const wxPoint & *pt*)

wxPoint& wxPoint::operator+= (const wxPoint & *pt*)

wxPoint& wxPoint::operator+= (const wxSize & *sz*)

wxPoint wxPoint::operator- (const wxPoint & *p1*, const wxPoint & *p2*)

wxPoint wxPoint::operator- (const wxPoint & *pt*, const wxSize & *sz*)

wxPoint wxPoint::operator- (const wxSize & *sz*, const wxPoint & *pt*)

wxPoint& wxPoint::operator-= (const wxPoint & *pt*)

wxPoint& wxPoint::operator-= (const wxSize & *sz*)

wxSize wxPoint::operator/ (const wxPoint & *sz,* int *factor*)

wxSize& wxPoint::operator/= (int *factor*)

wxPoint& wxPoint::operator= (const wxPoint & *pt*)

bool wxPoint::operator== (const wxPoint & *p1,* const wxPoint & *p2*)

void wxPoint::SetDefaults (const wxPoint & *pt*)

Combine this object with another one replacing the uninitialized values.

It is typically used like this:

```
if ( !pos.IsFullySpecified() )
{
    pos.SetDefaults(GetDefaultPosition());
}
```

See also

IsFullySpecified()

Since

2.9.2

21.533.4 Member Data Documentation

int wxPoint::x

x member.

int wxPoint::y

y member.

21.534 wxPoint2DDouble Class Reference

`#include <wx/geometry.h>`

Public Member Functions

- wxPoint2DDouble ()
- wxPoint2DDouble (wxDouble x, wxDouble y)
- wxPoint2DDouble (const wxPoint2DDouble &pt)
- wxPoint2DDouble (const wxPoint2DInt &pt)
- wxPoint2DDouble (const wxPoint &pt)
- void GetFloor (wxInt32 *x, wxInt32 *y) const
- void GetRounded (wxInt32 *x, wxInt32 *y) const
- wxDouble GetVectorLength () const
- wxDouble GetVectorAngle () const
- void SetVectorLength (wxDouble length)
- void SetVectorAngle (wxDouble degrees)

- void SetPolarCoordinates (wxDouble angle, wxDouble length)
- void Normalize ()
- wxDouble GetDistance (const wxPoint2DDouble &pt) const
- wxDouble GetDistanceSquare (const wxPoint2DDouble &pt) const
- wxDouble GetDotProduct (const wxPoint2DDouble &vec) const
- wxDouble GetCrossProduct (const wxPoint2DDouble &vec) const
- wxPoint2DDouble operator- ()
- wxPoint2DDouble & operator= (const wxPoint2DDouble &pt)
- wxPoint2DDouble & operator+= (const wxPoint2DDouble &pt)
- wxPoint2DDouble & operator-= (const wxPoint2DDouble &pt)
- wxPoint2DDouble & operator*= (const wxPoint2DDouble &pt)
- wxPoint2DDouble & operator*= (wxDouble n)
- wxPoint2DDouble & operator*= (wxInt32 n)
- wxPoint2DDouble & operator/= (const wxPoint2DDouble &pt)
- wxPoint2DDouble & operator/= (wxDouble n)
- wxPoint2DDouble & operator/= (wxInt32 n)
- bool operator== (const wxPoint2DDouble &pt) const
- bool operator!= (const wxPoint2DDouble &pt) const

Public Attributes

- wxDouble m_x
- wxDouble m_y

21.534.1 Constructor & Destructor Documentation

wxPoint2DDouble::wxPoint2DDouble ( )

wxPoint2DDouble::wxPoint2DDouble (wxDouble *x*, wxDouble *y*)

wxPoint2DDouble::wxPoint2DDouble (const wxPoint2DDouble & *pt*)

wxPoint2DDouble::wxPoint2DDouble (const wxPoint2DInt & *pt*)

wxPoint2DDouble::wxPoint2DDouble (const wxPoint & *pt*)

21.534.2 Member Function Documentation

wxDouble wxPoint2DDouble::GetCrossProduct (const wxPoint2DDouble & *vec*) const

wxDouble wxPoint2DDouble::GetDistance (const wxPoint2DDouble & *pt*) const

wxDouble wxPoint2DDouble::GetDistanceSquare (const wxPoint2DDouble & *pt*) const

wxDouble wxPoint2DDouble::GetDotProduct (const wxPoint2DDouble & *vec*) const

void wxPoint2DDouble::GetFloor (wxInt32 * *x*, wxInt32 * *y*) const

void wxPoint2DDouble::GetRounded (wxInt32 * *x*, wxInt32 * *y*) const

wxDouble wxPoint2DDouble::GetVectorAngle ( ) const

wxDouble wxPoint2DDouble::GetVectorLength ( ) const

void wxPoint2DDouble::Normalize ( )

bool wxPoint2DDouble::operator!= (const wxPoint2DDouble & *pt*) const

wxPoint2DDouble& wxPoint2DDouble::operator*= (const wxPoint2DDouble & *pt*)

wxPoint2DDouble& wxPoint2DDouble::operator*= (wxDouble *n*)

wxPoint2DDouble& wxPoint2DDouble::operator*= (wxInt32 *n*)

wxPoint2DDouble& wxPoint2DDouble::operator+= (const wxPoint2DDouble & *pt*)

wxPoint2DDouble wxPoint2DDouble::operator- ( )

wxPoint2DDouble& wxPoint2DDouble::operator-= (const wxPoint2DDouble & *pt*)

wxPoint2DDouble& wxPoint2DDouble::operator/= (const wxPoint2DDouble & *pt*)

wxPoint2DDouble& wxPoint2DDouble::operator/= (wxDouble *n*)

wxPoint2DDouble& wxPoint2DDouble::operator/= (wxInt32 *n*)

wxPoint2DDouble& wxPoint2DDouble::operator= (const wxPoint2DDouble & *pt*)

bool wxPoint2DDouble::operator== (const wxPoint2DDouble & *pt*) const

void wxPoint2DDouble::SetPolarCoordinates (wxDouble *angle,* wxDouble *length*)

void wxPoint2DDouble::SetVectorAngle (wxDouble *degrees*)

void wxPoint2DDouble::SetVectorLength (wxDouble *length*)

21.534.3 Member Data Documentation

wxDouble wxPoint2DDouble::m_x

wxDouble wxPoint2DDouble::m_y

21.535 wxPoint2DInt Class Reference

`#include <wx/geometry.h>`

Public Member Functions

- wxPoint2DInt ()
- wxPoint2DInt (wxInt32 x, wxInt32 y)
- wxPoint2DInt (const wxPoint2DInt &pt)
- wxPoint2DInt (const wxPoint &pt)
- void GetFloor (wxInt32 *x, wxInt32 *y) const
- void GetRounded (wxInt32 *x, wxInt32 *y) const
- wxDouble GetVectorLength () const
- wxDouble GetVectorAngle () const
- void SetVectorLength (wxDouble length)
- void SetVectorAngle (wxDouble degrees)
- void SetPolarCoordinates (wxInt32 angle, wxInt32 length)

- void Normalize ()
- wxDouble GetDistance (const wxPoint2DInt &pt) const
- wxDouble GetDistanceSquare (const wxPoint2DInt &pt) const
- wxInt32 GetDotProduct (const wxPoint2DInt &vec) const
- wxInt32 GetCrossProduct (const wxPoint2DInt &vec) const
- wxPoint2DInt operator- ()
- wxPoint2DInt & operator= (const wxPoint2DInt &pt)
- wxPoint2DInt & operator+= (const wxPoint2DInt &pt)
- wxPoint2DInt & operator-= (const wxPoint2DInt &pt)
- wxPoint2DInt & operator*= (const wxPoint2DInt &pt)
- wxPoint2DInt & operator*= (wxDouble n)
- wxPoint2DInt & operator*= (wxInt32 n)
- wxPoint2DInt & operator/= (const wxPoint2DInt &pt)
- wxPoint2DInt & operator/= (wxDouble n)
- wxPoint2DInt & operator/= (wxInt32 n)
- operator wxPoint () const
- bool operator== (const wxPoint2DInt &pt) const
- bool operator!= (const wxPoint2DInt &pt) const

Public Attributes

- wxInt32 m_x
- wxInt32 m_y

21.535.1 Constructor & Destructor Documentation

wxPoint2DInt::wxPoint2DInt ( )

wxPoint2DInt::wxPoint2DInt (wxInt32 *x*, wxInt32 *y*)

wxPoint2DInt::wxPoint2DInt (const wxPoint2DInt & *pt*)

wxPoint2DInt::wxPoint2DInt (const wxPoint & *pt*)

21.535.2 Member Function Documentation

wxInt32 wxPoint2DInt::GetCrossProduct (const wxPoint2DInt & *vec*) const

wxDouble wxPoint2DInt::GetDistance (const wxPoint2DInt & *pt*) const

wxDouble wxPoint2DInt::GetDistanceSquare (const wxPoint2DInt & *pt*) const

wxInt32 wxPoint2DInt::GetDotProduct (const wxPoint2DInt & *vec*) const

void wxPoint2DInt::GetFloor (wxInt32 * *x*, wxInt32 * *y*) const

void wxPoint2DInt::GetRounded (wxInt32 * *x*, wxInt32 * *y*) const

wxDouble wxPoint2DInt::GetVectorAngle ( ) const

wxDouble wxPoint2DInt::GetVectorLength ( ) const

void wxPoint2DInt::Normalize ( )

wxPoint2DInt::operator **wxPoint** ( ) const

bool wxPoint2DInt::operator!= (const **wxPoint2DInt** & *pt*) const

wxPoint2DInt& wxPoint2DInt::operator*= (const **wxPoint2DInt** & *pt*)

wxPoint2DInt& wxPoint2DInt::operator*= (**wxDouble** *n*)

wxPoint2DInt& wxPoint2DInt::operator*= (**wxInt32** *n*)

wxPoint2DInt& wxPoint2DInt::operator+= (const **wxPoint2DInt** & *pt*)

wxPoint2DInt wxPoint2DInt::operator- ( )

wxPoint2DInt& wxPoint2DInt::operator-= (const **wxPoint2DInt** & *pt*)

wxPoint2DInt& wxPoint2DInt::operator/= (const **wxPoint2DInt** & *pt*)

wxPoint2DInt& wxPoint2DInt::operator/= (**wxDouble** *n*)

wxPoint2DInt& wxPoint2DInt::operator/= (**wxInt32** *n*)

wxPoint2DInt& wxPoint2DInt::operator= (const **wxPoint2DInt** & *pt*)

bool wxPoint2DInt::operator== (const **wxPoint2DInt** & *pt*) const

void wxPoint2DInt::SetPolarCoordinates (**wxInt32** *angle,* **wxInt32** *length*)

void wxPoint2DInt::SetVectorAngle (**wxDouble** *degrees*)

void wxPoint2DInt::SetVectorLength (**wxDouble** *length*)

21.535.3 Member Data Documentation

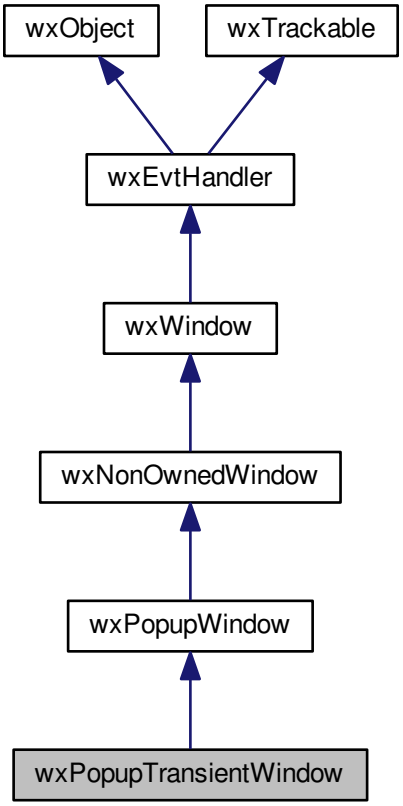
wxInt32 wxPoint2DInt::m_x

wxInt32 wxPoint2DInt::m_y

21.536 wxPopupTransientWindow Class Reference

`#include <wx/popupwin.h>`

Inheritance diagram for wxPopupTransientWindow:



21.536.1 Detailed Description

A wxPopupWindow which disappears automatically when the user clicks mouse outside it or if it loses focus in any other way.

This window can be useful for implementing custom combobox-like controls for example.

Library: wxCore

Category: Managed Windows

See also

wxPopupWindow

Public Member Functions

- wxPopupTransientWindow ()

  Default constructor.

- wxPopupTransientWindow (wxWindow *parent, int flags=wxBORDER_NONE)
  Constructor.
- virtual void Popup (wxWindow *focus=NULL)
  Popup the window (this will show it too).
- virtual void Dismiss ()
  Hide the window.
- virtual bool ProcessLeftDown (wxMouseEvent &event)
  Called when a mouse is pressed while the popup is shown.

Protected Member Functions

- virtual void OnDismiss ()
  This is called when the popup is disappeared because of anything else but direct call to Dismiss().

Additional Inherited Members

21.536.2 Constructor & Destructor Documentation

wxPopupTransientWindow::wxPopupTransientWindow ( )

Default constructor.

wxPopupTransientWindow::wxPopupTransientWindow (wxWindow * *parent,* int *flags* = wxBORDER_NONE)

Constructor.

21.536.3 Member Function Documentation

virtual void wxPopupTransientWindow::Dismiss ( ) [virtual]

Hide the window.

virtual void wxPopupTransientWindow::OnDismiss ( ) [protected], [virtual]

This is called when the popup is disappeared because of anything else but direct call to Dismiss().

virtual void wxPopupTransientWindow::Popup (wxWindow * *focus* = NULL) [virtual]

Popup the window (this will show it too).

If *winFocus* is non-NULL, it will be kept focused while this window is shown, otherwise this window itself will receive focus. In any case, the popup will disappear automatically if it loses focus because of a user action.

See also

Dismiss()

virtual bool wxPopupTransientWindow::ProcessLeftDown (wxMouseEvent & *event*) [virtual]

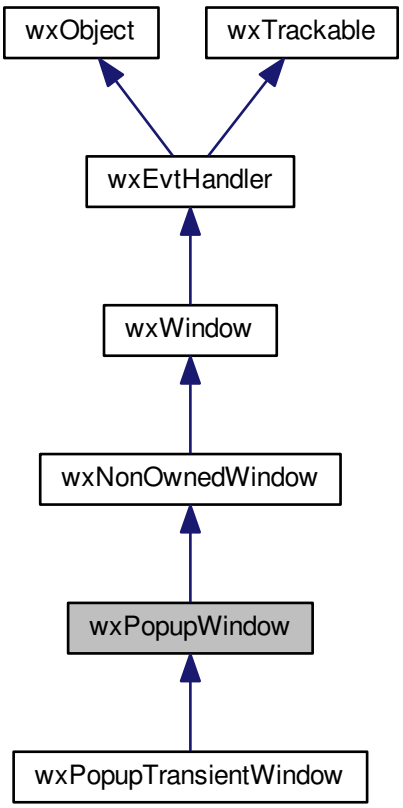
Called when a mouse is pressed while the popup is shown.

Return true from here to prevent its normal processing by the popup (which consists in dismissing it if the mouse is clicked outside it).

21.537 wxPopupWindow Class Reference

`#include <wx/popupwin.h>`

Inheritance diagram for wxPopupWindow:



21.537.1 Detailed Description

A special kind of top level window used for popup menus, combobox popups and such.

Library: wxCore

Category: Managed Windows

See also

wxDialog, wxFrame

Public Member Functions

- wxPopupWindow ()

Default constructor.

- wxPopupWindow (wxWindow *parent, int flags=wxBORDER_NONE)

  Constructor.

- bool Create (wxWindow *parent, int flags=wxBORDER_NONE)

  Create method for two-step creation.

- virtual void Position (const wxPoint &ptOrigin, const wxSize &sizePopup)

  Move the popup window to the right position, i.e. such that it is entirely visible.

Additional Inherited Members

21.537.2 Constructor & Destructor Documentation

wxPopupWindow::wxPopupWindow ( )

Default constructor.

wxPopupWindow::wxPopupWindow (wxWindow * *parent,* int *flags* = wxBORDER_NONE)

Constructor.

21.537.3 Member Function Documentation

bool wxPopupWindow::Create (wxWindow * *parent,* int *flags* = wxBORDER_NONE)

Create method for two-step creation.

virtual void wxPopupWindow::Position (const wxPoint & *ptOrigin,* const wxSize & *sizePopup*) `[virtual]`

Move the popup window to the right position, i.e. such that it is entirely visible.

The popup is positioned at ptOrigin + size if it opens below and to the right (default), at ptOrigin - sizePopup if it opens above and to the left etc.

Parameters

ptOrigin	Must be given in screen coordinates!
sizePopup	The size of the popup window

21.538 wxPosition Class Reference

`#include <wx/position.h>`

21.538.1 Detailed Description

This class represents the position of an item in any kind of grid of rows and columns such as wxGridBagSizer, or wxHVScrolledWindow.

Library: wxBase

Category: Data Structures

See also

wxPoint, wxSize

Public Member Functions

- wxPosition ()

  Construct a new wxPosition, setting the row and column to the default value of (0, 0).
- wxPosition (int row, int col)

  Construct a new wxPosition, setting the row and column to the value of (row, col).
- int GetCol () const

  A synonym for GetColumn().
- int GetColumn () const

  Get the current row value.
- int GetRow () const

  Get the current row value.
- void SetCol (int column)

  A synonym for SetColumn().
- void SetColumn (int column)

  Set a new column value.
- void SetRow (int row)

  Set a new row value.

Miscellaneous operators

- bool operator== (const wxPosition &pos) const
- bool operator!= (const wxPosition &pos) const
- wxPosition & operator+= (const wxPosition &pos)
- wxPosition & operator-= (const wxPosition &pos)
- wxPosition & operator+= (const wxSize &size)
- wxPosition & operator-= (const wxSize &size)
- wxPosition operator+ (const wxPosition &pos) const
- wxPosition operator- (const wxPosition &pos) const
- wxPosition operator+ (const wxSize &size) const
- wxPosition operator- (const wxSize &size) const

21.538.2 Constructor & Destructor Documentation

wxPosition::wxPosition ( )

Construct a new wxPosition, setting the row and column to the default value of (0, 0).

wxPosition::wxPosition (int *row,* int *col*)

Construct a new wxPosition, setting the row and column to the value of (*row*, *col*).

21.538.3 Member Function Documentation

int wxPosition::GetCol ( ) const

A synonym for GetColumn().

int wxPosition::GetColumn ( ) const

Get the current row value.

int wxPosition::GetRow ( ) const

Get the current row value.

bool wxPosition::operator!= (const wxPosition & *pos*) const

wxPosition wxPosition::operator+ (const wxPosition & *pos*) const

wxPosition wxPosition::operator+ (const wxSize & *size*) const

wxPosition& wxPosition::operator+= (const wxPosition & *pos*)

wxPosition& wxPosition::operator+= (const wxSize & *size*)

wxPosition wxPosition::operator- (const wxPosition & *pos*) const

wxPosition wxPosition::operator- (const wxSize & *size*) const

wxPosition& wxPosition::operator-= (const wxPosition & *pos*)

wxPosition& wxPosition::operator-= (const wxSize & *size*)

bool wxPosition::operator== (const wxPosition & *pos*) const

void wxPosition::SetCol (int *column*)

A synonym for SetColumn().

void wxPosition::SetColumn (int *column*)

Set a new column value.

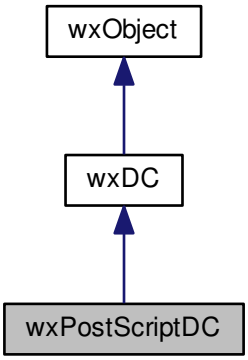
void wxPosition::SetRow (int *row*)

Set a new row value.

21.539 wxPostScriptDC Class Reference

`#include <wx/dcps.h>`

Inheritance diagram for wxPostScriptDC:



21.539.1 Detailed Description

This defines the wxWidgets Encapsulated PostScript device context, which can write PostScript files on any platform.

See wxDC for descriptions of the member functions.

Library: wxBase

Category: Device Contexts

Public Member Functions

- wxPostScriptDC ()
- wxPostScriptDC (const wxPrintData &printData)

  Constructs a PostScript printer device context from a wxPrintData object.

Additional Inherited Members

21.539.2 Constructor & Destructor Documentation

wxPostScriptDC::wxPostScriptDC ( )

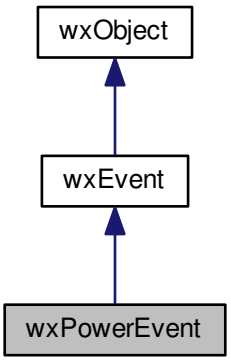
wxPostScriptDC::wxPostScriptDC (const wxPrintData & *printData*)

Constructs a PostScript printer device context from a wxPrintData object.

21.540 wxPowerEvent Class Reference

```
#include <wx/power.h>
```

Inheritance diagram for wxPowerEvent:



21.540.1 Detailed Description

The power events are generated when the system power state changes, e.g.

the system is suspended, hibernated, plugged into or unplugged from the wall socket and so on. wxPowerEvents are emitted by wxWindows.

Notice that currently only suspend and resume events are generated and only under MS Windows platform. To avoid the need to change the code using this event later when these events are implemented on the other platforms please use the test `ifdef wxHAS_POWER_EVENTS` instead of directly testing for the platform in your code: this symbol will be defined for all platforms supporting the power events.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxPowerEvent& event)

Event macros:

- EVT_POWER_SUSPENDING(func):

  Warning

  > This event and the possibility to veto suspend was removed from MSW systems starting from Windows Vista. wxPowerResourceBlocker can be used to prevent the system from suspending under both XP and later systems, use it instead of handling this event.

  System is about to be suspended, this event can be vetoed to prevent suspend from taking place.

- EVT_POWER_SUSPENDED(func): System is about to suspend: normally the application should quickly (i.e. without user intervention) close all the open files and network connections here, possibly remembering them to reopen them later when the system is resumed.
- EVT_POWER_SUSPEND_CANCEL(func): System suspension was cancelled because some application vetoed it.
- EVT_POWER_RESUME(func): System resumed from suspend: normally the application should restore the state in which it had been before the suspension.

Library: wxBase

Category: Events

See also

wxGetPowerType(), wxGetBatteryState()

Public Member Functions

- wxPowerEvent ()
- wxPowerEvent (wxEventType evtType)
- void Veto ()

  Call this to prevent suspend from taking place in `wxEVT_POWER_SUSPENDING` handler (it is ignored for all the others).
- bool IsVetoed () const

  Returns whether Veto has been called.

Additional Inherited Members

21.540.2 Constructor & Destructor Documentation

wxPowerEvent::wxPowerEvent ( )

wxPowerEvent::wxPowerEvent (wxEventType *evtType*)

21.540.3 Member Function Documentation

bool wxPowerEvent::IsVetoed ( ) const

Returns whether Veto has been called.

void wxPowerEvent::Veto ( )

Call this to prevent suspend from taking place in `wxEVT_POWER_SUSPENDING` handler (it is ignored for all the others).

21.541 wxPowerResource Class Reference

```
#include <wx/power.h>
```

21.541.1 Detailed Description

Helper functions for acquiring and releasing the given power resource.

If an application performs a long running task without user interaction it is often necessary to prevent the system from automatically suspending or powering off the screen and Acquire() method can be used to do this.

Notice that currently this functionality is only implemented for MSW and OSX and on the latter only wxPOWER_↩ RESOURCE_SYSTEM is supported for versions earlier than 10.9.

If possible, use wxPowerResourceBlocker class to ensure that Release() is called instead of calling it manually.

Since

3.1.0

Library: wxBase

Category: Miscellaneous

See also

wxPowerResourceBlocker

Static Public Member Functions

- static bool Acquire (wxPowerResourceKind kind, const wxString &reason=wxString())

  Acquire a power resource for the application.
- static void Release (wxPowerResourceKind kind)

  Release a previously acquired power resource.

21.541.2 Member Function Documentation

static bool wxPowerResource::Acquire (wxPowerResourceKind *kind,* const wxString & *reason* = wxString ()) `[static]`

Acquire a power resource for the application.

If successful, the system will not automatically power of the screen or suspend until Release() is called.

Every call to Acquire **must** be matched by a corresponding call to Release() or the system will not suspend until the application ends, use wxPowerResourceBlocker to ensure that this happens.

Parameters

kind	Power resource required, either wxPOWER_RESOURCE_SCREEN or wxPOWER_RESOURCE_SYSTEM.
reason	Optional reason may be specified which might be used on some platforms to inform the user what is preventing power saving. It should usually describe the operation requiring the resource and specifying it is strongly recommended.

Returns

Returns true if the acquisition was successful.

See also

Release()

static void wxPowerResource::Release (wxPowerResourceKind *kind*) `[static]`

Release a previously acquired power resource.

Release **must** be called for every Acquire() call made to restore normal power saving behaviour

Parameters

kind	Power resource to be released.

See also

Acquire()

21.542 wxPowerResourceBlocker Class Reference

`#include <wx/power.h>`

21.542.1 Detailed Description

Helper RAII class ensuring that power resources are released.

A wxPowerResourceBlocker object acquires a power resource in the constructor and releases it in the destructor making it impossible to to forget to release the power resource (which would prevent suspending or screen power off until the application ends).

Example:

```
void MyWindow::DoSomething()
{
    wxPowerResourceBlocker
        blocker(wxPOWER_RESOURCE_SYSTEM, "Downloading something important");

    if ( !blocker.IsInEffect() )
    {
        // If the resource could not be acquired, tell the user that he has
        // to keep the system alive
        wxLogMessage("Warning: system may suspend while downloading.");
    }

    // Run an important download and the system will not suspend while downloading
    for ( int i = 0; i < download.size(); ++i )
        download.readByte();

    // wxPOWER_RESOURCE_SYSTEM automatically released here.
}
```

Since

3.1.0

Library: wxBase

Category: Miscellaneous

See also

wxPowerResource

Public Member Functions

- wxPowerResourceBlocker (wxPowerResourceKind kind, const wxString &reason=wxString())

  Acquires the power resource.
- bool IsInEffect () const

  Returns whether the power resource could be acquired.
- ~wxPowerResourceBlocker ()

  Releases the power resource.

21.542.2 Constructor & Destructor Documentation

wxPowerResourceBlocker::wxPowerResourceBlocker (wxPowerResourceKind *kind,* const wxString & *reason =* wxString ()) [explicit]

Acquires the power resource.

Uses the same parameters as wxPowerResource::Acquire().

wxPowerResourceBlocker::~wxPowerResourceBlocker ( )

Releases the power resource.

See also

wxPowerResource::Release()

21.542.3 Member Function Documentation

bool wxPowerResourceBlocker::IsInEffect ( ) const

Returns whether the power resource could be acquired.

This can be used to inform the user that the application will not prevent automatic suspending.

See also

wxPowerResource::Acquire()

21.543 wxPreferencesEditor Class Reference

```
#include <wx/preferences.h>
```

21.543.1 Detailed Description

Manage preferences dialog.

This class encapsulates the differences – both in appearance and behaviour – between preferences dialogs on different platforms. In particular, OS X preferences look very different from the typical notebook control used on other platforms, and both OS X and GTK+ preferences windows are modeless unlike Windows options dialogs that are typically modal.

wxPreferencesEditor is able to hide the differences by hiding the creation of preferences window from the API. Instead, you create an instance of wxPreferencesEditor and add page descriptions in the form of wxPreferences↩ Page using its AddPage() method. After setting up the editor object, you must call Show() to present preferences to the user.

Note

Notice that this class is not derived from wxWindow and hence doesn't represent a window, even if its Show() method does create one internally.

Library: wxCore

Since

2.9.5

Public Member Functions

- wxPreferencesEditor (const wxString &title=wxString())

  Constructor.
- ~wxPreferencesEditor ()

  Destructor.
- void AddPage (wxPreferencesPage *page)

  Add a new page to the editor.
- virtual void Show (wxWindow *parent)

  Show the preferences dialog or bring it to the top if it's already shown.
- void Dismiss ()

  Hide the currently shown dialog, if any.

Static Public Member Functions

- static bool ShouldApplyChangesImmediately () static bool ShownModally()

  Returns whether changes to values in preferences pages should be applied immediately or only when the user clicks the OK button.

21.543.2 Constructor & Destructor Documentation

wxPreferencesEditor::wxPreferencesEditor (const wxString & *title* = wxString ( ))

Constructor.

Creates an empty editor, use AddPage() to add controls to it.

Parameters

title	The title overriding the default title of the top level window used by the editor. It is recommended to not specify this parameter to use the native convention for the preferences dialogs instead.

wxPreferencesEditor::~wxPreferencesEditor ( )

Destructor.

Destroying this object hides the associated preferences window if it is open at the moment.

The destructor is non-virtual as this class is not supposed to be derived from.

21.543.3 Member Function Documentation

void wxPreferencesEditor::AddPage (wxPreferencesPage * *page*)

Add a new page to the editor.

The editor takes ownership of the page and will delete it from its destructor (but not sooner).

See also

wxPreferencesPage, wxStockPreferencesPage

void wxPreferencesEditor::Dismiss ( )

Hide the currently shown dialog, if any.

This is typically called to dismiss the dialog if the object whose preferences it is editing was closed.

static bool wxPreferencesEditor::ShouldApplyChangesImmediately ( ) [static]

Returns whether changes to values in preferences pages should be applied immediately or only when the user clicks the OK button.

Currently, changes are applied immediately on OS X and GTK+.

The preprocessor macro `wxHAS_PREF_EDITOR_APPLY_IMMEDIATELY` is defined in this case as well. Returns whether the preferences dialog is shown modally.

If this method returns false, as it currently does in wxGTK and wxOSX, Show() simply makes the dialog visible and returns immediately. If it returns true, as it does in wxMSW and under the other platforms, then the dialog is shown modally, i.e. Show() blocks until the user dismisses it.

Notice that it isn't necessary to test the return value of this method to use this class normally, its interface is designed to work in both cases. However it can sometimes be necessary to call it if the program needs to handle modal dialogs specially, e.g. perhaps to block some periodic background update operation while a modal dialog is shown.

virtual void wxPreferencesEditor::Show (wxWindow * *parent*) [virtual]

Show the preferences dialog or bring it to the top if it's already shown.

Notice that this method may or may not block depending on the platform, i.e. depending on whether the dialog is modal or not.

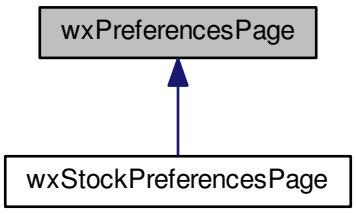
Parameters

parent	The window that invokes the preferences. Call Dismiss() before it's destroyed.

21.544 wxPreferencesPage Class Reference

`#include <wx/preferences.h>`

Inheritance diagram for wxPreferencesPage:



21.544.1 Detailed Description

One page of preferences dialog.

This is the base class for implementation of application's preferences. Its methods return various properties of the page, such as title or icon. The actual page is created by CreateWindow().

See also

wxStockPreferencesPage

Library: wxCore

Since

2.9.5

Public Member Functions

- wxPreferencesPage ()
  Constructor.
- virtual ~wxPreferencesPage ()
  Destructor.
- virtual wxString GetName () const =0
  Return name of the page.
- virtual wxBitmap GetLargeIcon () const =0
  Return 32x32 icon used for the page on some platforms.
- virtual wxWindow * CreateWindow (wxWindow *parent)=0
  Create a window for this page.

21.544.2 Constructor & Destructor Documentation

wxPreferencesPage::wxPreferencesPage ( )

Constructor.

virtual wxPreferencesPage::~wxPreferencesPage ( ) [virtual]

Destructor.

21.544.3 Member Function Documentation

virtual wxWindow* wxPreferencesPage::CreateWindow (wxWindow * *parent*) [pure virtual]

Create a window for this page.

The window will be placed into the preferences dialog in platform-specific manner. Depending on the platform, this method may be called before showing the preferences window, when switching to its tab or even more than once. Don't make assumptions about the number of times or the specific time when it is called.

The caller takes ownership of the window.

wxPanel is usually used, but doesn't have to be.

Parameters

parent	Parent window to use.

virtual wxBitmap wxPreferencesPage::GetLargeIcon ( ) const `[pure virtual]`

Return 32x32 icon used for the page on some platforms.

Currently only used on OS X.

Note

This method is only pure virtual on platforms that require it (OS X). On other platforms, it has default implementation that returns an invalid bitmap. The preprocessor symbol `wxHAS_PREF_EDITOR_ICONS` is defined if this method must be implemented.

Implemented in wxStockPreferencesPage.

virtual wxString wxPreferencesPage::GetName ( ) const `[pure virtual]`

Return name of the page.

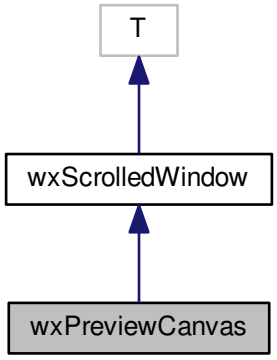
The name is used for notebook tab's label, icon label etc., depending on the platform.

Implemented in wxStockPreferencesPage.

21.545 wxPreviewCanvas Class Reference

`#include <wx/print.h>`

Inheritance diagram for wxPreviewCanvas:



21.545.1 Detailed Description

A preview canvas is the default canvas used by the print preview system to display the preview.

Library: wxCore

Category: Printing Framework

See also

wxPreviewFrame, wxPreviewControlBar, wxPrintPreview

Public Member Functions

- wxPreviewCanvas (wxPrintPreview *preview, wxWindow *parent, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name="canvas")

  Constructor.

- virtual ~wxPreviewCanvas ()

  Destructor.

- void OnPaint (wxPaintEvent &event)

  Calls wxPrintPreview::PaintPage() to refresh the canvas.

Additional Inherited Members

21.545.2 Constructor & Destructor Documentation

wxPreviewCanvas::wxPreviewCanvas (wxPrintPreview * *preview*, wxWindow * *parent*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = "canvas")

Constructor.

virtual wxPreviewCanvas::~wxPreviewCanvas ( ) [virtual]

Destructor.

21.545.3 Member Function Documentation

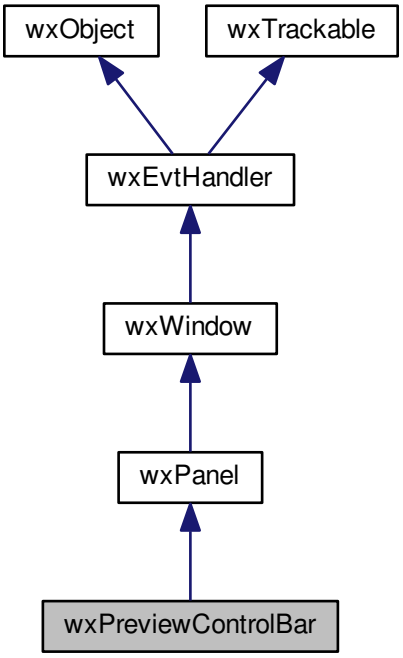
void wxPreviewCanvas::OnPaint (wxPaintEvent & *event*)

Calls wxPrintPreview::PaintPage() to refresh the canvas.

21.546 wxPreviewControlBar Class Reference

```
#include <wx/print.h>
```

Inheritance diagram for wxPreviewControlBar:



21.546.1 Detailed Description

This is the default implementation of the preview control bar, a panel with buttons and a zoom control.

You can derive a new class from this and override some or all member functions to change the behaviour and appearance; or you can leave it as it is.

Library: wxCore

Category: Printing Framework

See also

wxPreviewFrame, wxPreviewCanvas, wxPrintPreview

Public Member Functions

- wxPreviewControlBar (wxPrintPreview *preview, long buttons, wxWindow *parent, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name="panel")

  Constructor.
- virtual ~wxPreviewControlBar ()

  Destructor.
- virtual void CreateButtons ()

Creates buttons, according to value of the button style flags.

- virtual wxPrintPreviewBase * GetPrintPreview () const

  Gets the print preview object associated with the control bar.

- virtual int GetZoomControl ()

  Gets the current zoom setting in percent.

- virtual void SetZoomControl (int percent)

  Sets the zoom control.

Additional Inherited Members

21.546.2 Constructor & Destructor Documentation

wxPreviewControlBar::wxPreviewControlBar (wxPrintPreview * *preview,* long *buttons,* wxWindow * *parent,* const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `"panel"`)

Constructor.

The *buttons* parameter may be a combination of the following, using the bitwise 'or' operator:

- wxPREVIEW_PRINT: Create a print button.
- wxPREVIEW_NEXT: Create a next page button.
- wxPREVIEW_PREVIOUS: Create a previous page button.
- wxPREVIEW_ZOOM: Create a zoom control.
- wxPREVIEW_DEFAULT: Equivalent to a combination of `wxPREVIEW_PREVIOUS`, `wxPREVIEW_NEXT` and `wxPREVIEW_ZOOM`.

virtual wxPreviewControlBar::~wxPreviewControlBar ( ) `[virtual]`

Destructor.

21.546.3 Member Function Documentation

virtual void wxPreviewControlBar::CreateButtons ( ) `[virtual]`

Creates buttons, according to value of the button style flags.

Todo which flags??

virtual wxPrintPreviewBase* wxPreviewControlBar::GetPrintPreview ( ) const `[virtual]`

Gets the print preview object associated with the control bar.

virtual int wxPreviewControlBar::GetZoomControl ( ) `[virtual]`

Gets the current zoom setting in percent.

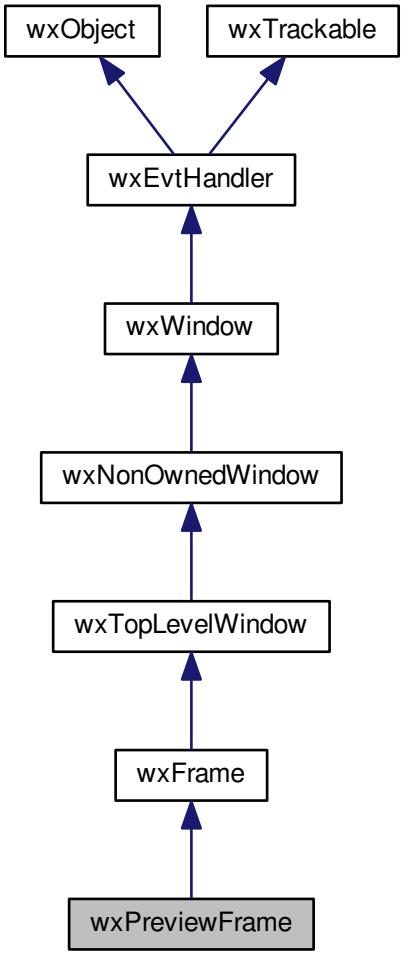
virtual void wxPreviewControlBar::SetZoomControl (int *percent*) `[virtual]`

Sets the zoom control.

21.547 wxPreviewFrame Class Reference

`#include <wx/print.h>`

Inheritance diagram for wxPreviewFrame:



21.547.1 Detailed Description

This class provides the default method of managing the print preview interface.

Member functions may be overridden to replace functionality, or the class may be used without derivation.

Library: wxCore

Category: Printing Framework

See also

wxPreviewCanvas, wxPreviewControlBar, wxPrintPreview

Public Member Functions

- wxPreviewFrame (wxPrintPreviewBase *preview, wxWindow *parent, const wxString &title="Print Preview", const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor.
- virtual ~wxPreviewFrame ()

  Destructor.
- virtual void CreateCanvas ()

  Creates a wxPreviewCanvas.
- virtual void CreateControlBar ()

  Creates a wxPreviewControlBar.
- virtual void Initialize ()

  Initializes the frame elements and prepares for showing it.
- virtual void InitializeWithModality (wxPreviewFrameModalityKind kind)

  Initializes the frame elements and prepares for showing it with the given modality kind.
- void OnCloseWindow (wxCloseEvent &event)

  Enables any disabled frames in the application, and deletes the print preview object, implicitly deleting any printout objects associated with the print preview object.

Additional Inherited Members

21.547.2 Constructor & Destructor Documentation

wxPreviewFrame::wxPreviewFrame (wxPrintPreviewBase * *preview*, wxWindow * *parent*, const wxString & *title* = `"Print Preview"`, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor.

Pass a print preview object plus other normal frame arguments. The print preview object will be destroyed by the frame when it closes.

virtual wxPreviewFrame::~wxPreviewFrame ( ) `[virtual]`

Destructor.

21.547.3 Member Function Documentation

virtual void wxPreviewFrame::CreateCanvas ( ) `[virtual]`

Creates a wxPreviewCanvas.

Override this function to allow a user-defined preview canvas object to be created.

virtual void wxPreviewFrame::CreateControlBar ( ) [virtual]

Creates a wxPreviewControlBar.

Override this function to allow a user-defined preview control bar object to be created.

virtual void wxPreviewFrame::Initialize ( ) [virtual]

Initializes the frame elements and prepares for showing it.

Calling this method is equivalent to calling InitializeWithModality() with wxPreviewFrame_AppModal argument, please see its documentation for more details.

Please notice that this function is virtual mostly for backwards compatibility only, there is no real need to override it as it's never called by wxWidgets itself.

virtual void wxPreviewFrame::InitializeWithModality (wxPreviewFrameModalityKind *kind*) [virtual]

Initializes the frame elements and prepares for showing it with the given modality kind.

This method creates the frame elements by calling CreateCanvas() and CreateControlBar() methods (which may be overridden to customize them) and prepares to show the frame according to the value of *kind* parameter:

- If it is wxPreviewFrame_AppModal, all the other application windows will be disabled when this frame is shown. This is the same behaviour as that of simple Initialize().
- If it is wxPreviewFrame_WindowModal, only the parent window of the preview frame will be disabled when it is shown.
- And if it is wxPreviewFrame_NonModal, no windows at all will be disabled while the preview is shown.

Notice that this function (or Initialize()) must be called by the application prior to showing the frame but you still must call `Show(true)` to actually show it afterwards.

Parameters

kind	The modality kind of preview frame.

Since

2.9.2

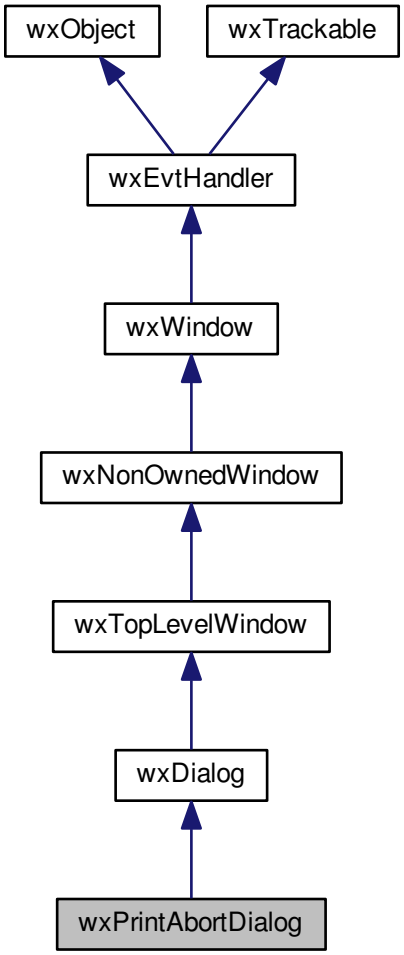
void wxPreviewFrame::OnCloseWindow (wxCloseEvent & *event*)

Enables any disabled frames in the application, and deletes the print preview object, implicitly deleting any printout objects associated with the print preview object.

21.548 wxPrintAbortDialog Class Reference

`#include <wx/print.h>`

Inheritance diagram for wxPrintAbortDialog:



21.548.1 Detailed Description

The dialog created by default by the print framework that enables aborting the printing process.

Public Member Functions

- wxPrintAbortDialog (wxWindow *parent, const wxString &documentTitle, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE, const wxString &name="dialog")
- void SetProgress (int currentPage, int totalPages, int currentCopy, int totalCopies)

Additional Inherited Members

21.548.2 Constructor & Destructor Documentation

wxPrintAbortDialog::wxPrintAbortDialog (wxWindow * *parent*, const wxString & *documentTitle*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE, const wxString & *name* = "dialog")

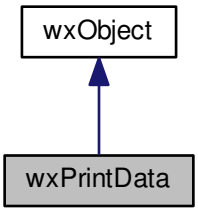
21.548.3 Member Function Documentation

void wxPrintAbortDialog::SetProgress (int *currentPage*, int *totalPages*, int *currentCopy*, int *totalCopies*)

21.549 wxPrintData Class Reference

`#include <wx/cmndata.h>`

Inheritance diagram for wxPrintData:



21.549.1 Detailed Description

This class holds a variety of information related to printers and printer device contexts.

This class is used to create a wxPrinterDC and a wxPostScriptDC. It is also used as a data member of wxPrintDialogData and wxPageSetupDialogData, as part of the mechanism for transferring data between the print dialogs and the application.

Library: wxCore

Category: Printing Framework, Data Structures

See also

Printing Framework Overview, wxPrintDialog, wxPageSetupDialog, wxPrintDialogData, wxPageSetupDialogData, wxPrintDialog Overview, wxPrinterDC, wxPostScriptDC

Public Member Functions

- wxPrintData ()

  Default constructor.
- wxPrintData (const wxPrintData &data)

  Copy constructor.
- virtual ~wxPrintData ()

  Destructor.

- wxPrintBin GetBin () const

  Returns the current bin (papersource).
- bool GetCollate () const

  Returns true if collation is on.
- bool GetColour () const

  Returns true if colour printing is on.
- wxDuplexMode GetDuplex () const

  Returns the duplex mode.
- int GetNoCopies () const

  Returns the number of copies requested by the user.
- wxPrintOrientation GetOrientation () const

  Gets the orientation.
- wxPaperSize GetPaperId () const

  Returns the paper size id.
- const wxString & GetPrinterName () const

  Returns the printer name.
- wxPrintQuality GetQuality () const

  Returns the current print quality.
- bool IsOk () const

  Returns true if the print data is valid for using in print dialogs.
- void SetBin (wxPrintBin flag)

  Sets the current bin.
- void SetCollate (bool flag)

  Sets collation to on or off.
- void SetColour (bool flag)

  Sets colour printing on or off.
- void SetDuplex (wxDuplexMode mode)

  Returns the duplex mode.
- void SetNoCopies (int n)

  Sets the default number of copies to be printed out.
- void SetOrientation (wxPrintOrientation orientation)

  Sets the orientation.
- void SetPaperId (wxPaperSize paperId)

  Sets the paper id.
- void SetPrinterName (const wxString &printerName)

  Sets the printer name.
- void SetQuality (wxPrintQuality quality)

  Sets the desired print quality.
- wxPrintData & operator= (const wxPrintData &data)

  Assigns print data to this object.
- wxString GetFilename () const
- void SetFilename (const wxString &filename)
- wxPrintMode GetPrintMode () const
- void SetPrintMode (wxPrintMode printMode)

Additional Inherited Members

21.549.2 Constructor & Destructor Documentation

wxPrintData::wxPrintData ( )

Default constructor.

wxPrintData::wxPrintData (const wxPrintData & *data*)

Copy constructor.

virtual wxPrintData::~wxPrintData ( ) [virtual]

Destructor.

21.549.3 Member Function Documentation

wxPrintBin wxPrintData::GetBin ( ) const

Returns the current bin (papersource).

By default, the system is left to select the bin (`wxPRINTBIN_DEFAULT` is returned).

See SetBin() for the full list of bin values.

bool wxPrintData::GetCollate ( ) const

Returns true if collation is on.

bool wxPrintData::GetColour ( ) const

Returns true if colour printing is on.

wxDuplexMode wxPrintData::GetDuplex ( ) const

Returns the duplex mode.

One of wxDUPLEX_SIMPLEX, wxDUPLEX_HORIZONTAL, wxDUPLEX_VERTICAL.

wxString wxPrintData::GetFilename ( ) const

int wxPrintData::GetNoCopies ( ) const

Returns the number of copies requested by the user.

wxPrintOrientation wxPrintData::GetOrientation ( ) const

Gets the orientation.

This can be wxLANDSCAPE or wxPORTRAIT.

wxPaperSize wxPrintData::GetPaperId ( ) const

Returns the paper size id.

See also

SetPaperId()

const wxString& wxPrintData::GetPrinterName ( ) const

Returns the printer name.

If the printer name is the empty string, it indicates that the default printer should be used.

wxPrintMode wxPrintData::GetPrintMode ( ) const

wxPrintQuality wxPrintData::GetQuality ( ) const

Returns the current print quality.

This can be a positive integer, denoting the number of dots per inch, or one of the following identifiers:

- wxPRINT_QUALITY_HIGH
- wxPRINT_QUALITY_MEDIUM
- wxPRINT_QUALITY_LOW
- wxPRINT_QUALITY_DRAFT

On input you should pass one of these identifiers, but on return you may get back a positive integer indicating the current resolution setting.

bool wxPrintData::IsOk ( ) const

Returns true if the print data is valid for using in print dialogs.

This can return false on Windows if the current printer is not set, for example. On all other platforms, it returns true.

wxPrintData& wxPrintData::operator= (const wxPrintData & *data*)

Assigns print data to this object.

void wxPrintData::SetBin (wxPrintBin *flag*)

Sets the current bin.

void wxPrintData::SetCollate (bool *flag*)

Sets collation to on or off.

void wxPrintData::SetColour (bool *flag*)

Sets colour printing on or off.

void wxPrintData::SetDuplex (wxDuplexMode *mode*)

Returns the duplex mode.

One of wxDUPLEX_SIMPLEX, wxDUPLEX_HORIZONTAL, wxDUPLEX_VERTICAL.

void wxPrintData::SetFilename (const wxString & *filename*)

void wxPrintData::SetNoCopies (int *n*)

Sets the default number of copies to be printed out.

void wxPrintData::SetOrientation (wxPrintOrientation *orientation*)

Sets the orientation.

This can be wxLANDSCAPE or wxPORTRAIT.

void wxPrintData::SetPaperId (wxPaperSize *paperId*)

Sets the paper id.

This indicates the type of paper to be used. For a mapping between paper id, paper size and string name, see wxPrintPaperDatabase in "paper.h" (not yet documented).

void wxPrintData::SetPrinterName (const wxString & *printerName*)

Sets the printer name.

This can be the empty string to indicate that the default printer should be used.

void wxPrintData::SetPrintMode (wxPrintMode *printMode*)

void wxPrintData::SetQuality (wxPrintQuality *quality*)

Sets the desired print quality.

This can be a positive integer, denoting the number of dots per inch, or one of the following identifiers:

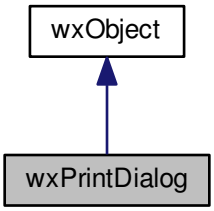
- wxPRINT_QUALITY_HIGH
- wxPRINT_QUALITY_MEDIUM
- wxPRINT_QUALITY_LOW
- wxPRINT_QUALITY_DRAFT

On input you should pass one of these identifiers, but on return you may get back a positive integer indicating the current resolution setting.

21.550 wxPrintDialog Class Reference

#include <wx/printdlg.h>

Inheritance diagram for wxPrintDialog:



21.550.1 Detailed Description

This class represents the print and print setup common dialogs.

You may obtain a wxPrinterDC device context from a successfully dismissed print dialog.

Library: wxCore

Category: Printing Framework

See also

Printing Framework Overview, wxPrintDialog Overview

Public Member Functions

- wxPrintDialog (wxWindow *parent, wxPrintDialogData *data=NULL)

  Constructor.
- wxPrintDialog (wxWindow *parent, wxPrintData *data)
- virtual ~wxPrintDialog ()

  Destructor.
- virtual wxDC * GetPrintDC ()

  Returns the device context created by the print dialog, if any.
- virtual wxPrintDialogData & GetPrintDialogData ()

  Returns the print dialog data associated with the print dialog.
- virtual wxPrintData & GetPrintData ()

  Returns the print data associated with the print dialog.
- virtual int ShowModal ()

  Shows the dialog, returning `wxID_OK` if the user pressed OK, and `wxID_CANCEL` otherwise.

Additional Inherited Members

21.550.2 Constructor & Destructor Documentation

wxPrintDialog::wxPrintDialog (wxWindow * *parent,* wxPrintDialogData * *data* = NULL)

Constructor.

Pass a parent window, and optionally a pointer to a block of print data, which will be copied to the print dialog's print data.

See also

wxPrintDialogData

wxPrintDialog::wxPrintDialog (wxWindow * *parent,* wxPrintData * *data*)

virtual wxPrintDialog::~wxPrintDialog ( ) [virtual]

Destructor.

If GetPrintDC() has not been called, the device context obtained by the dialog (if any) will be deleted.

21.550.3 Member Function Documentation

virtual wxPrintData& wxPrintDialog::GetPrintData ( ) [virtual]

Returns the print data associated with the print dialog.

virtual wxDC* wxPrintDialog::GetPrintDC ( ) [virtual]

Returns the device context created by the print dialog, if any.

When this function has been called, the ownership of the device context is transferred to the application, so it must then be deleted explicitly.

virtual wxPrintDialogData& wxPrintDialog::GetPrintDialogData ( ) [virtual]

Returns the print dialog data associated with the print dialog.

virtual int wxPrintDialog::ShowModal ( ) [virtual]

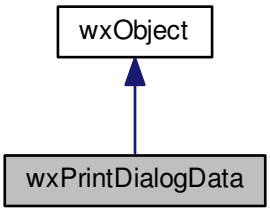
Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

After this function is called, a device context may be retrievable using GetPrintDC().

21.551 wxPrintDialogData Class Reference

```
#include <wx/cmndata.h>
```

Inheritance diagram for wxPrintDialogData:



21.551.1 Detailed Description

This class holds information related to the visual characteristics of wxPrintDialog.

It contains a wxPrintData object with underlying printing settings.

Library: wxCore

Category: Printing Framework, Common Dialogs, Data Structures

See also

Printing Framework Overview, wxPrintDialog, wxPrintDialog Overview

Public Member Functions

- wxPrintDialogData ()

  Default constructor.
- wxPrintDialogData (const wxPrintDialogData &dialogData)

  Copy constructor.
- wxPrintDialogData (const wxPrintData &printData)

  Construct an object from a print dialog data object.
- virtual ~wxPrintDialogData ()

  Destructor.
- void EnableHelp (bool flag)

  Enables or disables the "Help" button.
- void EnablePageNumbers (bool flag)

  Enables or disables the "Page numbers" controls.
- void EnablePrintToFile (bool flag)

  Enables or disables the "Print to file" checkbox.
- void EnableSelection (bool flag)

  Enables or disables the "Selection" radio button.
- bool GetAllPages () const

  Returns true if the user requested that all pages be printed.
- bool GetCollate () const

Returns true if the user requested that the document(s) be collated.

- int GetFromPage () const

  Returns the from page number, as entered by the user.
- int GetMaxPage () const

  Returns the maximum page number.
- int GetMinPage () const

  Returns the minimum page number.
- int GetNoCopies () const

  Returns the number of copies requested by the user.
- wxPrintData & GetPrintData ()

  Returns a reference to the internal wxPrintData object.
- bool GetPrintToFile () const

  Returns true if the user has selected printing to a file.
- bool GetSelection () const

  Returns true if the user requested that the selection be printed (where "selection" is a concept specific to the application).
- int GetToPage () const

  Returns the "print to" page number, as entered by the user.
- bool IsOk () const

  Returns true if the print data is valid for using in print dialogs.
- void SetCollate (bool flag)

  Sets the "Collate" checkbox to true or false.
- void SetFromPage (int page)

  Sets the from page number.
- void SetMaxPage (int page)

  Sets the maximum page number.
- void SetMinPage (int page)

  Sets the minimum page number.
- void SetNoCopies (int n)

  Sets the default number of copies the user has requested to be printed out.
- void SetPrintData (const wxPrintData &printData)

  Sets the internal wxPrintData.
- void SetPrintToFile (bool flag)

  Sets the "Print to file" checkbox to true or false.
- void SetSelection (bool flag)

  Selects the "Selection" radio button.
- void SetSetupDialog (bool flag)
- void SetToPage (int page)

  Sets the "print to" page number.
- void operator= (const wxPrintData &data)

  Assigns print data to this object.
- void operator= (const wxPrintDialogData &data)

  Assigns another print dialog data object to this object.

Additional Inherited Members

21.551.2 Constructor & Destructor Documentation

wxPrintDialogData::wxPrintDialogData ( )

Default constructor.

wxPrintDialogData::wxPrintDialogData (const wxPrintDialogData & *dialogData*)

Copy constructor.

wxPrintDialogData::wxPrintDialogData (const wxPrintData & *printData*)

Construct an object from a print dialog data object.

virtual wxPrintDialogData::~wxPrintDialogData ( ) [virtual]

Destructor.

21.551.3 Member Function Documentation

void wxPrintDialogData::EnableHelp (bool *flag*)

Enables or disables the "Help" button.

void wxPrintDialogData::EnablePageNumbers (bool *flag*)

Enables or disables the "Page numbers" controls.

void wxPrintDialogData::EnablePrintToFile (bool *flag*)

Enables or disables the "Print to file" checkbox.

void wxPrintDialogData::EnableSelection (bool *flag*)

Enables or disables the "Selection" radio button.

bool wxPrintDialogData::GetAllPages ( ) const

Returns true if the user requested that all pages be printed.

bool wxPrintDialogData::GetCollate ( ) const

Returns true if the user requested that the document(s) be collated.

int wxPrintDialogData::GetFromPage ( ) const

Returns the *from* page number, as entered by the user.

int wxPrintDialogData::GetMaxPage ( ) const

Returns the *maximum* page number.

int wxPrintDialogData::GetMinPage ( ) const

Returns the *minimum* page number.

int wxPrintDialogData::GetNoCopies ( ) const

Returns the number of copies requested by the user.

wxPrintData& wxPrintDialogData::GetPrintData ( )

Returns a reference to the internal wxPrintData object.

bool wxPrintDialogData::GetPrintToFile ( ) const

Returns true if the user has selected printing to a file.

bool wxPrintDialogData::GetSelection ( ) const

Returns true if the user requested that the selection be printed (where "selection" is a concept specific to the application).

int wxPrintDialogData::GetToPage ( ) const

Returns the *"print to"* page number, as entered by the user.

bool wxPrintDialogData::IsOk ( ) const

Returns true if the print data is valid for using in print dialogs.

This can return false on Windows if the current printer is not set, for example. On all other platforms, it returns true.

void wxPrintDialogData::operator= (const wxPrintData & *data*)

Assigns print data to this object.

void wxPrintDialogData::operator= (const wxPrintDialogData & *data*)

Assigns another print dialog data object to this object.

void wxPrintDialogData::SetCollate (bool *flag*)

Sets the "Collate" checkbox to true or false.

void wxPrintDialogData::SetFromPage (int *page*)

Sets the *from* page number.

void wxPrintDialogData::SetMaxPage (int *page*)

Sets the *maximum* page number.

void wxPrintDialogData::SetMinPage (int *page*)

Sets the *minimum* page number.

void wxPrintDialogData::SetNoCopies (int *n*)

Sets the default number of copies the user has requested to be printed out.

void wxPrintDialogData::SetPrintData (const wxPrintData & *printData*)

Sets the internal wxPrintData.

void wxPrintDialogData::SetPrintToFile (bool *flag*)

Sets the "Print to file" checkbox to true or false.

void wxPrintDialogData::SetSelection (bool *flag*)

Selects the "Selection" radio button.

The effect of printing the selection depends on how the application implements this command, if at all.

void wxPrintDialogData::SetSetupDialog (bool *flag*)

Deprecated This function has been deprecated since version 2.5.4.

Determines whether the dialog to be shown will be the Print dialog (pass false) or Print Setup dialog (pass true).

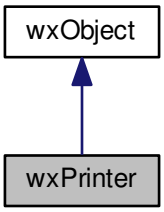
void wxPrintDialogData::SetToPage (int *page*)

Sets the *"print to"* page number.

21.552 wxPrinter Class Reference

`#include <wx/print.h>`

Inheritance diagram for wxPrinter:



21.552.1 Detailed Description

This class represents the Windows or PostScript printer, and is the vehicle through which printing may be launched by an application.

Printing can also be achieved through using of lower functions and classes, but this and associated classes provide a more convenient and general method of printing.

Library: wxCore

Category: Printing Framework

See also

Printing Framework Overview, wxPrinterDC, wxPrintDialog, wxPrintout, wxPrintPreview

Public Member Functions

- wxPrinter (wxPrintDialogData *data=NULL)

  Constructor.
- virtual wxPrintAbortDialog * CreateAbortWindow (wxWindow *parent, wxPrintout *printout)

  Creates the default printing abort window, with a cancel button.
- bool GetAbort () const

  Returns true if the user has aborted the print job.
- virtual wxPrintDialogData & GetPrintDialogData () const

  Returns the print data associated with the printer object.
- virtual bool Print (wxWindow *parent, wxPrintout *printout, bool prompt=true)

  Starts the printing process.
- virtual wxDC * PrintDialog (wxWindow *parent)

  Invokes the print dialog.
- virtual void ReportError (wxWindow *parent, wxPrintout *printout, const wxString &message)

  Default error-reporting function.
- virtual bool Setup (wxWindow *parent)

  Invokes the print setup dialog.

Static Public Member Functions

- static wxPrinterError GetLastError ()

  Return last error.

Additional Inherited Members

21.552.2 Constructor & Destructor Documentation

wxPrinter::wxPrinter (wxPrintDialogData * *data* = `NULL`)

Constructor.

Pass an optional pointer to a block of print dialog data, which will be copied to the printer object's local data.

See also

wxPrintDialogData, wxPrintData

21.552.3 Member Function Documentation

virtual wxPrintAbortDialog* wxPrinter::CreateAbortWindow (wxWindow * *parent*, wxPrintout * *printout*) [virtual]

Creates the default printing abort window, with a cancel button.

bool wxPrinter::GetAbort ( ) const

Returns true if the user has aborted the print job.

static wxPrinterError wxPrinter::GetLastError ( ) [static]

Return last error.

Valid after calling Print(), PrintDialog() or wxPrintPreview::Print().

These functions set last error to `wxPRINTER_NO_ERROR` if no error happened.

Returned value is one of the following:

wxPRINTER_NO_ERROR	No error happened.
wxPRINTER_CANCELLED	The user cancelled printing.
wxPRINTER_ERROR	There was an error during printing.

virtual wxPrintDialogData& wxPrinter::GetPrintDialogData ( ) const [virtual]

Returns the print data associated with the printer object.

virtual bool wxPrinter::Print (wxWindow * *parent*, wxPrintout * *printout*, bool *prompt* = `true`) [virtual]

Starts the printing process.

Provide a parent window, a user-defined wxPrintout object which controls the printing of a document, and whether the print dialog should be invoked first.

Print() could return false if there was a problem initializing the printer device context (current printer not set, for example) or the user cancelled printing. Call GetLastError() to get detailed information about the kind of the error.

virtual wxDC* wxPrinter::PrintDialog (wxWindow * *parent*) [virtual]

Invokes the print dialog.

If successful (the user did not press Cancel and no error occurred), a suitable device context will be returned; otherwise NULL is returned; call GetLastError() to get detailed information about the kind of the error.

Remarks

The application must delete this device context to avoid a memory leak.

virtual void wxPrinter::ReportError (wxWindow * *parent*, wxPrintout * *printout*, const wxString & *message*) [virtual]

Default error-reporting function.

virtual bool wxPrinter::Setup (wxWindow * *parent*) [virtual]

Invokes the print setup dialog.

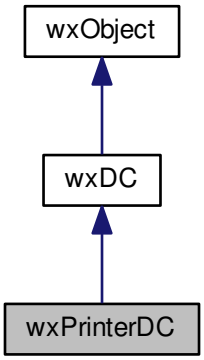
Remarks

The setup dialog is obsolete from Windows 95, though retained for backward compatibility.

21.553 wxPrinterDC Class Reference

`#include <wx/dcprint.h>`

Inheritance diagram for wxPrinterDC:



21.553.1 Detailed Description

A printer device context is specific to MSW and Mac, and allows access to any printer with a Windows or Macintosh driver.

See wxDC for further information on device contexts, and wxDC::GetSize() for advice on achieving the correct scaling for the page.

Library: wxCore

Category: Printing Framework

See also

Printing Framework Overview, wxDC

Public Member Functions

- wxPrinterDC (const wxPrintData &printData)

  Constructor.

- wxRect GetPaperRect () const

  Return the rectangle in device coordinates that corresponds to the full paper area, including the nonprinting regions of the paper.

Additional Inherited Members

21.553.2 Constructor & Destructor Documentation

wxPrinterDC::wxPrinterDC (const wxPrintData & *printData*)

Constructor.

Pass a wxPrintData object with information necessary for setting up a suitable printer device context. This is the recommended way to construct a wxPrinterDC. Make sure you specify a reference to a wxPrintData object, not a pointer - you may not even get a warning if you pass a pointer instead.

21.553.3 Member Function Documentation

wxRect wxPrinterDC::GetPaperRect ( ) const

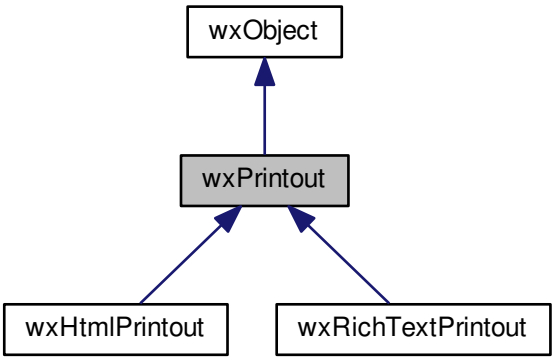
Return the rectangle in device coordinates that corresponds to the full paper area, including the nonprinting regions of the paper.

The point (0,0) in device coordinates is the top left corner of the page rectangle, which is the printable area on MSW and Mac. The coordinates of the top left corner of the paper rectangle will therefore have small negative values, while the bottom right coordinates will be somewhat larger than the values returned by wxDC::GetSize().

21.554 wxPrintout Class Reference

`#include <wx/print.h>`

Inheritance diagram for wxPrintout:



21.554.1 Detailed Description

This class encapsulates the functionality of printing out an application document.

A new class must be derived and members overridden to respond to calls such as OnPrintPage() and HasPage() and to render the print image onto an associated wxDC. Instances of this class are passed to wxPrinter::Print() or to a wxPrintPreview object to initiate printing or previewing.

Your derived wxPrintout is responsible for drawing both the preview image and the printed page. If your windows' drawing routines accept an arbitrary DC as an argument, you can re-use those routines within your wxPrintout subclass to draw the printout image. You may also add additional drawing elements within your wxPrintout subclass, like headers, footers, and/or page numbers. However, the image on the printed page will often differ from the image drawn on the screen, as will the print preview image – not just in the presence of headers and footers, but typically in scale. A high-resolution printer presents a much larger drawing surface (i.e., a higher-resolution DC); a zoomed-out preview image presents a much smaller drawing surface (lower-resolution DC). By using the routines FitThisSize↩ ToXXX() and/or MapScreenSizeToXXX() within your wxPrintout subclass to set the user scale and origin of the associated DC, you can easily use a single drawing routine to draw on your application's windows, to create the print preview image, and to create the printed paper image, and achieve a common appearance to the preview image and the printed page.

Library: wxCore

Category: Printing Framework

See also

Printing Framework Overview, wxPrinterDC, wxPrintDialog, wxPageSetupDialog, wxPrinter, wxPrintPreview

Public Member Functions

- wxPrintout (const wxString &title="Printout")

  Constructor.
- virtual ~wxPrintout ()

  Destructor.
- void FitThisSizeToPage (const wxSize &imageSize)

  Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the page rectangle and the origin is at the top left corner of the page rectangle.
- void FitThisSizeToPageMargins (const wxSize &imageSize, const wxPageSetupDialogData &pageSetup↩ Data)

  Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the page margins set in the given wxPageSetupDialogData object.
- void FitThisSizeToPaper (const wxSize &imageSize)

  Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the paper and the origin is at the top left corner of the paper.
- wxDC * GetDC () const

  Returns the device context associated with the printout (given to the printout at start of printing or previewing).
- wxRect GetLogicalPageMarginsRect (const wxPageSetupDialogData &pageSetupData) const

  Return the rectangle corresponding to the page margins specified by the given wxPageSetupDialogData object in the associated wxDC's logical coordinates for the current user scale and device origin.
- wxRect GetLogicalPageRect () const

  Return the rectangle corresponding to the page in the associated wxDC 's logical coordinates for the current user scale and device origin.
- wxRect GetLogicalPaperRect () const

  Return the rectangle corresponding to the paper in the associated wxDC 's logical coordinates for the current user scale and device origin.

- void GetPPIPrinter (int *w, int *h) const

  Returns the number of pixels per logical inch of the printer device context.
- void GetPPIScreen (int *w, int *h) const

  Returns the number of pixels per logical inch of the screen device context.
- virtual void GetPageInfo (int *minPage, int *maxPage, int *pageFrom, int *pageTo)

  Called by the framework to obtain information from the application about minimum and maximum page values that the user can select, and the required page range to be printed.
- void GetPageSizeMM (int *w, int *h) const

  Returns the size of the printer page in millimetres.
- void GetPageSizePixels (int *w, int *h) const

  Returns the size of the printer page in pixels, called the page rectangle.
- wxRect GetPaperRectPixels () const

  Returns the rectangle that corresponds to the entire paper in pixels, called the paper rectangle.
- virtual wxString GetTitle () const

  Returns the title of the printout.
- virtual bool HasPage (int pageNum)

  Should be overridden to return true if the document has this page, or false if not.
- virtual bool IsPreview () const

  Returns true if the printout is currently being used for previewing.
- wxPrintPreview * GetPreview () const

  Returns the associated preview object if any.
- void MapScreenSizeToDevice ()

  Set the user scale and device origin of the wxDC associated with this wxPrintout so that one screen pixel maps to one device pixel on the DC.
- void MapScreenSizeToPage ()

  This sets the user scale of the wxDC associated with this wxPrintout to the same scale as MapScreenSizeToPaper() but sets the logical origin to the top left corner of the page rectangle.
- void MapScreenSizeToPageMargins (const wxPageSetupDialogData &pageSetupData)

  This sets the user scale of the wxDC associated with this wxPrintout to the same scale as MapScreenSizeToPage↩ Margins() but sets the logical origin to the top left corner of the page margins specified by the given wxPageSetup↩ DialogData object.
- void MapScreenSizeToPaper ()

  Set the user scale and device origin of the wxDC associated with this wxPrintout so that the printed page matches the screen size as closely as possible and the logical origin is in the top left corner of the paper rectangle.
- void OffsetLogicalOrigin (wxCoord xoff, wxCoord yoff)

  Shift the device origin by an amount specified in logical coordinates.
- virtual bool OnBeginDocument (int startPage, int endPage)

  Called by the framework at the start of document printing.
- virtual void OnBeginPrinting ()

  Called by the framework at the start of printing.
- virtual void OnEndDocument ()

  Called by the framework at the end of document printing.
- virtual void OnEndPrinting ()

  Called by the framework at the end of printing.
- virtual void OnPreparePrinting ()

  Called once by the framework before any other demands are made of the wxPrintout object.
- virtual bool OnPrintPage (int pageNum)=0

  Called by the framework when a page should be printed.
- void SetLogicalOrigin (wxCoord x, wxCoord y)

  Set the device origin of the associated wxDC so that the current logical point becomes the new logical origin.

Additional Inherited Members

21.554.2 Constructor & Destructor Documentation

wxPrintout::wxPrintout (const wxString & *title* = "Printout")

Constructor.

Pass an optional title argument - the current filename would be a good idea. This will appear in the printing list (at least in MSW)

virtual wxPrintout::~wxPrintout ( ) [virtual]

Destructor.

21.554.3 Member Function Documentation

void wxPrintout::FitThisSizeToPage (const wxSize & *imageSize*)

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the page rectangle and the origin is at the top left corner of the page rectangle.

On MSW and Mac, the page rectangle is the printable area of the page. On other platforms and PostScript printing, the page rectangle is the entire paper.

Use this if you want your printed image as large as possible, but with the caveat that on some platforms, portions of the image might be cut off at the edges.

void wxPrintout::FitThisSizeToPageMargins (const wxSize & *imageSize,* const wxPageSetupDialogData & *pageSetupData*)

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the page margins set in the given wxPageSetupDialogData object.

This function provides the greatest consistency across all platforms because it does not depend on having access to the printable area of the paper.

Remarks

On Mac, the native wxPageSetupDialog does not let you set the page margins; you'll have to provide your own mechanism, or you can use the Mac-only class wxMacPageMarginsDialog.

void wxPrintout::FitThisSizeToPaper (const wxSize & *imageSize*)

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the given image size fits entirely within the paper and the origin is at the top left corner of the paper.

Use this if you're managing your own page margins.

Note

With most printers, the region around the edges of the paper are not printable so that the edges of the image could be cut off.

wxDC* wxPrintout::GetDC ( ) const

Returns the device context associated with the printout (given to the printout at start of printing or previewing).

The application can use GetDC() to obtain a device context to draw on.

This will be a wxPrinterDC if printing under Windows or Mac, a wxPostScriptDC if printing on other platforms, and a wxMemoryDC if previewing.

wxRect wxPrintout::GetLogicalPageMarginsRect (const wxPageSetupDialogData & *pageSetupData*) const

Return the rectangle corresponding to the page margins specified by the given wxPageSetupDialogData object in the associated wxDC's logical coordinates for the current user scale and device origin.

The page margins are specified with respect to the edges of the paper on all platforms.

wxRect wxPrintout::GetLogicalPageRect ( ) const

Return the rectangle corresponding to the page in the associated wxDC 's logical coordinates for the current user scale and device origin.

On MSW and Mac, this will be the printable area of the paper. On other platforms and PostScript printing, this will be the full paper rectangle.

wxRect wxPrintout::GetLogicalPaperRect ( ) const

Return the rectangle corresponding to the paper in the associated wxDC 's logical coordinates for the current user scale and device origin.

virtual void wxPrintout::GetPageInfo (int * *minPage,* int * *maxPage,* int * *pageFrom,* int * *pageTo*) [virtual]

Called by the framework to obtain information from the application about minimum and maximum page values that the user can select, and the required page range to be printed.

By default this returns (1, 32000) for the page minimum and maximum values, and (1, 1) for the required page range.

minPage must be greater than zero and *maxPage* must be greater than *minPage*.

Reimplemented in wxRichTextPrintout.

void wxPrintout::GetPageSizeMM (int * *w,* int * *h*) const

Returns the size of the printer page in millimetres.

wxPerl Note: In wxPerl this method takes no arguments and returns a 2-element list (w, h).

void wxPrintout::GetPageSizePixels (int * *w,* int * *h*) const

Returns the size of the printer page in pixels, called the page rectangle.

The page rectangle has a top left corner at (0,0) and a bottom right corner at (w,h). These values may not be the same as the values returned from wxDC::GetSize(); if the printout is being used for previewing, a memory device context is used, which uses a bitmap size reflecting the current preview zoom. The application must take this discrepancy into account if previewing is to be supported.

wxRect wxPrintout::GetPaperRectPixels ( ) const

Returns the rectangle that corresponds to the entire paper in pixels, called the paper rectangle.

This distinction between paper rectangle and page rectangle reflects the fact that most printers cannot print all the way to the edge of the paper. The page rectangle is a rectangle whose top left corner is at (0,0) and whose width and height are given by wxDC::GetPageSizePixels().

On MSW and Mac, the page rectangle gives the printable area of the paper, while the paper rectangle represents the entire paper, including non-printable borders. Thus, the rectangle returned by wxDC::GetPaperRectPixels() will have a top left corner whose coordinates are small negative numbers and the bottom right corner will have values somewhat larger than the width and height given by wxDC::GetPageSizePixels().

On other platforms and for PostScript printing, the paper is treated as if its entire area were printable, so this function will return the same rectangle as the page rectangle.

void wxPrintout::GetPPIPrinter (int * *w*, int * *h*) const

Returns the number of pixels per logical inch of the printer device context.

Dividing the printer PPI by the screen PPI can give a suitable scaling factor for drawing text onto the printer.

Remember to multiply this by a scaling factor to take the preview DC size into account. Or you can just use the FitThisSizeToXXX() and MapScreenSizeToXXX routines below, which do most of the scaling calculations for you.

wxPerl Note: In wxPerl this method takes no arguments and returns a 2-element list (w, h).

void wxPrintout::GetPPIScreen (int * *w*, int * *h*) const

Returns the number of pixels per logical inch of the screen device context.

Dividing the printer PPI by the screen PPI can give a suitable scaling factor for drawing text onto the printer.

If you are doing your own scaling, remember to multiply this by a scaling factor to take the preview DC size into account.

wxPerl Note: In wxPerl this method takes no arguments and returns a 2-element list (w, h).

wxPrintPreview* wxPrintout::GetPreview ( ) const

Returns the associated preview object if any.

If this printout object is used for previewing, returns the associated wxPrintPreview. Otherwise returns NULL.

The returned pointer is not owned by the printout and must not be deleted.

See also

IsPreview()

Since

2.9.1.

virtual wxString wxPrintout::GetTitle ( ) const `[virtual]`

Returns the title of the printout.

Todo the python note here was wrong

virtual bool wxPrintout::HasPage (int *pageNum*) `[virtual]`

Should be overridden to return true if the document has this page, or false if not.

Returning false signifies the end of the document. By default, HasPage behaves as if the document has only one page.

Reimplemented in wxRichTextPrintout.

virtual bool wxPrintout::IsPreview ( ) const `[virtual]`

Returns true if the printout is currently being used for previewing.

See also

GetPreview()

void wxPrintout::MapScreenSizeToDevice ( )

Set the user scale and device origin of the wxDC associated with this wxPrintout so that one screen pixel maps to one device pixel on the DC.

That is, the user scale is set to (1,1) and the device origin is set to (0,0).

Use this if you want to do your own scaling prior to calling wxDC drawing calls, for example, if your underlying model is floating-point and you want to achieve maximum drawing precision on high-resolution printers.

You can use the GetLogicalXXXRect() routines below to obtain the paper rectangle, page rectangle, or page margins rectangle to perform your own scaling.

Note

While the underlying drawing model of Mac OS X is floating-point, wxWidgets's drawing model scales from integer coordinates.

void wxPrintout::MapScreenSizeToPage ( )

This sets the user scale of the wxDC associated with this wxPrintout to the same scale as MapScreenSizeToPaper() but sets the logical origin to the top left corner of the page rectangle.

void wxPrintout::MapScreenSizeToPageMargins (const wxPageSetupDialogData & *pageSetupData*)

This sets the user scale of the wxDC associated with this wxPrintout to the same scale as MapScreenSizeToPage↩ Margins() but sets the logical origin to the top left corner of the page margins specified by the given wxPageSetup↩ DialogData object.

void wxPrintout::MapScreenSizeToPaper ( )

Set the user scale and device origin of the wxDC associated with this wxPrintout so that the printed page matches the screen size as closely as possible and the logical origin is in the top left corner of the paper rectangle.

That is, a 100-pixel object on screen should appear at the same size on the printed page. (It will, of course, be larger or smaller in the preview image, depending on the zoom factor.)

Use this if you want WYSIWYG behaviour, e.g., in a text editor.

void wxPrintout::OffsetLogicalOrigin (wxCoord *xoff*, wxCoord *yoff*)

Shift the device origin by an amount specified in logical coordinates.

virtual bool wxPrintout::OnBeginDocument (int *startPage*, int *endPage*) [virtual]

Called by the framework at the start of document printing.

Return false from this function cancels the print job.

OnBeginDocument() is called once for every copy printed.

Remarks

The base OnBeginDocument() must be called (and the return value checked) from within the overridden function, since it calls wxDC::StartDoc().

virtual void wxPrintout::OnBeginPrinting ( ) [virtual]

Called by the framework at the start of printing.

OnBeginPrinting() is called once for every print job (regardless of how many copies are being printed).

virtual void wxPrintout::OnEndDocument ( ) [virtual]

Called by the framework at the end of document printing.

OnEndDocument() is called once for every copy printed.

Remarks

The base OnEndDocument() must be called from within the overridden function, since it calls wxDC::EndDoc().

virtual void wxPrintout::OnEndPrinting ( ) [virtual]

Called by the framework at the end of printing.

OnEndPrinting is called once for every print job (regardless of how many copies are being printed).

virtual void wxPrintout::OnPreparePrinting ( ) [virtual]

Called once by the framework before any other demands are made of the wxPrintout object.

This gives the object an opportunity to calculate the number of pages in the document, for example.

Reimplemented in wxRichTextPrintout.

virtual bool wxPrintout::OnPrintPage (int *pageNum*) [pure virtual]

Called by the framework when a page should be printed.

Returning false cancels the print job.

Implemented in wxRichTextPrintout.

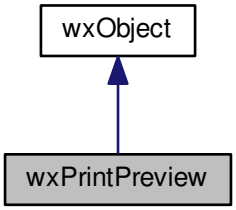
void wxPrintout::SetLogicalOrigin (wxCoord *x*, wxCoord *y*)

Set the device origin of the associated wxDC so that the current logical point becomes the new logical origin.

21.555 wxPrintPreview Class Reference

`#include <wx/print.h>`

Inheritance diagram for wxPrintPreview:



21.555.1 Detailed Description

Objects of this class manage the print preview process.

The object is passed a wxPrintout object, and the wxPrintPreview object itself is passed to a wxPreviewFrame object. Previewing is started by initializing and showing the preview frame. Unlike wxPrinter::Print(), flow of control returns to the application immediately after the frame is shown.

Note

The preview shown is only exact on Windows. On other platforms, the wxDC used for preview is different from what is used for printing and the results may be significantly different, depending on how is the output created. In particular, printing code relying on wxDC::GetTextExtent() heavily (for example, wxHtmlEasyPrinting and other wxHTML classes do) is affected. It is recommended to use native preview functionality on platforms that offer it (OS X, GTK+).

Library: wxCore

Category: Printing Framework

See also

Printing Framework Overview, wxPrinterDC, wxPrintDialog, wxPrintout, wxPrinter, wxPreviewCanvas, wxPreviewControlBar, wxPreviewFrame

Public Member Functions

- wxPrintPreview (wxPrintout *printout, wxPrintout *printoutForPrinting=NULL, wxPrintDialogData *data=NULL)

Constructor.

- wxPrintPreview (wxPrintout *printout, wxPrintout *printoutForPrinting, wxPrintData *data)
- ~wxPrintPreview ()

  Destructor.
- virtual wxPreviewCanvas * GetCanvas () const

  Gets the preview window used for displaying the print preview image.
- virtual int GetCurrentPage () const

  Gets the page currently being previewed.
- virtual wxFrame * GetFrame () const

  Gets the frame used for displaying the print preview canvas and control bar.
- virtual int GetMaxPage () const

  Returns the maximum page number.
- virtual int GetMinPage () const

  Returns the minimum page number.
- virtual wxPrintout * GetPrintout () const

  Gets the preview printout object associated with the wxPrintPreview object.
- virtual wxPrintout * GetPrintoutForPrinting () const

  Gets the printout object to be used for printing from within the preview interface, or NULL if none exists.
- virtual bool IsOk () const

  Returns true if the wxPrintPreview is valid, false otherwise.
- virtual bool PaintPage (wxPreviewCanvas *canvas, wxDC &dc)

  This refreshes the preview window with the preview image.
- virtual bool Print (bool prompt)

  Invokes the print process using the second wxPrintout object supplied in the wxPrintPreview constructor.
- virtual bool RenderPage (int pageNum)

  Renders a page into a wxMemoryDC.
- virtual void SetCanvas (wxPreviewCanvas *window)

  Sets the window to be used for displaying the print preview image.
- virtual bool SetCurrentPage (int pageNum)

  Sets the current page to be previewed.
- virtual void SetFrame (wxFrame *frame)

  Sets the frame to be used for displaying the print preview canvas and control bar.
- virtual void SetPrintout (wxPrintout *printout)

  Associates a printout object with the wxPrintPreview object.
- virtual void SetZoom (int percent)

  Sets the percentage preview zoom, and refreshes the preview canvas accordingly.

Additional Inherited Members

21.555.2 Constructor & Destructor Documentation

wxPrintPreview::wxPrintPreview (wxPrintout * *printout,* wxPrintout * *printoutForPrinting* = NULL, wxPrintDialogData * *data* = NULL)

Constructor.

Pass a printout object, an optional printout object to be used for actual printing, and the address of an optional block of printer data, which will be copied to the print preview object's print data.

If *printoutForPrinting* is non-NULL, a **"Print..."** button will be placed on the preview frame so that the user can print directly from the preview interface.

Remarks

Do not explicitly delete the printout objects once this constructor has been called, since they will be deleted in the wxPrintPreview destructor. The same does not apply to the *data* argument.

Use IsOk() to check whether the wxPrintPreview object was created correctly.

wxPrintPreview::wxPrintPreview (wxPrintout * *printout,* wxPrintout * *printoutForPrinting,* wxPrintData * *data*)

wxPrintPreview::~wxPrintPreview ( )

Destructor.

Deletes both print preview objects, so do not destroy these objects in your application.

21.555.3 Member Function Documentation

virtual wxPreviewCanvas* wxPrintPreview::GetCanvas ( ) const `[virtual]`

Gets the preview window used for displaying the print preview image.

virtual int wxPrintPreview::GetCurrentPage ( ) const `[virtual]`

Gets the page currently being previewed.

virtual wxFrame* wxPrintPreview::GetFrame ( ) const `[virtual]`

Gets the frame used for displaying the print preview canvas and control bar.

virtual int wxPrintPreview::GetMaxPage ( ) const `[virtual]`

Returns the maximum page number.

virtual int wxPrintPreview::GetMinPage ( ) const `[virtual]`

Returns the minimum page number.

virtual wxPrintout* wxPrintPreview::GetPrintout ( ) const `[virtual]`

Gets the preview printout object associated with the wxPrintPreview object.

virtual wxPrintout* wxPrintPreview::GetPrintoutForPrinting ( ) const `[virtual]`

Gets the printout object to be used for printing from within the preview interface, or NULL if none exists.

virtual bool wxPrintPreview::IsOk ( ) const `[virtual]`

Returns true if the wxPrintPreview is valid, false otherwise.

It could return false if there was a problem initializing the printer device context (current printer not set, for example).

virtual bool wxPrintPreview::PaintPage (wxPreviewCanvas * *canvas*, wxDC & *dc*) [virtual]

This refreshes the preview window with the preview image.

It must be called from the preview window's OnPaint member.

The implementation simply blits the preview bitmap onto the canvas, creating a new preview bitmap if none exists.

virtual bool wxPrintPreview::Print (bool *prompt*) [virtual]

Invokes the print process using the second wxPrintout object supplied in the wxPrintPreview constructor.

Will normally be called by the **Print**... panel item on the preview frame's control bar.

Returns false in case of error – call wxPrinter::GetLastError() to get detailed information about the kind of the error.

virtual bool wxPrintPreview::RenderPage (int *pageNum*) [virtual]

Renders a page into a wxMemoryDC.

Used internally by wxPrintPreview.

virtual void wxPrintPreview::SetCanvas (wxPreviewCanvas * *window*) [virtual]

Sets the window to be used for displaying the print preview image.

virtual bool wxPrintPreview::SetCurrentPage (int *pageNum*) [virtual]

Sets the current page to be previewed.

virtual void wxPrintPreview::SetFrame (wxFrame * *frame*) [virtual]

Sets the frame to be used for displaying the print preview canvas and control bar.

virtual void wxPrintPreview::SetPrintout (wxPrintout * *printout*) [virtual]

Associates a printout object with the wxPrintPreview object.

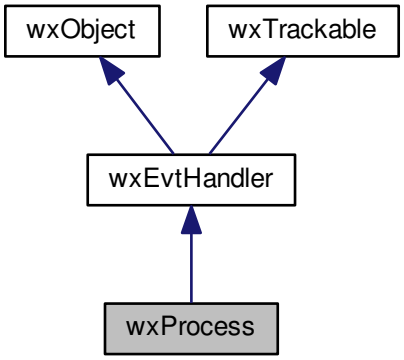
virtual void wxPrintPreview::SetZoom (int *percent*) [virtual]

Sets the percentage preview zoom, and refreshes the preview canvas accordingly.

21.556 wxProcess Class Reference

`#include <wx/process.h>`

Inheritance diagram for wxProcess:



21.556.1 Detailed Description

The objects of this class are used in conjunction with the wxExecute() function.

When a wxProcess object is passed to wxExecute(), its OnTerminate() virtual method is called when the process terminates. This allows the program to be (asynchronously) notified about the process termination and also retrieve its exit status which is unavailable from wxExecute() in the case of asynchronous execution.

Note

If the `wxEVT_END_PROCESS` event sent after termination is processed by the parent, then it is responsible for deleting the wxProcess object which sent it. However, if it is not processed, the object will **delete itself** and so the library users should only delete those objects whose notifications have been processed (and call wxProcess::Detach for others). This also means that unless you're going to process the `wxEVT_END_PROCESS` event, you **must** allocate the wxProcess class on the heap.

wxProcess also supports IO redirection of the child process. For this, you have to call its Redirect() method before passing it to wxExecute(). If the child process was launched successfully, GetInputStream(), GetOutputStream() and GetErrorStream() can then be used to retrieve the streams corresponding to the child process standard output, input and error output respectively.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxProcessEvent& event)

Event macros for events emitted by this class:

- EVT_END_PROCESS(id, func): Process a `wxEVT_END_PROCESS` event, sent by wxProcess::OnTerminate upon the external process termination.

Library: wxBase

Category: Application and Process Management

wxPerl Note: In wxPerl this class has an additional `Destroy` method, for explicit destruction.

See also

wxExecute(), External Program Execution Sample

Public Member Functions

- wxProcess (wxEvtHandler *parent=NULL, int id=-1)

  Constructs a process object.
- wxProcess (int flags)

  Creates an object without any associated parent (and hence no id neither) but allows to specify the flags which can have the value of `wxPROCESS_DEFAULT` or `wxPROCESS_REDIRECT`.
- virtual ~wxProcess ()

  Destroys the wxProcess object.
- void CloseOutput ()

  Closes the output stream (the one connected to the stdin of the child process).
- void Detach ()

  Detaches this event handler from the parent specified in the constructor (see wxEvtHandler::Unlink() for a similar but not identical function).
- wxInputStream * GetErrorStream () const

  Returns an input stream which corresponds to the standard error output (stderr) of the child process.
- wxInputStream * GetInputStream () const

  It returns an input stream corresponding to the standard output stream of the subprocess.
- wxOutputStream * GetOutputStream () const

  It returns an output stream corresponding to the input stream of the subprocess.
- long GetPid () const

  Returns the process ID of the process launched by Open() or set by wxExecute() (if you passed this wxProcess as argument).
- bool IsErrorAvailable () const

  Returns true if there is data to be read on the child process standard error stream.
- bool IsInputAvailable () const

  Returns true if there is data to be read on the child process standard output stream.
- bool IsInputOpened () const

  Returns true if the child process standard output stream is opened.
- virtual void OnTerminate (int pid, int status)

  It is called when the process with the pid pid finishes.
- void Redirect ()

  Turns on redirection.
- void SetPriority (unsigned priority)

  Sets the priority of the process, between 0 (lowest) and 100 (highest).

Static Public Member Functions

- static bool Exists (int pid)

  Returns true if the given process exists in the system.
- static wxKillError Kill (int pid, wxSignal sig=wxSIGTERM, int flags=wxKILL_NOCHILDREN)

  Send the specified signal to the given process.
- static wxProcess * Open (const wxString &cmd, int flags=wxEXEC_ASYNC)

  This static method replaces the standard `popen()` function: it launches the process specified by the cmd parameter and returns the wxProcess object which can be used to retrieve the streams connected to the standard input, output and error output of the child process.

Additional Inherited Members

21.556.2 Constructor & Destructor Documentation

wxProcess::wxProcess (wxEvtHandler * *parent* = NULL, int *id* = -1)

Constructs a process object.

id is only used in the case you want to use wxWidgets events. It identifies this object, or another window that will receive the event.

If the *parent* parameter is different from NULL, it will receive a wxEVT_END_PROCESS notification event (you should insert EVT_END_PROCESS macro in the event table of the parent to handle it) with the given *id*.

Parameters

parent	The event handler parent.
id	id of an event.

wxProcess::wxProcess (int *flags*)

Creates an object without any associated parent (and hence no id neither) but allows to specify the *flags* which can have the value of wxPROCESS_DEFAULT or wxPROCESS_REDIRECT.

Specifying the former value has no particular effect while using the latter one is equivalent to calling Redirect().

virtual wxProcess::~wxProcess ( ) [virtual]

Destroys the wxProcess object.

21.556.3 Member Function Documentation

void wxProcess::CloseOutput ( )

Closes the output stream (the one connected to the stdin of the child process).

This function can be used to indicate to the child process that there is no more data to be read - usually, a filter program will only terminate when the input stream is closed.

Notice that GetOutputStream() will return NULL after the output stream is closed.

void wxProcess::Detach ( )

Detaches this event handler from the parent specified in the constructor (see wxEvtHandler::Unlink() for a similar but not identical function).

Normally, a wxProcess object is deleted by its parent when it receives the notification about the process termination.

However, it might happen that the parent object is destroyed before the external process is terminated (e.g. a window from which this external process was launched is closed by the user) and in this case it **should** not delete the wxProcess object, but **should** call Detach() instead.

After the wxProcess object is detached from its parent, no notification events will be sent to the parent and the object will delete itself upon reception of the process termination notification.

static bool wxProcess::Exists (int *pid*) [static]

Returns true if the given process exists in the system.

See also

Kill(), Exec sample

wxInputStream* wxProcess::GetErrorStream ( ) const

Returns an input stream which corresponds to the standard error output (stderr) of the child process.

wxInputStream* wxProcess::GetInputStream ( ) const

It returns an input stream corresponding to the standard output stream of the subprocess.

If it is NULL, you have not turned on the redirection.

See also

Redirect().

wxOutputStream* wxProcess::GetOutputStream ( ) const

It returns an output stream corresponding to the input stream of the subprocess.

If it is NULL, you have not turned on the redirection or already called CloseOutput().

See also

Redirect().

long wxProcess::GetPid ( ) const

Returns the process ID of the process launched by Open() or set by wxExecute() (if you passed this wxProcess as argument).

bool wxProcess::IsErrorAvailable ( ) const

Returns true if there is data to be read on the child process standard error stream.

See also

IsInputAvailable()

bool wxProcess::IsInputAvailable ( ) const

Returns true if there is data to be read on the child process standard output stream.

This allows to write simple (and extremely inefficient) polling-based code waiting for a better mechanism in future wxWidgets versions. See the exec sample for an example of using this function.

See also

IsInputOpened()

bool wxProcess::IsInputOpened ( ) const

Returns true if the child process standard output stream is opened.

static wxKillError wxProcess::Kill (int *pid*, wxSignal *sig* = wxSIGTERM, int *flags* = wxKILL_NOCHILDREN) [static]

Send the specified signal to the given process.

Possible signal values can be one of the wxSignal enumeration values.

`wxSIGNONE`, `wxSIGKILL` and `wxSIGTERM` have the same meaning under both Unix and Windows but all the other signals are equivalent to `wxSIGTERM` under Windows.

The *flags* parameter can be `wxKILL_NOCHILDREN` (the default), or `wxKILL_CHILDREN`, in which case the child processes of this process will be killed too. Note that under Unix, for `wxKILL_CHILDREN` to work you should have created the process passing `wxEXEC_MAKE_GROUP_LEADER`.

Returns the element of wxKillError enum.

See also

Exists(), wxKill(), Exec sample

virtual void wxProcess::OnTerminate (int *pid*, int *status*) [virtual]

It is called when the process with the pid *pid* finishes.

It raises a wxWidgets event when it isn't overridden.

Note that this function won't be called if you Kill() the process.

Parameters

pid	The pid of the process which has just terminated.
status	The exit code of the process.

static wxProcess* wxProcess::Open (const wxString & *cmd*, int *flags* = wxEXEC_ASYNC) [static]

This static method replaces the standard `popen()` function: it launches the process specified by the *cmd* parameter and returns the wxProcess object which can be used to retrieve the streams connected to the standard input, output and error output of the child process.

If the process couldn't be launched, NULL is returned.

Remarks

In any case the returned pointer should **not** be deleted, rather the process object will be destroyed automatically when the child process terminates. This does mean that the child process should be told to quit before the main program exits to avoid memory leaks.

Parameters

cmd	The command to execute, including optional arguments.
flags	The flags to pass to wxExecute(). Note: `wxEXEC_SYNC` should not be used.

Returns

A pointer to new wxProcess object or NULL on error.

See also

wxExecute()

void wxProcess::Redirect ( )

Turns on redirection.

wxExecute() will try to open a couple of pipes to catch the subprocess stdio. The caught input stream is returned by GetOutputStream() as a non-seekable stream. The caught output stream is returned by GetInputStream() as a non-seekable stream.

void wxProcess::SetPriority (unsigned *priority*)

Sets the priority of the process, between 0 (lowest) and 100 (highest).

It can only be set before the process is created.

The following symbolic constants can be used in addition to raw values in 0..100 range:

- **wxPRIORITY_MIN:** 0
- **wxPRIORITY_DEFAULT:** 50
- **wxPRIORITY_MAX:** 100

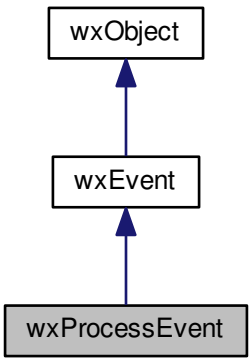
Since

2.9.5

21.557 wxProcessEvent Class Reference

`#include <wx/process.h>`

Inheritance diagram for wxProcessEvent:



21.557.1 Detailed Description

A process event is sent to the wxEvtHandler specified to wxProcess when a process is terminated.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxProcessEvent& event)

Event macros:

- EVT_END_PROCESS(id, func): Process a `wxEVT_END_PROCESS` event. *id* is the identifier of the process object (the id passed to the wxProcess constructor) or a window to receive the event.

Library: wxBase

Category: Events

See also

wxProcess, Events and Event Handling

Public Member Functions

- wxProcessEvent (int id=0, int pid=0, int exitcode=0)

  Constructor.
- int GetExitCode ()

  Returns the exist status.
- int GetPid ()

  Returns the process id.

Additional Inherited Members

21.557.2 Constructor & Destructor Documentation

wxProcessEvent::wxProcessEvent (int *id* = `0`, int *pid* = `0`, int *exitcode* = `0`)

Constructor.

Takes a wxProcessObject or window id, a process id and an exit status.

21.557.3 Member Function Documentation

int wxProcessEvent::GetExitCode ( )

Returns the exist status.

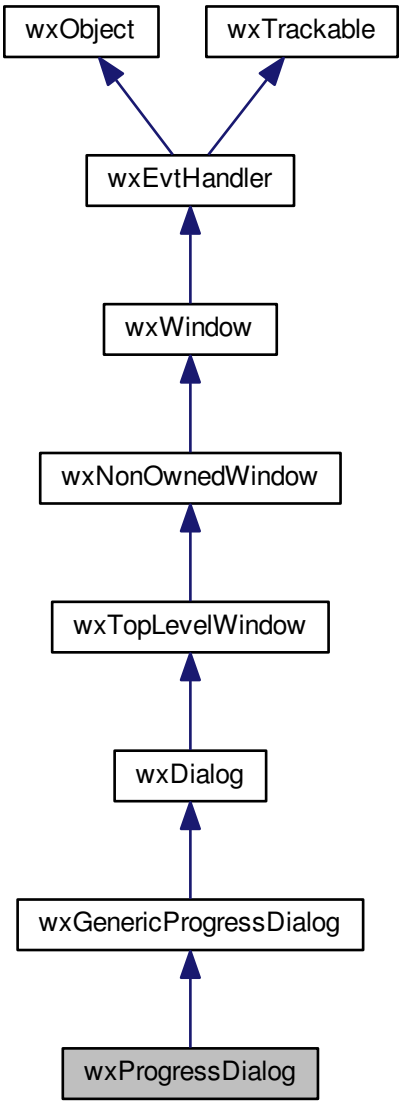
int wxProcessEvent::GetPid ( )

Returns the process id.

21.558 wxProgressDialog Class Reference

```
#include <wx/progdlg.h>
```

Inheritance diagram for wxProgressDialog:



21.558.1 Detailed Description

If supported by the platform this class will provide the platform's native progress dialog, else it will simply be the *wxGenericProgressDialog*.

Public Member Functions

- wxProgressDialog (const wxString &title, const wxString &message, int maximum=100, wxWindow *parent=NULL, int style=wxPD_APP_MODAL|wxPD_AUTO_HIDE)

Additional Inherited Members

21.558.2 Constructor & Destructor Documentation

wxProgressDialog::wxProgressDialog (const wxString & *title,* const wxString & *message,* int *maximum* = 100, wxWindow * *parent* = NULL, int *style* = wxPD_APP_MODAL|wxPD_AUTO_HIDE)

21.559 wxPropagateOnce Class Reference

`#include <wx/event.h>`

21.559.1 Detailed Description

Helper class to temporarily lower propagation level.

Public Member Functions

- wxPropagateOnce (wxEvent &event)
- ~wxPropagateOnce ()

21.559.2 Constructor & Destructor Documentation

wxPropagateOnce::wxPropagateOnce (wxEvent & *event*)

wxPropagateOnce::~wxPropagateOnce ( )

21.560 wxPropagationDisabler Class Reference

`#include <wx/event.h>`

21.560.1 Detailed Description

Helper class to temporarily change an event to not propagate.

Public Member Functions

- wxPropagationDisabler (wxEvent &event)
- ~wxPropagationDisabler ()

21.560.2 Constructor & Destructor Documentation

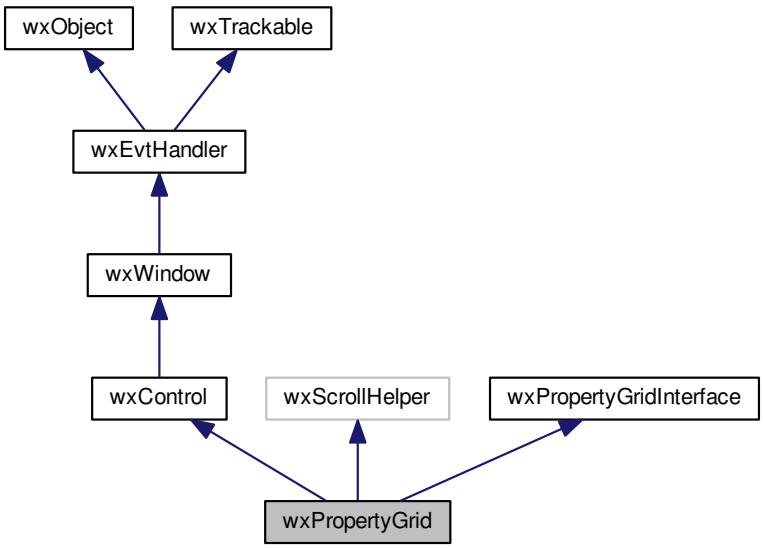
wxPropagationDisabler::wxPropagationDisabler (wxEvent & *event*)

wxPropagationDisabler::~wxPropagationDisabler ( )

21.561 wxPropertyGrid Class Reference

`#include <wx/propgrid/propgrid.h>`

Inheritance diagram for wxPropertyGrid:



21.561.1 Detailed Description

wxPropertyGrid is a specialized grid for editing properties - in other words name = value pairs.

List of ready-to-use property classes include strings, numbers, flag sets, fonts, colours and many others. It is possible, for example, to categorize properties, set up a complete tree-hierarchy, add more than two columns, and set arbitrary per-property attributes.

Please note that most member functions are inherited and as such not documented on this page. This means you will probably also want to read wxPropertyGridInterface class reference.

See also wxPropertyGrid Overview.

21.561.2 Window Styles

See wxPropertyGrid Window Styles.

21.561.3 Event Handling

To process input from a property grid control, use these event handler macros to direct input to member functions that take a wxPropertyGridEvent argument.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxPropertyGridEvent& event)

Event macros for events emitted by this class:

- EVT_PG_SELECTED (id, func): Respond to `wxEVT_PG_SELECTED` event, generated when a property selection has been changed, either by user action or by indirect program function. For instance, collaps-

ing a parent property programmatically causes any selected child property to become unselected, and may therefore cause this event to be generated.

- EVT_PG_CHANGED(id, func): Respond to `wxEVT_PG_CHANGED` event, generated when property value has been changed by the user.
- EVT_PG_CHANGING(id, func): Respond to `wxEVT_PG_CHANGING` event, generated when property value is about to be changed by user. Use wxPropertyGridEvent::GetValue() to take a peek at the pending value, and wxPropertyGridEvent::Veto() to prevent change from taking place, if necessary.
- EVT_PG_HIGHLIGHTED(id, func): Respond to `wxEVT_PG_HIGHLIGHTED` event, which occurs when mouse moves over a property. Event's property is NULL if hovered area does not belong to any property.
- EVT_PG_RIGHT_CLICK(id, func): Respond to `wxEVT_PG_RIGHT_CLICK` event, which occurs when property is clicked on with right mouse button.
- EVT_PG_DOUBLE_CLICK(id, func): Respond to `wxEVT_PG_DOUBLE_CLICK` event, which occurs when property is double-clicked on with left mouse button.
- EVT_PG_ITEM_COLLAPSED(id, func): Respond to `wxEVT_PG_ITEM_COLLAPSED` event, generated when user collapses a property or category.
- EVT_PG_ITEM_EXPANDED(id, func): Respond to `wxEVT_PG_ITEM_EXPANDED` event, generated when user expands a property or category.
- EVT_PG_LABEL_EDIT_BEGIN(id, func): Respond to `wxEVT_PG_LABEL_EDIT_BEGIN` event, generated when user is about to begin editing a property label. You can veto this event to prevent the action.
- EVT_PG_LABEL_EDIT_ENDING(id, func): Respond to `wxEVT_PG_LABEL_EDIT_ENDING` event, generated when user is about to end editing of a property label. You can veto this event to prevent the action.
- EVT_PG_COL_BEGIN_DRAG(id, func): Respond to `wxEVT_PG_COL_BEGIN_DRAG` event, generated when user starts resizing a column - can be vetoed.
- EVT_PG_COL_DRAGGING, (id, func): Respond to `wxEVT_PG_COL_DRAGGING`, event, generated when a column resize by user is in progress. This event is also generated when user double-clicks the splitter in order to recenter it.
- EVT_PG_COL_END_DRAG(id, func): Respond to `wxEVT_PG_COL_END_DRAG` event, generated after column resize by user has finished.

Remarks

Use Freeze() and Thaw() respectively to disable and enable drawing. This will also delay sorting etc. miscellaneous calculations to the last possible moment.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPropertyGrid ()

  Two step constructor.
- wxPropertyGrid (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxPG_DEFAULT_STYLE, const wxString &name=wx↩PropertyGridNameStr)

  Constructor.
- virtual ~wxPropertyGrid ()

Destructor.

- void AddActionTrigger (int action, int keycode, int modifiers=0)

  Adds given key combination to trigger given action.
- bool AddToSelection (wxPGPropArg id)

  Adds given property into selection.
- void BeginLabelEdit (unsigned int colIndex=0)

  Creates label editor wxTextCtrl for given column, for property that is currently selected.
- bool ChangePropertyValue (wxPGPropArg id, wxVariant newValue)

  Changes value of a property, as if from an editor.
- void CenterSplitter (bool enableAutoResizing=false)

  Centers the splitter.
- virtual void Clear ()

  Deletes all properties.
- void ClearActionTriggers (int action)

  Clears action triggers for given action.
- virtual bool CommitChangesFromEditor (wxUint32 flags=0)

  Forces updating the value of property from the editor control.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxPG_DEFAULT_STYLE, const wxString &name=wxPropertyGridNameStr)

  Two step creation.
- void DedicateKey (int keycode)

  Dedicates a specific keycode to wxPropertyGrid.
- bool EnableCategories (bool enable)

  Enables or disables (shows/hides) categories according to parameter enable.
- void EndLabelEdit (bool commit=true)

  Destroys label editor wxTextCtrl, if any.
- bool EnsureVisible (wxPGPropArg id)

  Scrolls and/or expands items to ensure that the given item is visible.
- wxSize FitColumns ()

  Reduces column sizes to minimum possible, while still retaining fully visible grid contents (labels, images).
- wxTextCtrl * GetLabelEditor () const

  Returns currently active label editor, NULL if none.
- wxWindow * GetPanel ()

  Returns wxWindow that the properties are painted on, and which should be used as the parent for editor controls.
- wxColour GetCaptionBackgroundColour () const

  Returns current category caption background colour.
- wxFont & GetCaptionFont ()

  Returns current category caption font.
- wxColour GetCaptionForegroundColour () const

  Returns current category caption text colour.
- wxColour GetCellBackgroundColour () const

  Returns current cell background colour.
- wxColour GetCellDisabledTextColour () const

  Returns current cell text colour when disabled.
- wxColour GetCellTextColour () const

  Returns current cell text colour.
- unsigned int GetColumnCount () const

  Returns number of columns currently on grid.
- wxColour GetEmptySpaceColour () const

  Returns colour of empty space below properties.

- int GetFontHeight () const

  Returns height of highest characters of used font.
- wxPropertyGrid * GetGrid ()

  Returns pointer to itself.
- wxRect GetImageRect (wxPGProperty *property, int item) const

  Returns rectangle of custom paint image.
- wxSize GetImageSize (wxPGProperty *property=NULL, int item=-1) const

  Returns size of the custom paint image in front of property.
- wxPGProperty * GetLastItem (int flags=wxPG_ITERATE_DEFAULT)

  Returns last item which could be iterated using given flags.
- wxColour GetLineColour () const

  Returns colour of lines between cells.
- wxColour GetMarginColour () const

  Returns background colour of margin.
- wxPGProperty * GetRoot () const

  Returns "root property".
- int GetRowHeight () const

  Returns height of a single grid row (in pixels).
- wxPGProperty * GetSelectedProperty () const

  Returns currently selected property.
- wxPGProperty * GetSelection () const

  Returns currently selected property.
- wxColour GetSelectionBackgroundColour () const

  Returns current selection background colour.
- wxColour GetSelectionForegroundColour () const

  Returns current selection text colour.
- wxPGSortCallback GetSortFunction () const

  Returns the property sort function (default is NULL).
- int GetSplitterPosition (unsigned int splitterIndex=0) const

  Returns current splitter x position.
- wxTextCtrl * GetEditorTextCtrl () const

  Returns wxTextCtrl active in currently selected property, if any.
- const wxPGCell & GetUnspecifiedValueAppearance () const

  Returns current appearance of unspecified value cells.
- wxString GetUnspecifiedValueText (int argFlags=0) const

  Returns (visual) text representation of the unspecified property value.
- int GetVerticalSpacing () const

  Returns current vertical spacing.
- wxPropertyGridHitTestResult HitTest (const wxPoint &pt) const

  Returns information about arbitrary position in the grid.
- bool IsAnyModified () const

  Returns true if any property has been modified by the user.
- bool IsEditorFocused () const

  Returns true if a property editor control has focus.
- bool IsFrozen () const

  Returns true if updating is frozen (ie.
- void MakeColumnEditable (unsigned int column, bool editable=true)

  Makes given column editable by user.
- void OnTLPChanging (wxWindow *newTLP)

  It is recommended that you call this function any time your code causes wxPropertyGrid's top-level parent to change.
- void RefreshEditor ()

Refreshes any active editor control.

- virtual void RefreshProperty (wxPGProperty *p)

  Redraws given property.
- void ResetColours ()

  Resets all colours to the original system values.
- void ResetColumnSizes (bool enableAutoResizing=false)

  Resets column sizes and splitter positions, based on proportions.
- bool RemoveFromSelection (wxPGPropArg id)

  Removes given property from selection.
- bool SelectProperty (wxPGPropArg id, bool focus=false)

  Selects a property.
- void SetCaptionBackgroundColour (const wxColour &col)

  Sets category caption background colour.
- void SetCaptionTextColour (const wxColour &col)

  Sets category caption text colour.
- void SetCellBackgroundColour (const wxColour &col)

  Sets default cell background colour - applies to property cells.
- void SetCellDisabledTextColour (const wxColour &col)

  Sets cell text colour for disabled properties.
- void SetCellTextColour (const wxColour &col)

  Sets default cell text colour - applies to property name and value text.
- void SetColumnCount (int colCount)

  Set number of columns (2 or more).
- void SetCurrentCategory (wxPGPropArg id)

  Sets the 'current' category - Append will add non-category properties under it.
- void SetEmptySpaceColour (const wxColour &col)

  Sets colour of empty space below properties.
- void SetLineColour (const wxColour &col)

  Sets colour of lines between cells.
- void SetMarginColour (const wxColour &col)

  Sets background colour of margin.
- void SetSelection (const wxArrayPGProperty &newSelection)

  Set entire new selection from given list of properties.
- void SetSelectionBackgroundColour (const wxColour &col)

  Sets selection background colour - applies to selected property name background.
- void SetSelectionTextColour (const wxColour &col)

  Sets selection foreground colour - applies to selected property name text.
- void SetSortFunction (wxPGSortCallback sortFunction)

  Sets the property sorting function.
- void SetSplitterPosition (int newxpos, int col=0)

  Sets x coordinate of the splitter.
- void SetSplitterLeft (bool privateChildrenToo=false)

  Moves splitter as left as possible, while still allowing all labels to be shown in full.
- void SetUnspecifiedValueAppearance (const wxPGCell &cell)

  Sets appearance of value cells representing an unspecified property value.
- void SetVerticalSpacing (int vspacing)

  Sets vertical spacing.

wxPropertyGrid customization

Reimplement these member functions in derived class for better control over wxPropertyGrid behaviour.

- virtual void DoShowPropertyError (wxPGProperty *property, const wxString &msg)
  Override in derived class to display error messages in custom manner (these message usually only result from validation failure).
- virtual void DoHidePropertyError (wxPGProperty *property)
  Override in derived class to hide an error displayed by DoShowPropertyError().
- virtual wxStatusBar * GetStatusBar ()
  Return wxStatusBar that is used by this wxPropertyGrid.

Property development functions

These member functions are usually only called when creating custom user properties.

- void EditorsValueWasModified ()
  Call when editor widget's contents is modified.
- void EditorsValueWasNotModified ()
  Reverse of EditorsValueWasModified().
- wxVariant GetUncommittedPropertyValue ()
  Returns most up-to-date value of selected property.
- bool IsEditorsValueModified () const
  Returns true if editor's value was marked modified.
- void ShowPropertyError (wxPGPropArg id, const wxString &msg)
  Shows an brief error message that is related to a property.
- bool WasValueChangedInEvent () const
  You can use this member function, for instance, to detect in wxPGProperty::OnEvent() if wxPGProperty::SetValueInEvent() was already called in wxPGEditor::OnEvent().

Static Public Member Functions

- static void AutoGetTranslation (bool enable)
  This static function enables or disables automatic use of wxGetTranslation() for following strings: wxEnumProperty list labels, wxFlagsProperty child property labels.
- static wxPGEditor * RegisterEditorClass (wxPGEditor *editor, bool noDefCheck=false)
  Forwards to DoRegisterEditorClass with empty name.
- static wxPGEditor * DoRegisterEditorClass (wxPGEditor *editor, const wxString &name, bool noDefCheck=false)
  Registers a new editor class.

Additional Inherited Members

21.561.4 Constructor & Destructor Documentation

wxPropertyGrid::wxPropertyGrid ( )

Two step constructor.

Call Create() when this constructor is called to build up the wxPropertyGrid

wxPropertyGrid::wxPropertyGrid (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxPG_DEFAULT_STYLE, const wxString & *name* = `wxPropertyGridNameStr`)

Constructor.

The styles to be used are styles valid for the wxWindow.

See also

wxPropertyGrid Window Styles.

virtual wxPropertyGrid::~wxPropertyGrid ( ) [virtual]

Destructor.

21.561.5 Member Function Documentation

void wxPropertyGrid::AddActionTrigger (int *action,* int *keycode,* int *modifiers* = 0)

Adds given key combination to trigger given action.

Here is a sample code to make Enter key press move focus to the next property.

```
propGrid->AddActionTrigger(wxPG_ACTION_NEXT_PROPERTY,
                           WXK_RETURN);
propGrid->DedicateKey(WXK_RETURN);
```

Parameters

action	Which action to trigger. See wxPropertyGrid Action Identifiers.
keycode	Which keycode triggers the action.
modifiers	Which key event modifiers, in addition to keycode, are needed to trigger the action.

bool wxPropertyGrid::AddToSelection (wxPGPropArg *id*)

Adds given property into selection.

If wxPG_EX_MULTIPLE_SELECTION extra style is not used, then this has same effect as calling SelectProperty().

Remarks

Multiple selection is not supported for categories. This means that if you have properties selected, you cannot add category to selection, and also if you have category selected, you cannot add other properties to selection. This member function will fail silently in these cases, even returning true.

static void wxPropertyGrid::AutoGetTranslation (bool *enable*) [static]

This static function enables or disables automatic use of wxGetTranslation() for following strings: wxEnumProperty list labels, wxFlagsProperty child property labels.

Default is false.

void wxPropertyGrid::BeginLabelEdit (unsigned int *colIndex* = 0)

Creates label editor wxTextCtrl for given column, for property that is currently selected.

When multiple selection is enabled, this applies to whatever property GetSelection() returns.

Parameters

colIndex	Which column's label to edit. Note that you should not use value 1, which is reserved for property value column.

See also

EndLabelEdit(), MakeColumnEditable()

void wxPropertyGrid::CenterSplitter (bool *enableAutoResizing* = `false`)

Centers the splitter.

Parameters

enableAuto↩ Resizing	If true, automatic column resizing is enabled (only applicapple if window style wxPG_SPLI↩ TTER_AUTO_CENTER is used).

bool wxPropertyGrid::ChangePropertyValue (wxPGPropArg *id*, wxVariant *newValue*)

Changes value of a property, as if from an editor.

Use this instead of SetPropertyValue() if you need the value to run through validation process, and also send the property change event.

Returns

Returns true if value was successfully changed.

virtual void wxPropertyGrid::Clear ( ) [virtual]

Deletes all properties.

Implements wxPropertyGridInterface.

void wxPropertyGrid::ClearActionTriggers (int *action*)

Clears action triggers for given action.

Parameters

action	Which action to trigger. wxPropertyGrid Action Identifiers.

virtual bool wxPropertyGrid::CommitChangesFromEditor (wxUint32 *flags* = 0) [virtual]

Forces updating the value of property from the editor control.

Note that `wxEVT_PG_CHANGING` and `wxEVT_PG_CHANGED` are dispatched using ProcessEvent, meaning your event handlers will be called immediately.

Returns

Returns true if anything was changed.

bool wxPropertyGrid::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxPG_DEFAULT_STYLE, const wxString & *name* = `wxPropertyGridNameStr`)

Two step creation.

Whenever the control is created without any parameters, use Create to actually create it. Don't access the control's public methods before this is called

See also

wxPropertyGrid Window Styles.

void wxPropertyGrid::DedicateKey (int *keycode*)

Dedicates a specific keycode to wxPropertyGrid.

This means that such key presses will not be redirected to editor controls.

Using this function allows, for example, navigation between properties using arrow keys even when the focus is in the editor control.

virtual void wxPropertyGrid::DoHidePropertyError (wxPGProperty * *property*) [virtual]

Override in derived class to hide an error displayed by DoShowPropertyError().

See also

DoShowPropertyError()

static wxPGEditor* wxPropertyGrid::DoRegisterEditorClass (wxPGEditor * *editor,* const wxString & *name,* bool *noDefCheck* = false) [static]

Registers a new editor class.

Returns

Returns pointer to the editor class instance that should be used.

virtual void wxPropertyGrid::DoShowPropertyError (wxPGProperty * *property,* const wxString & *msg*) [virtual]

Override in derived class to display error messages in custom manner (these message usually only result from validation failure).

Remarks

If you implement this, then you also need to implement DoHidePropertyError() - possibly to do nothing, if error does not need hiding (e.g. it was logged or shown in a message box).

See also

DoHidePropertyError()

void wxPropertyGrid::EditorsValueWasModified ( )

Call when editor widget's contents is modified.

For example, this is called when changes text in wxTextCtrl (used in wxStringProperty and wxIntProperty).

Remarks

This function should only be called by custom properties.

See also

wxPGProperty::OnEvent()

void wxPropertyGrid::EditorsValueWasNotModified ( )

Reverse of EditorsValueWasModified().

Remarks

This function should only be called by custom properties.

bool wxPropertyGrid::EnableCategories (bool *enable*)

Enables or disables (shows/hides) categories according to parameter enable.

Remarks

This functions deselects selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

void wxPropertyGrid::EndLabelEdit (bool *commit* = `true`)

Destroys label editor wxTextCtrl, if any.

Parameters

commit	Use true (default) to store edited label text in property cell data.

See also

BeginLabelEdit(), MakeColumnEditable()

bool wxPropertyGrid::EnsureVisible (wxPGPropArg *id*)

Scrolls and/or expands items to ensure that the given item is visible.

Returns

Returns true if something was actually done.

wxSize wxPropertyGrid::FitColumns ( )

Reduces column sizes to minimum possible, while still retaining fully visible grid contents (labels, images).

Returns

Minimum size for the grid to still display everything.

Remarks

Does not work well with wxPG_SPLITTER_AUTO_CENTER window style.

```
This function only works properly if grid size prior to call was
already fairly large.

Note that you can also get calculated column widths by calling
GetState->GetColumnWidth() immediately after this function
returns.
```

wxColour wxPropertyGrid::GetCaptionBackgroundColour ( ) const

Returns current category caption background colour.

wxFont& wxPropertyGrid::GetCaptionFont ( )

Returns current category caption font.

wxColour wxPropertyGrid::GetCaptionForegroundColour ( ) const

Returns current category caption text colour.

wxColour wxPropertyGrid::GetCellBackgroundColour ( ) const

Returns current cell background colour.

wxColour wxPropertyGrid::GetCellDisabledTextColour ( ) const

Returns current cell text colour when disabled.

wxColour wxPropertyGrid::GetCellTextColour ( ) const

Returns current cell text colour.

unsigned int wxPropertyGrid::GetColumnCount ( ) const

Returns number of columns currently on grid.

wxTextCtrl* wxPropertyGrid::GetEditorTextCtrl ( ) const

Returns wxTextCtrl active in currently selected property, if any.

Takes wxOwnerDrawnComboBox into account.

wxColour wxPropertyGrid::GetEmptySpaceColour ( ) const

Returns colour of empty space below properties.

int wxPropertyGrid::GetFontHeight ( ) const

Returns height of highest characters of used font.

wxPropertyGrid* wxPropertyGrid::GetGrid ( )

Returns pointer to itself.

Dummy function that enables same kind of code to use wxPropertyGrid and wxPropertyGridManager.

wxRect wxPropertyGrid::GetImageRect (wxPGProperty * *property*, int *item*) const

Returns rectangle of custom paint image.

Parameters

property	Return image rectangle for this property.
item	Which choice of property to use (each choice may have different image).

wxSize wxPropertyGrid::GetImageSize (wxPGProperty * *property* = NULL, int *item* = -1) const

Returns size of the custom paint image in front of property.

Parameters

property	Return image rectangle for this property. If this argument is NULL, then preferred size is returned.
item	Which choice of property to use (each choice may have different image).

wxTextCtrl* wxPropertyGrid::GetLabelEditor ( ) const

Returns currently active label editor, NULL if none.

wxPGProperty* wxPropertyGrid::GetLastItem (int *flags* = wxPG_ITERATE_DEFAULT)

Returns last item which could be iterated using given flags.

Parameters

flags	See wxPropertyGridIterator Flags.

wxColour wxPropertyGrid::GetLineColour ( ) const

Returns colour of lines between cells.

wxColour wxPropertyGrid::GetMarginColour ( ) const

Returns background colour of margin.

wxWindow* wxPropertyGrid::GetPanel ( )

Returns wxWindow that the properties are painted on, and which should be used as the parent for editor controls.

wxPGProperty* wxPropertyGrid::GetRoot ( ) const

Returns "root property".

It does not have name, etc. and it is not visible. It is only useful for accessing its children.

int wxPropertyGrid::GetRowHeight ( ) const

Returns height of a single grid row (in pixels).

wxPGProperty* wxPropertyGrid::GetSelectedProperty ( ) const

Returns currently selected property.

wxPGProperty* wxPropertyGrid::GetSelection ( ) const

Returns currently selected property.

wxColour wxPropertyGrid::GetSelectionBackgroundColour ( ) const

Returns current selection background colour.

wxColour wxPropertyGrid::GetSelectionForegroundColour ( ) const

Returns current selection text colour.

wxPGSortCallback wxPropertyGrid::GetSortFunction ( ) const

Returns the property sort function (default is NULL).

See also

SetSortFunction

int wxPropertyGrid::GetSplitterPosition (unsigned int *splitterIndex* = 0) const

Returns current splitter x position.

virtual wxStatusBar* wxPropertyGrid::GetStatusBar ( ) [virtual]

Return wxStatusBar that is used by this wxPropertyGrid.

You can reimplement this member function in derived class to override the default behaviour of using the top-level wxFrame's status bar, if any.

wxVariant wxPropertyGrid::GetUncommittedPropertyValue ( )

Returns most up-to-date value of selected property.

This will return value different from GetSelectedProperty()->GetValue() only when text editor is activate and string edited by user represents valid, uncommitted property value.

const wxPGCell& wxPropertyGrid::GetUnspecifiedValueAppearance ( ) const

Returns current appearance of unspecified value cells.

See also

SetUnspecifiedValueAppearance()

wxString wxPropertyGrid::GetUnspecifiedValueText (int *argFlags* = 0) const

Returns (visual) text representation of the unspecified property value.

Parameters

argFlags	For internal use only.

int wxPropertyGrid::GetVerticalSpacing ( ) const

Returns current vertical spacing.

wxPropertyGridHitTestResult wxPropertyGrid::HitTest (const wxPoint & *pt*) const

Returns information about arbitrary position in the grid.

Parameters

pt	Coordinates in the virtual grid space. You may need to use wxScrolled<T>::CalcScrolled↩ Position() for translating wxPropertyGrid client coordinates into something this member function can use.

bool wxPropertyGrid::IsAnyModified ( ) const

Returns true if any property has been modified by the user.

bool wxPropertyGrid::IsEditorFocused ( ) const

Returns true if a property editor control has focus.

bool wxPropertyGrid::IsEditorsValueModified ( ) const

Returns true if editor's value was marked modified.

bool wxPropertyGrid::IsFrozen ( ) const

Returns true if updating is frozen (ie.

Freeze() called but not yet Thaw()).

void wxPropertyGrid::MakeColumnEditable (unsigned int *column*, bool *editable* = `true`)

Makes given column editable by user.

Parameters

column	The index of the column to make editable.
editable	Using false here will disable column from being editable.

See also

BeginLabelEdit(), EndLabelEdit()

void wxPropertyGrid::OnTLPChanging (wxWindow * *newTLP*)

It is recommended that you call this function any time your code causes wxPropertyGrid's top-level parent to change.

wxPropertyGrid's OnIdle() handler should be able to detect most changes, but it is not perfect.

Parameters

newTLP	New top-level parent that is about to be set. Old top-level parent window should still exist as the current one.

Remarks

This function is automatically called from wxPropertyGrid:: Reparent() and wxPropertyGridManager::↩ Reparent(). You only need to use it if you reparent wxPropertyGrid indirectly.

void wxPropertyGrid::RefreshEditor ( )

Refreshes any active editor control.

virtual void wxPropertyGrid::RefreshProperty (wxPGProperty * *p*) [virtual]

Redraws given property.

static wxPGEditor* wxPropertyGrid::RegisterEditorClass (wxPGEditor * *editor,* bool *noDefCheck* = false) [static]

Forwards to DoRegisterEditorClass with empty name.

bool wxPropertyGrid::RemoveFromSelection (wxPGPropArg *id*)

Removes given property from selection.

If property is not selected, an assertion failure will occur.

void wxPropertyGrid::ResetColours ( )

Resets all colours to the original system values.

void wxPropertyGrid::ResetColumnSizes (bool *enableAutoResizing* = false)

Resets column sizes and splitter positions, based on proportions.

Parameters

enableAuto↩ Resizing	If true, automatic column resizing is enabled (only applicapple if window style wxPG_SPLI↩ TTER_AUTO_CENTER is used).

See also

wxPropertyGridInterface::SetColumnProportion()

bool wxPropertyGrid::SelectProperty (wxPGPropArg *id,* bool *focus* = false)

Selects a property.

Editor widget is automatically created, but not focused unless focus is true.

Parameters

id	Property to select (name or pointer).
focus	If true, move keyboard focus to the created editor right away.

Returns

returns true if selection finished successfully. Usually only fails if current value in editor is not valid.

Remarks

In wxPropertyGrid 1.4, this member function used to generate `wxEVT_PG_SELECTED`. In wxWidgets 2.9 and later, it no longer does that.
This clears any previous selection.

See also

wxPropertyGridInterface::ClearSelection()

void wxPropertyGrid::SetCaptionBackgroundColour (const wxColour & *col*)

Sets category caption background colour.

void wxPropertyGrid::SetCaptionTextColour (const wxColour & *col*)

Sets category caption text colour.

void wxPropertyGrid::SetCellBackgroundColour (const wxColour & *col*)

Sets default cell background colour - applies to property cells.

Note that appearance of editor widgets may not be affected.

void wxPropertyGrid::SetCellDisabledTextColour (const wxColour & *col*)

Sets cell text colour for disabled properties.

void wxPropertyGrid::SetCellTextColour (const wxColour & *col*)

Sets default cell text colour - applies to property name and value text.

Note that appearance of editor widgets may not be affected.

void wxPropertyGrid::SetColumnCount (int *colCount*)

Set number of columns (2 or more).

void wxPropertyGrid::SetCurrentCategory (wxPGPropArg *id*)

Sets the 'current' category - Append will add non-category properties under it.

void wxPropertyGrid::SetEmptySpaceColour (const wxColour & *col*)

Sets colour of empty space below properties.

void wxPropertyGrid::SetLineColour (const wxColour & *col*)

Sets colour of lines between cells.

void wxPropertyGrid::SetMarginColour (const wxColour & *col*)

Sets background colour of margin.

void wxPropertyGrid::SetSelection (const wxArrayPGProperty & *newSelection*)

Set entire new selection from given list of properties.

void wxPropertyGrid::SetSelectionBackgroundColour (const wxColour & *col*)

Sets selection background colour - applies to selected property name background.

void wxPropertyGrid::SetSelectionTextColour (const wxColour & *col*)

Sets selection foreground colour - applies to selected property name text.

void wxPropertyGrid::SetSortFunction (wxPGSortCallback *sortFunction*)

Sets the property sorting function.

Parameters

sortFunction	The sorting function to be used. It should return a value greater than 0 if position of p1 is after p2. So, for instance, when comparing property names, you can use following implementation:

```
int MyPropertySortFunction(wxPropertyGrid* propGrid,
                           wxPGProperty* p1,
                           wxPGProperty* p2)
{
    return p1->GetBaseName().compare( p2->GetBaseName() );
}
```

Remarks

Default property sort function sorts properties by their labels (case-insensitively).

See also

GetSortFunction, wxPropertyGridInterface::Sort, wxPropertyGridInterface::SortChildren

void wxPropertyGrid::SetSplitterLeft (bool *privateChildrenToo* = `false`)

Moves splitter as left as possible, while still allowing all labels to be shown in full.

Parameters

privateChildren↩ Too	If false, will still allow private children to be cropped.

void wxPropertyGrid::SetSplitterPosition (int *newxpos,* int *col* = 0)

Sets x coordinate of the splitter.

Remarks

Splitter position cannot exceed grid size, and therefore setting it during form creation may fail as initial grid size is often smaller than desired splitter position, especially when sizers are being used.

void wxPropertyGrid::SetUnspecifiedValueAppearance (const wxPGCell & *cell*)

Sets appearance of value cells representing an unspecified property value.

Default appearance is blank.

Remarks

If you set the unspecified value to have any textual representation, then that will override "InlineHelp" attribute.

See also

wxPGProperty::SetValueToUnspecified(), wxPGProperty::IsValueUnspecified()

void wxPropertyGrid::SetVerticalSpacing (int *vspacing*)

Sets vertical spacing.

Can be 1, 2, or 3 - a value relative to font height. Value of 2 should be default on most platforms.

void wxPropertyGrid::ShowPropertyError (wxPGPropArg *id,* const wxString & *msg*)

Shows an brief error message that is related to a property.

bool wxPropertyGrid::WasValueChangedInEvent ( ) const

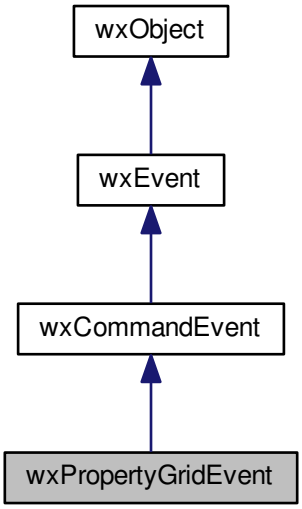
You can use this member function, for instance, to detect in wxPGProperty::OnEvent() if wxPGProperty::SetValue↩ InEvent() was already called in wxPGEditor::OnEvent().

It really only detects if was value was changed using wxPGProperty::SetValueInEvent(), which is usually used when a 'picker' dialog is displayed. If value was written by "normal means" in wxPGProperty::StringToValue() or IntTo↩ Value(), then this function will return false (on the other hand, wxPGProperty::OnEvent() is not even called in those cases).

21.562 wxPropertyGridEvent Class Reference

`#include <wx/propgrid/propgrid.h>`

Inheritance diagram for wxPropertyGridEvent:



21.562.1 Detailed Description

A property grid event holds information about events associated with wxPropertyGrid objects.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPropertyGridEvent (wxEventType commandType=0, int id=0)

  Constructor.
- wxPropertyGridEvent (const wxPropertyGridEvent &event)

  Copy constructor.
- ~wxPropertyGridEvent ()

  Destructor.
- bool CanVeto () const

  Returns true if you can veto the action that the event is signaling.
- unsigned int GetColumn () const

  Returns the column index associated with this event.
- wxPGProperty * GetMainParent () const

  Returns highest level non-category, non-root parent of property for which event occurred.
- wxPGProperty * GetProperty () const

  Returns property associated with this event.
- wxPGVFBFlags GetValidationFailureBehavior () const

  Returns current validation failure flags.

- wxString GetPropertyName () const

  Returns name of the associated property.
- wxVariant GetPropertyValue () const wxVariant GetValue() const

  Returns value of the associated property.
- void SetCanVeto (bool canVeto)

  Set if event can be vetoed.
- void SetProperty (wxPGProperty *p)

  Changes the property associated with this event.
- void SetValidationFailureBehavior (wxPGVFBFlags flags)

  Set override validation failure behaviour.
- void SetValidationFailureMessage (const wxString &message)

  Sets custom failure message for this time only.
- void Veto (bool veto=true)

  Call this from your event handler to veto action that the event is signaling.
- bool WasVetoed () const

  Returns true if event was vetoed.

Additional Inherited Members

21.562.2 Constructor & Destructor Documentation

wxPropertyGridEvent::wxPropertyGridEvent (wxEventType *commandType* = 0, int *id* = 0)

Constructor.

wxPropertyGridEvent::wxPropertyGridEvent (const wxPropertyGridEvent & *event*)

Copy constructor.

wxPropertyGridEvent::~wxPropertyGridEvent ( )

Destructor.

21.562.3 Member Function Documentation

bool wxPropertyGridEvent::CanVeto ( ) const

Returns true if you can veto the action that the event is signaling.

unsigned int wxPropertyGridEvent::GetColumn ( ) const

Returns the column index associated with this event.

For the column dragging events, it is the column to the left of the splitter being dragged

wxPGProperty* wxPropertyGridEvent::GetMainParent ( ) const

Returns highest level non-category, non-root parent of property for which event occurred.

Useful when you have nested properties with children.

Remarks

If immediate parent is root or category, this will return the property itself.

wxPGProperty* wxPropertyGridEvent::GetProperty ( ) const

Returns property associated with this event.

Remarks

You should assume that this property can always be NULL. For instance, `wxEVT_PG_SELECTED` is emitted not only when a new property is selected, but also when selection is cleared by user activity.

wxString wxPropertyGridEvent::GetPropertyName ( ) const

Returns name of the associated property.

Remarks

Property name is stored in event, so it remains accessible even after the associated property or the property grid has been deleted.

wxVariant wxPropertyGridEvent::GetPropertyValue ( ) const

Returns value of the associated property.

Works for all event types, but for `wxEVT_PG_CHANGING` this member function returns the value that is pending, so you can call Veto() if the value is not satisfactory.

Remarks

Property value is stored in event, so it remains accessible even after the associated property or the property grid has been deleted.

See also

GetPropertyValue()

wxPGVFBFlags wxPropertyGridEvent::GetValidationFailureBehavior ( ) const

Returns current validation failure flags.

void wxPropertyGridEvent::SetCanVeto (bool *canVeto*)

Set if event can be vetoed.

void wxPropertyGridEvent::SetProperty (wxPGProperty * *p*)

Changes the property associated with this event.

void wxPropertyGridEvent::SetValidationFailureBehavior (wxPGVFBFlags *flags*)

Set override validation failure behaviour.

Only effective if Veto() was also called, and only allowed if event type is `wxEVT_PG_CHANGING`.

void wxPropertyGridEvent::SetValidationFailureMessage (const wxString & *message*)

Sets custom failure message for this time only.

Only applies if wxPG_VFB_SHOW_MESSAGE is set in validation failure flags.

void wxPropertyGridEvent::Veto (bool *veto* = `true`)

Call this from your event handler to veto action that the event is signaling.

You can only veto a shutdown if wxPropertyGridEvent::CanVeto() returns true.

Remarks

Currently only `wxEVT_PG_CHANGING` supports vetoing.

bool wxPropertyGridEvent::WasVetoed ( ) const

Returns true if event was vetoed.

21.563 wxPropertyGridHitTestResult Struct Reference

`#include <wx/propgrid/propgridpagestate.h>`

21.563.1 Detailed Description

21.563.2 wxPropertyGridHitTestResult

A return value from wxPropertyGrid::HitTest(), contains all you need to know about an arbitrary location on the grid.

Public Member Functions

- wxPGProperty * GetProperty () const

Public Attributes

- int column

  Column.
- int splitter

  Index of splitter hit, -1 for none.
- int splitterHitOffset

  If splitter hit, offset to that.

Private Attributes

- wxPGProperty * property

  Property.

21.563.3 Member Function Documentation

wxPGProperty* wxPropertyGridHitTestResult::GetProperty ( ) const `[inline]`

21.563.4 Member Data Documentation

int wxPropertyGridHitTestResult::column

Column.

-1 for margin.

wxPGProperty* wxPropertyGridHitTestResult::property `[private]`

Property.

NULL if empty space below properties was hit

int wxPropertyGridHitTestResult::splitter

Index of splitter hit, -1 for none.

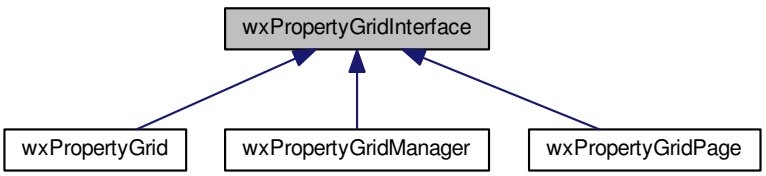
int wxPropertyGridHitTestResult::splitterHitOffset

If splitter hit, offset to that.

21.564 wxPropertyGridInterface Class Reference

`#include <wx/propgrid/propgridiface.h>`

Inheritance diagram for wxPropertyGridInterface:



21.564.1 Detailed Description

Most of the shared property manipulation interface shared by wxPropertyGrid, wxPropertyGridPage, and wx↩ PropertyGridManager is defined in this class.

Remarks

- In separate wxPropertyGrid component this class was known as wxPropertyContainerMethods.
- wxPropertyGridInterface's property operation member functions all accept a special wxPGPropArg id argument, using which you can refer to properties either by their pointer (for performance) or by their name (for conveniency).

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Types

- enum EditableStateFlags {
  SelectionState = 0x01,
  ExpandedState = 0x02,
  ScrollPosState = 0x04,
  PageState = 0x08,
  SplitterPosState = 0x10,
  DescBoxState = 0x20,
  AllStates }

Public Member Functions

- virtual ~wxPropertyGridInterface ()

  Destructor.
- wxPGProperty * Append (wxPGProperty *property)

  Appends property to the list.
- wxPGProperty * AppendIn (wxPGPropArg id, wxPGProperty *newProperty)

  Same as Append(), but appends under given parent property.
- void BeginAddChildren (wxPGPropArg id)

  In order to add new items into a property with private children (for instance, wxFlagsProperty), you need to call this method.
- virtual void Clear ()=0

  Deletes all properties.
- bool ClearSelection (bool validation=false)

  Clears current selection, if any.
- void ClearModifiedStatus ()

  Resets modified status of all properties.
- bool Collapse (wxPGPropArg id)

  Collapses given category or property with children.
- bool CollapseAll ()

  Collapses all items that can be collapsed.
- bool ChangePropertyValue (wxPGPropArg id, wxVariant newValue)

  Changes value of a property, as if by user.
- void DeleteProperty (wxPGPropArg id)

  Removes and deletes a property and any children.
- bool DisableProperty (wxPGPropArg id)

  Disables a property.
- bool EditorValidate ()

  Returns true if all property grid data changes have been committed.
- bool EnableProperty (wxPGPropArg id, bool enable=true)

  Enables or disables property.
- void EndAddChildren (wxPGPropArg id)

  Called after population of property with fixed children has finished.
- bool Expand (wxPGPropArg id)

  Expands given category or property with children.

- bool ExpandAll (bool expand=true)

  Expands all items that can be expanded.
- int GetColumnProportion (unsigned int column) const

  Returns auto-resize proportion of the given column.
- wxPGProperty * GetFirstChild (wxPGPropArg id)

  Returns id of first child of given property.
- wxPGProperty * GetFirst (int flags=wxPG_ITERATE_ALL)

  Returns id of first item that matches given criteria.
- wxPGProperty * GetProperty (const wxString &name) const

  Returns pointer to a property with given name (case-sensitive).
- void GetPropertiesWithFlag (wxArrayPGProperty *targetArr, wxPGProperty::FlagType flags, bool inverse=false, int iterFlags=(wxPG_ITERATE_PROPERTIES|wxPG_ITERATE_HIDDEN|wxPG_ITERATE_CATEGORIES)) const

  Adds to 'targetArr' pointers to properties that have given flags 'flags' set.
- wxVariant GetPropertyAttribute (wxPGPropArg id, const wxString &attrName) const

  Returns value of given attribute.
- wxColour GetPropertyBackgroundColour (wxPGPropArg id) const

  Returns background colour of first cell of a property.
- wxPropertyCategory * GetPropertyCategory (wxPGPropArg id) const

  Returns pointer of property's nearest parent category.
- void * GetPropertyClientData (wxPGPropArg id) const

  Returns client data (void) of a property.*
- wxPGProperty * GetPropertyByLabel (const wxString &label) const

  Returns first property which label matches given string.
- wxPGProperty * GetPropertyByName (const wxString &name) const

  Returns pointer to a property with given name (case-sensitive).
- wxPGProperty * GetPropertyByName (const wxString &name, const wxString &subname) const

  Returns child property 'subname' of property 'name'.
- const wxPGEditor * GetPropertyEditor (wxPGPropArg id) const

  Returns property's editor.
- wxString GetPropertyHelpString (wxPGPropArg id) const

  Returns help string associated with a property.
- wxBitmap * GetPropertyImage (wxPGPropArg id) const

  Returns property's custom value image (NULL of none).
- const wxString & GetPropertyLabel (wxPGPropArg id)

  Returns label of a property.
- wxString GetPropertyName (wxPGProperty *property)

  Returns property's name, by which it is globally accessible.
- wxColour GetPropertyTextColour (wxPGPropArg id) const

  Returns text colour of first cell of a property.
- wxValidator * GetPropertyValidator (wxPGPropArg id)

  Returns validator of a property as a reference, which you can pass to any number of SetPropertyValidator.
- wxVariant GetPropertyValue (wxPGPropArg id)

  Returns property's value as wxVariant.
- wxArrayInt GetPropertyValueAsArrayInt (wxPGPropArg id) const

  Return's property's value as wxArrayInt.
- wxArrayString GetPropertyValueAsArrayString (wxPGPropArg id) const

  Returns property's value as wxArrayString.
- bool GetPropertyValueAsBool (wxPGPropArg id) const

  Returns property's value as bool.
- wxDateTime GetPropertyValueAsDateTime (wxPGPropArg id) const

Return's property's value as wxDateTime.

- double GetPropertyValueAsDouble (wxPGPropArg id) const

  Returns property's value as double-precision floating point number.
- int GetPropertyValueAsInt (wxPGPropArg id) const

  Returns property's value as integer.
- long GetPropertyValueAsLong (wxPGPropArg id) const

  Returns property's value as integer.
- wxLongLong_t GetPropertyValueAsLongLong (wxPGPropArg id) const

  Returns property's value as native signed 64-bit integer.
- wxString GetPropertyValueAsString (wxPGPropArg id) const

  Returns property's value as wxString.
- unsigned long GetPropertyValueAsULong (wxPGPropArg id) const

  Returns property's value as unsigned integer.
- wxULongLong_t GetPropertyValueAsULongLong (wxPGPropArg id) const

  Returns property's value as native unsigned 64-bit integer.
- wxVariant GetPropertyValues (const wxString &listname=wxEmptyString, wxPGProperty *baseparent=NU↩ LL, long flags=0) const

  Returns a wxVariant list containing wxVariant versions of all property values.
- const wxArrayPGProperty & GetSelectedProperties () const

  Returns list of currently selected properties.
- wxPGProperty * GetSelection () const

  Returns currently selected property.
- virtual wxPGVIterator GetVIterator (int flags) const

  Similar to GetIterator(), but instead returns wxPGVIterator instance, which can be useful for forward-iterating through arbitrary property containers.
- bool HideProperty (wxPGPropArg id, bool hide=true, int flags=wxPG_RECURSE)

  Hides or reveals a property.
- wxPGProperty * Insert (wxPGPropArg priorThis, wxPGProperty *newProperty)

  Inserts property to the property container.
- wxPGProperty * Insert (wxPGPropArg parent, int index, wxPGProperty *newProperty)

  Inserts property to the property container.
- bool IsPropertyCategory (wxPGPropArg id) const

  Returns true if property is a category.
- bool IsPropertyEnabled (wxPGPropArg id) const

  Returns true if property is enabled.
- bool IsPropertyExpanded (wxPGPropArg id) const

  Returns true if given property is expanded.
- bool IsPropertyModified (wxPGPropArg id) const

  Returns true if property has been modified after value set or modify flag clear by software.
- virtual bool IsPropertySelected (wxPGPropArg id) const

  Returns true if property is selected.
- bool IsPropertyShown (wxPGPropArg id) const

  Returns true if property is shown (ie.
- bool IsPropertyValueUnspecified (wxPGPropArg id) const

  Returns true if property value is set to unspecified.
- void LimitPropertyEditing (wxPGPropArg id, bool limit=true)

  Disables (limit = true) or enables (limit = false) wxTextCtrl editor of a property, if it is not the sole mean to edit the value.
- wxPGProperty * RemoveProperty (wxPGPropArg id)

  Removes a property.
- wxPGProperty * ReplaceProperty (wxPGPropArg id, wxPGProperty *property)

Replaces property with id with newly created one.

- bool RestoreEditableState (const wxString &src, int restoreStates=AllStates)

  Restores user-editable state.
- wxString SaveEditableState (int includedStates=AllStates) const

  Used to acquire user-editable state (selected property, expanded properties, scrolled position, splitter positions).
- bool SetColumnProportion (unsigned int column, int proportion)

  Set proportion of a auto-stretchable column.
- void SetPropertyAttribute (wxPGPropArg id, const wxString &attrName, wxVariant value, long argFlags=0)

  Sets an attribute for this property.
- void SetPropertyAttributeAll (const wxString &attrName, wxVariant value)

  Sets property attribute for all applicapple properties.
- void SetPropertyBackgroundColour (wxPGPropArg id, const wxColour &colour, int flags=wxPG_RECURSE)

  Sets background colour of a property.
- void SetPropertyCell (wxPGPropArg id, int column, const wxString &text=wxEmptyString, const wxBitmap &bitmap=wxNullBitmap, const wxColour &fgCol=wxNullColour, const wxColour &bgCol=wxNullColour)

  Sets text, bitmap, and colours for given column's cell.
- void SetPropertyClientData (wxPGPropArg id, void ∗clientData)

  Sets client data (void∗) of a property.
- void SetPropertyColoursToDefault (wxPGPropArg id)

  Resets text and background colours of given property.
- void SetPropertyEditor (wxPGPropArg id, const wxPGEditor ∗editor)

  Sets editor for a property.
- void SetPropertyEditor (wxPGPropArg id, const wxString &editorName)

  Sets editor control of a property.
- void SetPropertyLabel (wxPGPropArg id, const wxString &newproplabel)

  Sets label of a property.
- void SetPropertyName (wxPGPropArg id, const wxString &newName)

  Sets name of a property.
- void SetPropertyReadOnly (wxPGPropArg id, bool set=true, int flags=wxPG_RECURSE)

  Sets property (and, recursively, its children) to have read-only value.
- void SetPropertyValueUnspecified (wxPGPropArg id)

  Sets property's value to unspecified.
- void SetPropertyValues (const wxVariantList &list, wxPGPropArg defaultCategory=wxNullProperty)

  Sets property values from a list of wxVariants.
- void SetPropertyValues (const wxVariant &list, wxPGPropArg defaultCategory=wxNullProperty)

  Sets property values from a list of wxVariants.
- void SetPropertyHelpString (wxPGPropArg id, const wxString &helpString)

  Associates the help string with property.
- void SetPropertyImage (wxPGPropArg id, wxBitmap &bmp)

  Set wxBitmap in front of the value.
- bool SetPropertyMaxLength (wxPGPropArg id, int maxLen)

  Sets max length of property's text.
- void SetPropertyTextColour (wxPGPropArg id, const wxColour &colour, int flags=wxPG_RECURSE)

  Sets text colour of a property.
- void SetPropertyValidator (wxPGPropArg id, const wxValidator &validator)

  Sets validator of a property.
- void SetPropertyValue (wxPGPropArg id, long value)

  Sets value (integer) of a property.
- void SetPropertyValue (wxPGPropArg id, int value)

  Sets value (integer) of a property.
- void SetPropertyValue (wxPGPropArg id, double value)

Sets value (floating point) of a property.

- void SetPropertyValue (wxPGPropArg id, bool value)

  Sets value (bool) of a property.
- void SetPropertyValue (wxPGPropArg id, const wxString &value)

  Sets value (string) of a property.
- void SetPropertyValue (wxPGPropArg id, const wxArrayString &value)

  Sets value (wxArrayString) of a property.
- void SetPropertyValue (wxPGPropArg id, const wxDateTime &value)

  Sets value (wxDateTime) of a property.
- void SetPropertyValue (wxPGPropArg id, wxObject *value)

  Sets value (wxObject) of a property.*
- void SetPropertyValue (wxPGPropArg id, wxObject &value)

  Sets value (wxObject&) of a property.
- void SetPropertyValue (wxPGPropArg id, wxLongLong_t value)

  Sets value (native 64-bit int) of a property.
- void SetPropertyValue (wxPGPropArg id, wxULongLong_t value)

  Sets value (native 64-bit unsigned int) of a property.
- void SetPropertyValue (wxPGPropArg id, const wxArrayInt &value)

  Sets value (wxArrayInt&) of a property.
- void SetPropertyValueString (wxPGPropArg id, const wxString &value)

  Sets value (wxString) of a property.
- void SetPropertyValue (wxPGPropArg id, wxVariant value)

  Sets value (wxVariant&) of a property.
- void SetValidationFailureBehavior (int vfbFlags)

  Adjusts how wxPropertyGrid behaves when invalid value is entered in a property.
- void Sort (int flags=0)

  Sorts all properties recursively.
- void SortChildren (wxPGPropArg id, int flags=0)

  Sorts children of a property.

- wxPropertyGridIterator GetIterator (int flags=wxPG_ITERATE_DEFAULT, wxPGProperty *firstProp=NULL)

  Returns iterator class instance.
- wxPropertyGridConstIterator GetIterator (int flags=wxPG_ITERATE_DEFAULT, wxPGProperty *firstProp=NULL) const

  Returns iterator class instance.

- wxPropertyGridIterator GetIterator (int flags, int startPos)

  Returns iterator class instance.
- wxPropertyGridConstIterator GetIterator (int flags, int startPos) const

  Returns iterator class instance.

Static Public Member Functions

- static void InitAllTypeHandlers ()

  Initializes all property types.
- static void RegisterAdditionalEditors ()

  Initializes additional property editors (SpinCtrl etc.).
- static void SetBoolChoices (const wxString &trueChoice, const wxString &falseChoice)

  Sets strings listed in the choice dropdown of a wxBoolProperty.
- static wxPGEditor * GetEditorByName (const wxString &editorName)

  Returns editor pointer of editor with given name;.

21.564.2 Member Enumeration Documentation

enum wxPropertyGridInterface::EditableStateFlags

Flags for wxPropertyGridInterface::SaveEditableState() and wxPropertyGridInterface::RestoreEditableState().

Enumerator

SelectionState Include selected property.

ExpandedState Include expanded/collapsed property information.

ScrollPosState Include scrolled position.

PageState Include selected page information. Only applies to wxPropertyGridManager.

SplitterPosState Include splitter position. Stored for each page.

DescBoxState Include description box size. Only applies to wxPropertyGridManager.

AllStates Include all supported user editable state information. This is usually the default value.

21.564.3 Constructor & Destructor Documentation

virtual wxPropertyGridInterface::~wxPropertyGridInterface ( ) [inline], [virtual]

Destructor.

21.564.4 Member Function Documentation

wxPGProperty* wxPropertyGridInterface::Append (wxPGProperty * *property*)

Appends property to the list.

wxPropertyGrid assumes ownership of the object. Becomes child of most recently added category.

Remarks

- wxPropertyGrid takes the ownership of the property pointer.
- If appending a category with name identical to a category already in the wxPropertyGrid, then newly created category is deleted, and most recently added category (under which properties are appended) is set to the one with same name. This allows easier adding of items to same categories in multiple passes.
- Does not automatically redraw the control, so you may need to call Refresh() when calling this function after control has been shown for the first time.
- This functions deselects selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PR↩ OPERTY is not respected, ie. selection is cleared even if editor had invalid value.

wxPGProperty* wxPropertyGridInterface::AppendIn (wxPGPropArg *id*, wxPGProperty * *newProperty*)

Same as Append(), but appends under given parent property.

Parameters

id	Name or pointer to parent property.
newProperty	Property to be added.

void wxPropertyGridInterface::BeginAddChildren (wxPGPropArg *id*)

In order to add new items into a property with private children (for instance, wxFlagsProperty), you need to call this method.

After populating has been finished, you need to call EndAddChildren().

See also

EndAddChildren()

bool wxPropertyGridInterface::ChangePropertyValue (wxPGPropArg *id,* wxVariant *newValue*)

Changes value of a property, as if by user.

Use this instead of SetPropertyValue() if you need the value to run through validation process, and also send the property change event.

Returns

Returns true if value was successfully changed.

virtual void wxPropertyGridInterface::Clear ( ) [pure virtual]

Deletes all properties.

Remarks

This functions deselects selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

Implemented in wxPropertyGrid, wxPropertyGridManager, and wxPropertyGridPage.

void wxPropertyGridInterface::ClearModifiedStatus ( )

Resets modified status of all properties.

bool wxPropertyGridInterface::ClearSelection (bool *validation* = false)

Clears current selection, if any.

Parameters

validation	If set to false, deselecting the property will always work, even if its editor had invalid value in it.

Returns

Returns true if successful or if there was no selection. May fail if validation was enabled and active editor had invalid value.

Remarks

In wxPropertyGrid 1.4, this member function used to send wxPG_EVT_SELECTED. In wxWidgets 2.9 and later, it no longer does that.

See also

wxPropertyGrid::SelectProperty()

bool wxPropertyGridInterface::Collapse (wxPGPropArg *id*)

Collapses given category or property with children.

Returns

Returns true if actually collapsed.

Remarks

This function may deselect selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

bool wxPropertyGridInterface::CollapseAll ( )

Collapses all items that can be collapsed.

Remarks

This functions clears selection. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

void wxPropertyGridInterface::DeleteProperty (wxPGPropArg *id*)

Removes and deletes a property and any children.

Parameters

id	Pointer or name of a property.

Remarks

If you delete a property in a wxPropertyGrid event handler, the actual deletion is postponed until the next idle event.

This functions deselects selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

bool wxPropertyGridInterface::DisableProperty (wxPGPropArg *id*)

Disables a property.

See also

EnableProperty(), wxPGProperty::Enable()

bool wxPropertyGridInterface::EditorValidate ( )

Returns true if all property grid data changes have been committed.

Usually only returns false if value in active editor has been invalidated by a wxValidator.

bool wxPropertyGridInterface::EnableProperty (wxPGPropArg *id,* bool *enable* = `true`)

Enables or disables property.

Disabled property usually appears as having grey text.

Parameters

id	Name or pointer to a property.
enable	If false, property is disabled instead.

See also

wxPGProperty::Enable()

void wxPropertyGridInterface::EndAddChildren (wxPGPropArg *id*)

Called after population of property with fixed children has finished.

See also

BeginAddChildren()

bool wxPropertyGridInterface::Expand (wxPGPropArg *id*)

Expands given category or property with children.

Returns

Returns true if actually expanded.

Remarks

This function may deselect selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

bool wxPropertyGridInterface::ExpandAll (bool *expand* = `true`)

Expands all items that can be expanded.

Remarks

This functions clears selection. Validation failure option wxPG_VFB_STAY_IN_PROPERTY is not respected, ie. selection is cleared even if editor had invalid value.

int wxPropertyGridInterface::GetColumnProportion (unsigned int *column*) const

Returns auto-resize proportion of the given column.

See also

SetColumnProportion()

static wxPGEditor* wxPropertyGridInterface::GetEditorByName (const wxString & *editorName*) `[static]`

Returns editor pointer of editor with given name;.

wxPGProperty* wxPropertyGridInterface::GetFirst (int *flags* = wxPG_ITERATE_ALL)

Returns id of first item that matches given criteria.

Parameters

flags	See wxPropertyGridIterator Flags.

wxPGProperty∗ wxPropertyGridInterface::GetFirstChild (wxPGPropArg *id*)

Returns id of first child of given property.

Remarks

Does not return private children!

wxPropertyGridIterator wxPropertyGridInterface::GetIterator (int *flags* = wxPG_ITERATE_DEFAULT, wxPGProperty ∗ *firstProp* = NULL)

Returns iterator class instance.

Parameters

flags	See wxPropertyGridIterator Flags. Value wxPG_ITERATE_DEFAULT causes iteration over everything except private child properties.
firstProp	Property to start iteration from. If NULL, then first child of root is used.

wxPropertyGridConstIterator wxPropertyGridInterface::GetIterator (int *flags* = wxPG_ITERATE_DEFAULT, wxPGProperty ∗ *firstProp* = NULL) const

Returns iterator class instance.

Parameters

flags	See wxPropertyGridIterator Flags. Value wxPG_ITERATE_DEFAULT causes iteration over everything except private child properties.
firstProp	Property to start iteration from. If NULL, then first child of root is used.

wxPropertyGridIterator wxPropertyGridInterface::GetIterator (int *flags*, int *startPos*)

Returns iterator class instance.

Parameters

flags	See wxPropertyGridIterator Flags. Value wxPG_ITERATE_DEFAULT causes iteration over everything except private child properties.
startPos	Either wxTOP or wxBOTTOM. wxTOP will indicate that iterations start from the first property from the top, and wxBOTTOM means that the iteration will instead begin from bottommost valid item.

wxPropertyGridConstIterator wxPropertyGridInterface::GetIterator (int *flags*, int *startPos*) const

Returns iterator class instance.

Parameters

flags	See wxPropertyGridIterator Flags. Value wxPG_ITERATE_DEFAULT causes iteration over everything except private child properties.
startPos	Either wxTOP or wxBOTTOM. wxTOP will indicate that iterations start from the first property from the top, and wxBOTTOM means that the iteration will instead begin from bottommost valid item.

void wxPropertyGridInterface::GetPropertiesWithFlag (wxArrayPGProperty * *targetArr*, wxPGProperty::FlagType *flags*, bool *inverse* = `false`, int *iterFlags* = (wxPG_ITERATE_PROPERTIES|wxPG_ITERATE_HIDDEN|wxPG_ITERATE_CATEGORIES)) const

Adds to 'targetArr' pointers to properties that have given flags 'flags' set.

However, if *'inverse'* is set to true, then only properties without given flags are stored.

Parameters

targetArr	

Todo docme

Parameters

flags	Property flags to use.
inverse	

docme

Parameters

iterFlags	Iterator flags to use. Default is everything expect private children. See wxPropertyGridIterator Flags.

wxPGProperty* wxPropertyGridInterface::GetProperty (const wxString & *name*) const

Returns pointer to a property with given name (case-sensitive).

If there is no property with such name, NULL pointer is returned.

Remarks

Properties which have non-category, non-root parent cannot be accessed globally by their name. Instead, use "<property>.<subproperty>" instead of "<subproperty>".

wxVariant wxPropertyGridInterface::GetPropertyAttribute (wxPGPropArg *id*, const wxString & *attrName*) const

Returns value of given attribute.

If none found, returns wxNullVariant.

wxColour wxPropertyGridInterface::GetPropertyBackgroundColour (wxPGPropArg *id*) const

Returns background colour of first cell of a property.

wxPGProperty* wxPropertyGridInterface::GetPropertyByLabel (const wxString & *label*) const

Returns first property which label matches given string.

NULL if none found. Note that this operation is very slow when compared to GetPropertyByName().

wxPGProperty* wxPropertyGridInterface::GetPropertyByName (const wxString & *name*) const

Returns pointer to a property with given name (case-sensitive).

If there is no property with such name, NULL pointer is returned.

Remarks

Properties which have non-category, non-root parent cannot be accessed globally by their name. Instead, use "<property>.<subproperty>" instead of "<subproperty>".

wxPGProperty* wxPropertyGridInterface::GetPropertyByName (const wxString & *name,* const wxString & *subname*) const

Returns child property 'subname' of property 'name'.

Same as calling GetPropertyByName("name.subname"), albeit slightly faster.

wxPropertyCategory* wxPropertyGridInterface::GetPropertyCategory (wxPGPropArg *id*) const

Returns pointer of property's nearest parent category.

If no category found, returns NULL.

void* wxPropertyGridInterface::GetPropertyClientData (wxPGPropArg *id*) const

Returns client data (void*) of a property.

const wxPGEditor* wxPropertyGridInterface::GetPropertyEditor (wxPGPropArg *id*) const

Returns property's editor.

wxString wxPropertyGridInterface::GetPropertyHelpString (wxPGPropArg *id*) const

Returns help string associated with a property.

wxBitmap* wxPropertyGridInterface::GetPropertyImage (wxPGPropArg *id*) const

Returns property's custom value image (NULL of none).

const wxString& wxPropertyGridInterface::GetPropertyLabel (wxPGPropArg *id*)

Returns label of a property.

wxString wxPropertyGridInterface::GetPropertyName (wxPGProperty * *property*)

Returns property's name, by which it is globally accessible.

wxColour wxPropertyGridInterface::GetPropertyTextColour (wxPGPropArg *id*) const

Returns text colour of first cell of a property.

wxValidator* wxPropertyGridInterface::GetPropertyValidator (wxPGPropArg *id*)

Returns validator of a property as a reference, which you can pass to any number of SetPropertyValidator.

wxVariant wxPropertyGridInterface::GetPropertyValue (wxPGPropArg *id*)

Returns property's value as wxVariant.

If property value is unspecified, Null variant is returned.

wxArrayInt wxPropertyGridInterface::GetPropertyValueAsArrayInt (wxPGPropArg *id*) const

Return's property's value as wxArrayInt.

wxArrayString wxPropertyGridInterface::GetPropertyValueAsArrayString (wxPGPropArg *id*) const

Returns property's value as wxArrayString.

bool wxPropertyGridInterface::GetPropertyValueAsBool (wxPGPropArg *id*) const

Returns property's value as bool.

wxDateTime wxPropertyGridInterface::GetPropertyValueAsDateTime (wxPGPropArg *id*) const

Return's property's value as wxDateTime.

double wxPropertyGridInterface::GetPropertyValueAsDouble (wxPGPropArg *id*) const

Returns property's value as double-precision floating point number.

int wxPropertyGridInterface::GetPropertyValueAsInt (wxPGPropArg *id*) const

Returns property's value as integer.

long wxPropertyGridInterface::GetPropertyValueAsLong (wxPGPropArg *id*) const

Returns property's value as integer.

wxLongLong_t wxPropertyGridInterface::GetPropertyValueAsLongLong (wxPGPropArg *id*) const

Returns property's value as native signed 64-bit integer.

wxString wxPropertyGridInterface::GetPropertyValueAsString (wxPGPropArg *id*) const

Returns property's value as wxString.

If property does not use string value type, then its value is converted using wxPGProperty::GetValueAsString().

unsigned long wxPropertyGridInterface::GetPropertyValueAsULong (wxPGPropArg *id*) const

Returns property's value as unsigned integer.

wxULongLong_t wxPropertyGridInterface::GetPropertyValueAsULongLong (wxPGPropArg *id*) const

Returns property's value as native unsigned 64-bit integer.

wxVariant wxPropertyGridInterface::GetPropertyValues (const wxString & *listname* = wxEmptyString, wxPGProperty * *baseparent* = NULL, long *flags* = 0) const

Returns a wxVariant list containing wxVariant versions of all property values.

Order is not guaranteed.

Parameters

listname	

Todo docme

Parameters

baseparent	

docme

Parameters

flags	Use wxPG_KEEP_STRUCTURE to retain category structure; each sub category will be its own wxVariantList of wxVariant.

```
Use wxPG_INC_ATTRIBUTES to include property attributes as well.
Each attribute will be stored as list variant named
"@@<propname>@@attr."
```

const wxArrayPGProperty& wxPropertyGridInterface::GetSelectedProperties ( ) const

Returns list of currently selected properties.

Remarks

wxArrayPGProperty should be compatible with std::vector API.

wxPGProperty* wxPropertyGridInterface::GetSelection ( ) const

Returns currently selected property.

NULL if none.

Remarks

When wxPG_EX_MULTIPLE_SELECTION extra style is used, this member function returns the focused property, that is the one which can have active editor.

virtual wxPGVIterator wxPropertyGridInterface::GetVIterator (int *flags*) const [virtual]

Similar to GetIterator(), but instead returns wxPGVIterator instance, which can be useful for forward-iterating through arbitrary property containers.

Parameters

flags	See wxPropertyGridIterator Flags.

Reimplemented in wxPropertyGridManager.

bool wxPropertyGridInterface::HideProperty (wxPGPropArg *id*, bool *hide* = true, int *flags* = wxPG_RECURSE)

Hides or reveals a property.

Parameters

id	

Todo docme

Parameters

hide	If true, hides property, otherwise reveals it.
flags	By default changes are applied recursively. Set this parameter wxPG_DONT_RECURSE to prevent this.

static void wxPropertyGridInterface::InitAllTypeHandlers ( ) [static]

Initializes *all* property types.

Causes references to most object files in the library, so calling this may cause significant increase in executable size when linking with static library.

wxPGProperty* wxPropertyGridInterface::Insert (wxPGPropArg *priorThis*, wxPGProperty * *newProperty*)

Inserts property to the property container.

Parameters

priorThis	New property is inserted just prior to this. Available only in the first variant. There are two versions of this function to allow this parameter to be either an id or name to a property.
newProperty	Pointer to the inserted property. wxPropertyGrid will take ownership of this object.

Returns

Returns newProperty.

Remarks

- wxPropertyGrid takes the ownership of the property pointer.
- While Append may be faster way to add items, make note that when both types of data storage (categoric and non-categoric) are active, Insert becomes even more slow. This is especially true if current mode is non-categoric.
- This functions deselects selected property, if any. Validation failure option wxPG_VFB_STAY_IN_PROPE↩ RTY is not respected, ie. selection is cleared even if editor had invalid value.

Example of use:

```
// append category
wxPGProperty* my_cat_id = propertygrid->Append( new wxPropertyCategory("My Category") );

...

// insert into category - using second variant
wxPGProperty* my_item_id_1 = propertygrid->Insert( my_cat_id, 0, new wxStringProperty("My
       String 1") );

// insert before to first item - using first variant
wxPGProperty* my_item_id_2 = propertygrid->Insert( my_item_id, new wxStringProperty("My String
       2") );
```

wxPGProperty* **wxPropertyGridInterface::Insert** (**wxPGPropArg** *parent,* **int** *index,* **wxPGProperty** * *newProperty*)

Inserts property to the property container.

See the other overload for more details.

Parameters

parent	New property is inserted under this category. Available only in the second variant. There are two versions of this function to allow this parameter to be either an id or name to a property.
index	Index under category. Available only in the second variant. If index is < 0, property is appended in category.
newProperty	Pointer to the inserted property. wxPropertyGrid will take ownership of this object.

Returns

Returns newProperty.

bool wxPropertyGridInterface::IsPropertyCategory (wxPGPropArg *id*) const

Returns true if property is a category.

bool wxPropertyGridInterface::IsPropertyEnabled (wxPGPropArg *id*) const

Returns true if property is enabled.

bool wxPropertyGridInterface::IsPropertyExpanded (wxPGPropArg *id*) const

Returns true if given property is expanded.

Naturally, always returns false for properties that cannot be expanded.

bool wxPropertyGridInterface::IsPropertyModified (wxPGPropArg *id*) const

Returns true if property has been modified after value set or modify flag clear by software.

virtual bool wxPropertyGridInterface::IsPropertySelected (wxPGPropArg *id*) const `[virtual]`

Returns true if property is selected.

Reimplemented in wxPropertyGridManager.

bool wxPropertyGridInterface::IsPropertyShown (wxPGPropArg *id*) const

Returns true if property is shown (ie.

HideProperty() with true not called for it).

bool wxPropertyGridInterface::IsPropertyValueUnspecified (wxPGPropArg *id*) const

Returns true if property value is set to unspecified.

void wxPropertyGridInterface::LimitPropertyEditing (wxPGPropArg *id*, bool *limit* = `true`)

Disables (limit = true) or enables (limit = false) wxTextCtrl editor of a property, if it is not the sole mean to edit the value.

static void wxPropertyGridInterface::RegisterAdditionalEditors ( ) [static]

Initializes additional property editors (SpinCtrl etc.).

Causes references to most object files in the library, so calling this may cause significant increase in executable size when linking with static library.

wxPGProperty* wxPropertyGridInterface::RemoveProperty (wxPGPropArg *id*)

Removes a property.

Does not delete the property object, but instead returns it.

Parameters

id	Pointer or name of a property.

Remarks

Removed property cannot have any children.

```
Also, if you remove property in a wxPropertyGrid event
handler, the actual removal is postponed until the next
idle event.
```

wxPGProperty* wxPropertyGridInterface::ReplaceProperty (wxPGPropArg *id,* wxPGProperty * *property*)

Replaces property with id with newly created one.

For example, this code replaces existing property named "Flags" with one that will have different set of items:

```
pg->ReplaceProperty("Flags",
    wxFlagsProperty("Flags", wxPG_LABEL, newItems))
```

See also

Insert()

bool wxPropertyGridInterface::RestoreEditableState (const wxString & *src,* int *restoreStates* = AllStates)

Restores user-editable state.

See also wxPropertyGridInterface::SaveEditableState().

Parameters

src	String generated by SaveEditableState.
restoreStates	Which parts to restore from source string. See list of editable state flags.

Returns

Returns false if there was problem reading the string.

Remarks

If some parts of state (such as scrolled or splitter position) fail to restore correctly, please make sure that you call this function after wxPropertyGrid size has been set (this may sometimes be tricky when sizers are used).

wxString wxPropertyGridInterface::SaveEditableState (int *includedStates* = AllStates) const

Used to acquire user-editable state (selected property, expanded properties, scrolled position, splitter positions).

Parameters

includedStates	Which parts of state to include. See list of editable state flags.

static void wxPropertyGridInterface::SetBoolChoices (const wxString & *trueChoice*, const wxString & *falseChoice*) [static]

Sets strings listed in the choice dropdown of a wxBoolProperty.

Defaults are "True" and "False", so changing them to, say, "Yes" and "No" may be useful in some less technical applications.

bool wxPropertyGridInterface::SetColumnProportion (unsigned int *column*, int *proportion*)

Set proportion of a auto-stretchable column.

wxPG_SPLITTER_AUTO_CENTER window style needs to be used to indicate that columns are auto- resizable.

Returns

Returns false on failure.

Remarks

You should call this for individual pages of wxPropertyGridManager (if used).

See also

GetColumnProportion()

void wxPropertyGridInterface::SetPropertyAttribute (wxPGPropArg *id*, const wxString & *attrName*, wxVariant *value*, long *argFlags* = 0)

Sets an attribute for this property.

Parameters

id	

Todo docme

Parameters

attrName	Text identifier of attribute. See wxPropertyGrid Property Attribute Identifiers.
value	Value of attribute.
argFlags	Optional. Use wxPG_RECURSE to set the attribute to child properties recursively.

Remarks

Setting attribute's value to Null variant will simply remove it from property's set of attributes.

void wxPropertyGridInterface::SetPropertyAttributeAll (const wxString & *attrName*, wxVariant *value*)

Sets property attribute for all applicapple properties.

Be sure to use this method only after all properties have been added to the grid.

void wxPropertyGridInterface::SetPropertyBackgroundColour (wxPGPropArg *id,* const wxColour & *colour,* int *flags =* `wxPG_RECURSE`)

Sets background colour of a property.

Parameters

id	Property name or pointer.
colour	New background colour.
flags	Default is wxPG_RECURSE which causes colour to be set recursively. Omit this flag to only set colour for the property in question and not any of its children.

void wxPropertyGridInterface::SetPropertyCell (wxPGPropArg *id*, int *column*, const wxString & *text* = wxEmptyString, const wxBitmap & *bitmap* = wxNullBitmap, const wxColour & *fgCol* = wxNullColour, const wxColour & *bgCol* = wxNullColour)

Sets text, bitmap, and colours for given column's cell.

Remarks

- You can set label cell by using column 0.
- You can use wxPG_LABEL as text to use default text for column.

void wxPropertyGridInterface::SetPropertyClientData (wxPGPropArg *id*, void * *clientData*)

Sets client data (void*) of a property.

Remarks

This untyped client data has to be deleted manually.

void wxPropertyGridInterface::SetPropertyColoursToDefault (wxPGPropArg *id*)

Resets text and background colours of given property.

void wxPropertyGridInterface::SetPropertyEditor (wxPGPropArg *id*, const wxPGEditor * *editor*)

Sets editor for a property.

Parameters

id	

Todo docme

Parameters

editor	For builtin editors, use wxPGEditor_X, where X is builtin editor's name (TextCtrl, Choice, etc. see wxPGEditor documentation for full list).

For custom editors, use pointer you received from wxPropertyGrid::RegisterEditorClass().

void wxPropertyGridInterface::SetPropertyEditor (wxPGPropArg *id*, const wxString & *editorName*)

Sets editor control of a property.

As editor argument, use editor name string, such as "TextCtrl" or "Choice".

void wxPropertyGridInterface::SetPropertyHelpString (wxPGPropArg *id,* const wxString & *helpString*)

Associates the help string with property.

Remarks

By default, text is shown either in the manager's "description" text box or in the status bar. If extra window style wxPG_EX_HELP_AS_TOOLTIPS is used, then the text will appear as a tooltip.

void wxPropertyGridInterface::SetPropertyImage (wxPGPropArg *id,* wxBitmap & *bmp*)

Set wxBitmap in front of the value.

Remarks

Bitmap will be scaled to a size returned by wxPropertyGrid::GetImageSize();

void wxPropertyGridInterface::SetPropertyLabel (wxPGPropArg *id,* const wxString & *newproplabel*)

Sets label of a property.

Remarks

- Properties under same parent may have same labels. However, property names must still remain unique.

bool wxPropertyGridInterface::SetPropertyMaxLength (wxPGPropArg *id,* int *maxLen*)

Sets max length of property's text.

void wxPropertyGridInterface::SetPropertyName (wxPGPropArg *id,* const wxString & *newName*)

Sets name of a property.

Parameters

id	Name or pointer of property which name to change.
newName	New name for property.

void wxPropertyGridInterface::SetPropertyReadOnly (wxPGPropArg *id,* bool *set* = `true`, int *flags* = `wxPG_RECURSE`)

Sets property (and, recursively, its children) to have read-only value.

In other words, user cannot change the value in the editor, but they can still copy it.

Parameters

id	Property name or pointer.
set	Use true to enable read-only, false to disable it.
flags	By default changes are applied recursively. Set this parameter wxPG_DONT_RECURSE to prevent this.

Remarks

This is mainly for use with textctrl editor. Only some other editors fully support it.

void wxPropertyGridInterface::SetPropertyTextColour (wxPGPropArg *id,* const wxColour & *colour,* int *flags =* `wxPG_RECURSE`)

Sets text colour of a property.

Parameters

id	Property name or pointer.
colour	New background colour.
flags	Default is wxPG_RECURSE which causes colour to be set recursively. Omit this flag to only set colour for the property in question and not any of its children.

void wxPropertyGridInterface::SetPropertyValidator (wxPGPropArg *id,* const wxValidator & *validator*)

Sets validator of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* long *value*)

Sets value (integer) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* int *value*)

Sets value (integer) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* double *value*)

Sets value (floating point) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* bool *value*)

Sets value (bool) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* const wxString & *value*)

Sets value (string) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* const wxArrayString & *value*)

Sets value (wxArrayString) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* const wxDateTime & *value*)

Sets value (wxDateTime) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* wxObject * *value*)

Sets value (wxObject*) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* wxObject & *value*)

Sets value (wxObject&) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* wxLongLong_t *value*)

Sets value (native 64-bit int) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* wxULongLong_t *value*)

Sets value (native 64-bit unsigned int) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* const wxArrayInt & *value*)

Sets value (wxArrayInt&) of a property.

void wxPropertyGridInterface::SetPropertyValue (wxPGPropArg *id,* wxVariant *value*)

Sets value (wxVariant&) of a property.

Remarks

Use wxPropertyGrid::ChangePropertyValue() instead if you need to run through validation process and send property change event.

void wxPropertyGridInterface::SetPropertyValues (const wxVariantList & *list,* wxPGPropArg *defaultCategory* = wxNullProperty)

Sets property values from a list of wxVariants.

void wxPropertyGridInterface::SetPropertyValues (const wxVariant & *list,* wxPGPropArg *defaultCategory* = wxNullProperty)

Sets property values from a list of wxVariants.

void wxPropertyGridInterface::SetPropertyValueString (wxPGPropArg *id,* const wxString & *value*)

Sets value (wxString) of a property.

Remarks

This method uses wxPGProperty::SetValueFromString(), which all properties should implement. This means that there should not be a type error, and instead the string is converted to property's actual value type.

void wxPropertyGridInterface::SetPropertyValueUnspecified (wxPGPropArg *id*)

Sets property's value to unspecified.

If it has children (it may be category), then the same thing is done to them.

void wxPropertyGridInterface::SetValidationFailureBehavior (int *vfbFlags*)

Adjusts how wxPropertyGrid behaves when invalid value is entered in a property.

Parameters

vfbFlags	See wxPropertyGrid Validation Failure behaviour Flags for possible values.

void wxPropertyGridInterface::Sort (int *flags* = 0)

Sorts all properties recursively.

Parameters

flags	This can contain any of the following options: wxPG_SORT_TOP_LEVEL_ONLY: Only sort categories and their immediate children. Sorting done by wxPG_AUTO_SORT option uses this.

See also

SortChildren, wxPropertyGrid::SetSortFunction

void wxPropertyGridInterface::SortChildren (wxPGPropArg *id*, int *flags* = 0)

Sorts children of a property.

Parameters

id	Name or pointer to a property.
flags	This can contain any of the following options: wxPG_RECURSE: Sorts recursively.

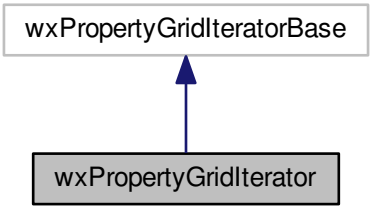
See also

Sort, wxPropertyGrid::SetSortFunction

21.565 wxPropertyGridIterator Class Reference

`#include <wx/propgrid/propgridpagestate.h>`

Inheritance diagram for wxPropertyGridIterator:



21.565.1 Detailed Description

21.565.2 wxPropertyGridIterator

Preferable way to iterate through contents of wxPropertyGrid, wxPropertyGridManager, and wxPropertyGridPage.

See wxPropertyGridInterface::GetIterator() for more information about usage.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- void Assign (const wxPropertyGridIteratorBase &it)
- bool AtEnd () const
- wxPGProperty * GetProperty () const

  Get current property.
- void Next (bool iterateChildren=true)

  Iterate to the next property.
- void Prev ()

  Iterate to the previous property.

21.565.3 Member Function Documentation

void wxPropertyGridIterator::Assign (const wxPropertyGridIteratorBase & *it*)

bool wxPropertyGridIterator::AtEnd ( ) const `[inline]`

wxPGProperty* wxPropertyGridIterator::GetProperty ( ) const `[inline]`

Get current property.

void wxPropertyGridIterator::Next (bool *iterateChildren* = `true`)

Iterate to the next property.

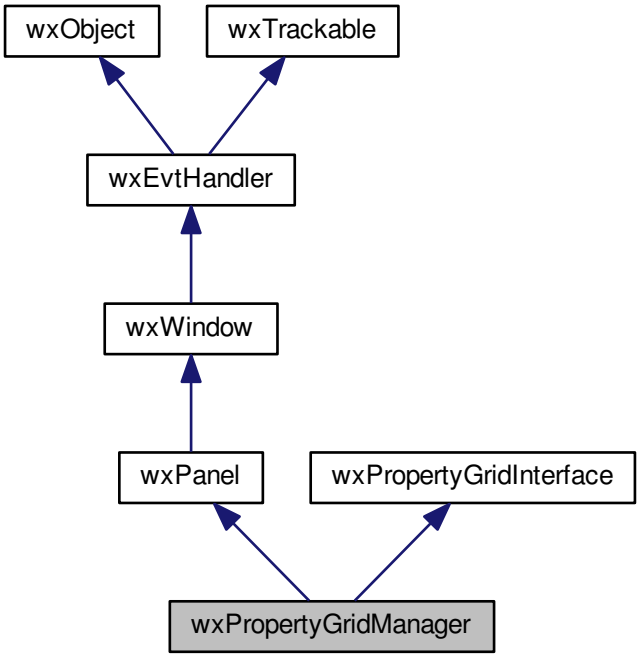
void wxPropertyGridIterator::Prev ( )

Iterate to the previous property.

21.566 wxPropertyGridManager Class Reference

`#include <wx/propgrid/manager.h>`

Inheritance diagram for wxPropertyGridManager:



21.566.1 Detailed Description

wxPropertyGridManager is an efficient multi-page version of wxPropertyGrid, which can optionally have toolbar for mode and page selection, a help text box, and a header.

wxPropertyGridManager inherits from wxPropertyGridInterface, and as such it has most property manipulation functions. However, only some of them affect properties on all pages (eg. GetPropertyByName() and ExpandAll()), while some (eg. Append()) only apply to the currently selected page.

To operate explicitly on properties on specific page, use wxPropertyGridManager::GetPage() to obtain pointer to page's wxPropertyGridPage object.

Visual methods, such as SetCellBackgroundColour() are only available in wxPropertyGrid. Use wxPropertyGrid↩ Manager::GetGrid() to obtain pointer to it.

Non-virtual iterators will not work in wxPropertyGridManager. Instead, you must acquire the internal grid (GetGrid()) or wxPropertyGridPage object (GetPage()).

wxPropertyGridManager constructor has exact same format as wxPropertyGrid constructor, and basically accepts same extra window style flags (albeit also has some extra ones).

Here's some example code for creating and populating a wxPropertyGridManager:

```
wxPropertyGridManager* pgMan = new wxPropertyGridManager(this,
      PGID,
    wxDefaultPosition, wxDefaultSize,
    // These and other similar styles are automatically
    // passed to the embedded wxPropertyGrid.
    wxPG_BOLD_MODIFIED|wxPG_SPLITTER_AUTO_CENTER|
    // Include toolbar.
    wxPG_TOOLBAR |
    // Include description box.
    wxPG_DESCRIPTION |
```

```
    // Plus defaults.
    wxPGMAN_DEFAULT_STYLE
   );

wxPropertyGridPage* page;

page = pgMan->AddPage("First Page");

page->Append( new wxPropertyCategory("Category A1") );

page->Append( new wxIntProperty("Number",wxPG_LABEL,1) );

page->Append( new wxColourProperty("Colour",wxPG_LABEL,*wxWHITE) );

page = pgMan->AddPage("Second Page");

page->Append( "Text",wxPG_LABEL,"(no text)" );

page->Append( new wxFontProperty("Font",wxPG_LABEL) );

// Display a header above the grid
pgMan->ShowHeader();
```

21.566.2 Window Styles

See wxPropertyGrid Window Styles.

21.566.3 Event Handling

See wxPropertyGrid Event Handling for more information.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPropertyGridPage * AddPage (const wxString &label=wxEmptyString, const wxBitmap &bmp=wxPG_NULL_BITMAP, wxPropertyGridPage *pageObj=NULL)

  Creates new property page.
- virtual void Clear ()

  Deletes all properties and all pages.
- void ClearPage (int page)

  Deletes all properties on given page.
- bool CommitChangesFromEditor (wxUint32 flags=0)

  Forces updating the value of property from the editor control.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxPGMAN_DEFAULT_STYLE, const wxString &name=wxPropertyGridManagerNameStr)

  Two step creation.
- bool EnableCategories (bool enable)

  Enables or disables (shows/hides) categories according to parameter enable.
- bool EnsureVisible (wxPGPropArg id)

  Selects page, scrolls and/or expands items to ensure that the given item is visible.
- int GetColumnCount (int page=-1) const

  Returns number of columns on given page.
- int GetDescBoxHeight () const

  Returns height of the description text box.

- wxPropertyGrid * GetGrid ()

  Returns pointer to the contained wxPropertyGrid.
- virtual wxPGVIterator GetVIterator (int flags) const

  Similar to GetIterator, but instead returns wxPGVIterator instance, which can be useful for forward-iterating through arbitrary property containers.
- wxPropertyGridPage * GetCurrentPage () const

  Returns currently selected page.
- wxPropertyGridPage * GetPage (unsigned int ind) const

  Returns page object for given page index.
- wxPropertyGridPage * GetPage (const wxString &name) const

  Returns page object for given page name.
- int GetPageByName (const wxString &name) const

  Returns index for a page name.
- size_t GetPageCount () const

  Returns number of managed pages.
- const wxString & GetPageName (int index) const

  Returns name of given page.
- wxPGProperty * GetPageRoot (int index) const

  Returns "root property" of the given page.
- int GetSelectedPage () const

  Returns index to currently selected page.
- wxPGProperty * GetSelectedProperty () const

  Alias for GetSelection().
- wxPGProperty * GetSelection () const

  Shortcut for GetGrid()->GetSelection().
- wxToolBar * GetToolBar () const

  Returns a pointer to the toolbar currently associated with the wxPropertyGridManager (if any).
- virtual wxPropertyGridPage * InsertPage (int index, const wxString &label, const wxBitmap &bmp=wxNull↩ Bitmap, wxPropertyGridPage *pageObj=NULL)

  Creates new property page.
- bool IsAnyModified () const

  Returns true if any property on any page has been modified by the user.
- bool IsFrozen () const

  Returns true if updating is frozen (ie.
- bool IsPageModified (size_t index) const

  Returns true if any property on given page has been modified by the user.
- virtual bool IsPropertySelected (wxPGPropArg id) const

  Returns true if property is selected.
- virtual bool RemovePage (int page)

  Removes a page.
- void SelectPage (int index)

  Select and displays a given page.
- void SelectPage (const wxString &label)

  Select and displays a given page (by label).
- void SelectPage (wxPropertyGridPage *page)

  Select and displays a given page.
- bool SelectProperty (wxPGPropArg id, bool focus=false)

  Select a property.
- void SetColumnCount (int colCount, int page=-1)

  Sets number of columns on given page (default is current page).
- void SetColumnTitle (int idx, const wxString &title)

Sets a column title.

- void SetDescription (const wxString &label, const wxString &content)

  Sets label and text in description box.
- void SetDescBoxHeight (int ht, bool refresh=true)

  Sets y coordinate of the description box splitter.
- void SetSplitterLeft (bool subProps=false, bool allPages=true)

  Moves splitter as left as possible, while still allowing all labels to be shown in full.
- void SetPageSplitterLeft (int page, bool subProps=false)

  Moves splitter as left as possible on an individual page, while still allowing all labels to be shown in full.
- void SetPageSplitterPosition (int page, int pos, int column=0)

  Sets splitter position on individual page.
- void SetSplitterPosition (int pos, int column=0)

  Sets splitter position for all pages.
- void ShowHeader (bool show=true)

  Show or hide the property grid header control.

Protected Member Functions

- virtual wxPropertyGrid * CreatePropertyGrid () const

  Creates property grid for the manager.

Additional Inherited Members

21.566.4 Member Function Documentation

wxPropertyGridPage* wxPropertyGridManager::AddPage (const wxString & *label* = wxEmptyString, const wxBitmap & *bmp* = `wxPG_NULL_BITMAP`, wxPropertyGridPage * *pageObj* = `NULL`)

Creates new property page.

Note that the first page is not created automatically.

Parameters

label	A label for the page. This may be shown as a toolbar tooltip etc.
bmp	Bitmap image for toolbar. If wxNullBitmap is used, then a built-in default image is used.
pageObj	wxPropertyGridPage instance. Manager will take ownership of this object. NULL indicates that a default page instance should be created.

Returns

Returns pointer to created property grid page.

Remarks

If toolbar is used, it is highly recommended that the pages are added when the toolbar is not turned off using window style flag switching. Otherwise toolbar buttons might not be added properly.

virtual void wxPropertyGridManager::Clear ( ) `[virtual]`

Deletes all properties and all pages.

Implements wxPropertyGridInterface.

void wxPropertyGridManager::ClearPage (int *page*)

Deletes all properties on given page.

bool wxPropertyGridManager::CommitChangesFromEditor (wxUint32 *flags* = 0)

Forces updating the value of property from the editor control.

Returns

Returns true if value was actually updated.

bool wxPropertyGridManager::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxPGMAN_DEFAULT_STYLE, const wxString & *name* = wxPropertyGridManagerNameStr)

Two step creation.

Whenever the control is created without any parameters, use Create to actually create it. Don't access the control's public methods before this is called.

See also

wxPropertyGrid Window Styles

virtual wxPropertyGrid* wxPropertyGridManager::CreatePropertyGrid ( ) const [protected], [virtual]

Creates property grid for the manager.

Reimplement in derived class to use subclassed wxPropertyGrid. However, if you do this then you must also use the two-step construction (ie. default constructor and Create() instead of constructor with arguments) when creating the manager.

bool wxPropertyGridManager::EnableCategories (bool *enable*)

Enables or disables (shows/hides) categories according to parameter enable.

Remarks

Calling his may not properly update toolbar buttons.

bool wxPropertyGridManager::EnsureVisible (wxPGPropArg *id*)

Selects page, scrolls and/or expands items to ensure that the given item is visible.

Returns

Returns true if something was actually done.

int wxPropertyGridManager::GetColumnCount (int *page* = -1) const

Returns number of columns on given page.

By the default, returns number of columns on current page.

wxPropertyGridPage* wxPropertyGridManager::GetCurrentPage ( ) const

Returns currently selected page.

int wxPropertyGridManager::GetDescBoxHeight ( ) const

Returns height of the description text box.

wxPropertyGrid* wxPropertyGridManager::GetGrid ( )

Returns pointer to the contained wxPropertyGrid.

This does not change after wxPropertyGridManager has been created, so you can safely obtain pointer once and use it for the entire lifetime of the manager instance.

wxPropertyGridPage* wxPropertyGridManager::GetPage (unsigned int *ind*) const

Returns page object for given page index.

wxPropertyGridPage* wxPropertyGridManager::GetPage (const wxString & *name*) const

Returns page object for given page name.

int wxPropertyGridManager::GetPageByName (const wxString & *name*) const

Returns index for a page name.

If no match is found, wxNOT_FOUND is returned.

size_t wxPropertyGridManager::GetPageCount ( ) const

Returns number of managed pages.

const wxString& wxPropertyGridManager::GetPageName (int *index*) const

Returns name of given page.

wxPGProperty* wxPropertyGridManager::GetPageRoot (int *index*) const

Returns "root property" of the given page.

It does not have name, etc. and it is not visible. It is only useful for accessing its children.

int wxPropertyGridManager::GetSelectedPage ( ) const

Returns index to currently selected page.

wxPGProperty* wxPropertyGridManager::GetSelectedProperty ( ) const

Alias for GetSelection().

wxPGProperty* wxPropertyGridManager::GetSelection ( ) const

Shortcut for GetGrid()->GetSelection().

wxToolBar* wxPropertyGridManager::GetToolBar ( ) const

Returns a pointer to the toolbar currently associated with the wxPropertyGridManager (if any).

virtual wxPGVIterator wxPropertyGridManager::GetVIterator (int *flags*) const `[virtual]`

Similar to GetIterator, but instead returns wxPGVIterator instance, which can be useful for forward-iterating through arbitrary property containers.

Reimplemented from wxPropertyGridInterface.

virtual wxPropertyGridPage* wxPropertyGridManager::InsertPage (int *index*, const wxString & *label*, const wxBitmap & *bmp* = wxNullBitmap, wxPropertyGridPage * *pageObj* = NULL) `[virtual]`

Creates new property page.

Note that the first page is not created automatically.

Parameters

index	Add to this position. -1 will add as the last item.
label	A label for the page. This may be shown as a toolbar tooltip etc.
bmp	Bitmap image for toolbar. If wxNullBitmap is used, then a built-in default image is used.
pageObj	wxPropertyGridPage instance. Manager will take ownership of this object. If NULL, default page object is constructed.

Returns

Returns pointer to created page.

bool wxPropertyGridManager::IsAnyModified ( ) const

Returns true if any property on any page has been modified by the user.

bool wxPropertyGridManager::IsFrozen ( ) const

Returns true if updating is frozen (ie.

Freeze() called but not yet Thaw()).

bool wxPropertyGridManager::IsPageModified (size_t *index*) const

Returns true if any property on given page has been modified by the user.

virtual bool wxPropertyGridManager::IsPropertySelected (wxPGPropArg *id*) const `[virtual]`

Returns true if property is selected.

Since selection is page based, this function checks every page in the manager.

Reimplemented from wxPropertyGridInterface.

virtual bool wxPropertyGridManager::RemovePage (int *page*) [virtual]

Removes a page.

Returns

Returns false if it was not possible to remove page in question.

void wxPropertyGridManager::SelectPage (int *index*)

Select and displays a given page.

Parameters

index	Index of page being seleced. Can be -1 to select nothing.

void wxPropertyGridManager::SelectPage (const wxString & *label*)

Select and displays a given page (by label).

void wxPropertyGridManager::SelectPage (wxPropertyGridPage * *page*)

Select and displays a given page.

bool wxPropertyGridManager::SelectProperty (wxPGPropArg *id,* bool *focus* = false)

Select a property.

See also

wxPropertyGrid::SelectProperty(), wxPropertyGridInterface::ClearSelection()

void wxPropertyGridManager::SetColumnCount (int *colCount,* int *page* = -1)

Sets number of columns on given page (default is current page).

Remarks

If you use header, then you should always use this member function to set the column count, instead of ones present in wxPropertyGrid or wxPropertyGridPage.

void wxPropertyGridManager::SetColumnTitle (int *idx,* const wxString & *title*)

Sets a column title.

Default title for column 0 is "Property", and "Value" for column 1.

Remarks

If header is not shown yet, then calling this member function will make it visible.

void wxPropertyGridManager::SetDescBoxHeight (int *ht,* bool *refresh* = true)

Sets y coordinate of the description box splitter.

void wxPropertyGridManager::SetDescription (const wxString & *label,* const wxString & *content*)

Sets label and text in description box.

void wxPropertyGridManager::SetPageSplitterLeft (int *page,* bool *subProps* = `false`)

Moves splitter as left as possible on an individual page, while still allowing all labels to be shown in full.

void wxPropertyGridManager::SetPageSplitterPosition (int *page,* int *pos,* int *column* = `0`)

Sets splitter position on individual page.

Remarks

If you use header, then you should always use this member function to set the splitter position, instead of ones present in wxPropertyGrid or wxPropertyGridPage.

void wxPropertyGridManager::SetSplitterLeft (bool *subProps* = `false`, bool *allPages* = `true`)

Moves splitter as left as possible, while still allowing all labels to be shown in full.

Parameters

subProps	If false, will still allow sub-properties (ie. properties which parent is not root or category) to be cropped.
allPages	If true, takes labels on all pages into account.

void wxPropertyGridManager::SetSplitterPosition (int *pos,* int *column* = `0`)

Sets splitter position for all pages.

Remarks

Splitter position cannot exceed grid size, and therefore setting it during form creation may fail as initial grid size is often smaller than desired splitter position, especially when sizers are being used.

If you use header, then you should always use this member function to set the splitter position, instead of ones present in wxPropertyGrid or wxPropertyGridPage.

void wxPropertyGridManager::ShowHeader (bool *show* = `true`)

Show or hide the property grid header control.

It is hidden by the default.

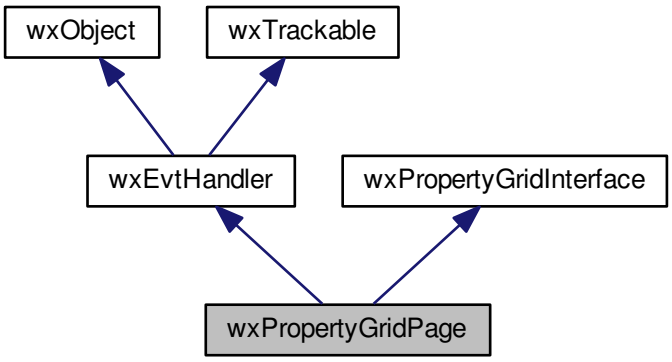
Remarks

Grid may look better if you use wxPG_NO_INTERNAL_BORDER window style when showing a header.

21.567 wxPropertyGridPage Class Reference

`#include <wx/propgrid/manager.h>`

Inheritance diagram for wxPropertyGridPage:



21.567.1 Detailed Description

Holder of property grid page information.

You can subclass this and give instance in wxPropertyGridManager::AddPage. It inherits from wxEvtHandler and can be used to process events specific to this page (id of events will still be same as manager's). If you don't want to use it to process all events of the page, you need to return false in the derived wxPropertyGridPage::IsHandling↩ AllEvents.

Please note that wxPropertyGridPage lacks many non-const property manipulation functions found in wxProperty↩ GridManager. Please use parent manager (m_manager member variable) when needed.

Please note that most member functions are inherited and as such not documented on this page. This means you will probably also want to read wxPropertyGridInterface class reference.

21.567.2 Event Handling

wxPropertyGridPage receives events emitted by its wxPropertyGridManager, but only those events that are specific to that page. If wxPropertyGridPage:: IsHandlingAllEvents returns false, then unhandled events are sent to the manager's parent, as usual.

See wxPropertyGrid Event Handling for more information.

Library: wxPropertyGrid

Category: wxPropertyGrid

Public Member Functions

- wxPropertyGridPage ()
- virtual ~wxPropertyGridPage ()
- virtual void Clear ()

  Deletes all properties on page.

- wxSize FitColumns ()

  Reduces column sizes to minimum possible that contents are still visibly (naturally some margin space will be applied as well).
- int GetIndex () const

  Returns page index in manager;.
- wxPGProperty * GetRoot () const

  Returns "root property".
- int GetSplitterPosition (int col=0) const

  Returns x-coordinate position of splitter on a page.
- int GetToolId () const

  Returns id of the tool bar item that represents this page on wxPropertyGridManager's wxToolBar.
- virtual void Init ()

  Do any member initialization in this method.
- virtual bool IsHandlingAllEvents () const

  Return false here to indicate unhandled events should be propagated to manager's parent, as normal.
- virtual void OnShow ()

  Called every time page is about to be shown.
- virtual void RefreshProperty (wxPGProperty *p)

  Refreshes given property on page.
- void SetSplitterPosition (int splitterPos, int col=0)

  Sets splitter position on page.

Friends

- class wxPropertyGridManager

Additional Inherited Members

21.567.3 Constructor & Destructor Documentation

wxPropertyGridPage::wxPropertyGridPage ( )

virtual wxPropertyGridPage::~wxPropertyGridPage ( ) [virtual]

21.567.4 Member Function Documentation

virtual void wxPropertyGridPage::Clear ( ) [virtual]

Deletes all properties on page.

Implements wxPropertyGridInterface.

wxSize wxPropertyGridPage::FitColumns ( )

Reduces column sizes to minimum possible that contents are still visibly (naturally some margin space will be applied as well).

Returns

Returns minimum size for the page to still display everything.

Remarks

This function only works properly if size of containing grid was already fairly large.

Note that you can also get calculated column widths by calling GetColumnWidth() immediately after this function returns.

int wxPropertyGridPage::GetIndex ( ) const `[inline]`

Returns page index in manager;.

wxPGProperty* wxPropertyGridPage::GetRoot ( ) const

Returns "root property".

It does not have name, etc. and it is not visible. It is only useful for accessing its children.

int wxPropertyGridPage::GetSplitterPosition (int *col* = 0) const

Returns x-coordinate position of splitter on a page.

int wxPropertyGridPage::GetToolId ( ) const

Returns id of the tool bar item that represents this page on wxPropertyGridManager's wxToolBar.

virtual void wxPropertyGridPage::Init ( ) `[virtual]`

Do any member initialization in this method.

Remarks

- Called every time the page is added into a manager.
  - You can add properties to the page here.

virtual bool wxPropertyGridPage::IsHandlingAllEvents ( ) const `[virtual]`

Return false here to indicate unhandled events should be propagated to manager's parent, as normal.

virtual void wxPropertyGridPage::OnShow ( ) `[virtual]`

Called every time page is about to be shown.

Useful, for instance, creating properties just-in-time.

virtual void wxPropertyGridPage::RefreshProperty (wxPGProperty * *p*) `[virtual]`

Refreshes given property on page.

void wxPropertyGridPage::SetSplitterPosition (int *splitterPos*, int *col* = 0)

Sets splitter position on page.

Remarks

Splitter position cannot exceed grid size, and therefore setting it during form creation may fail as initial grid size is often smaller than desired splitter position, especially when sizers are being used.

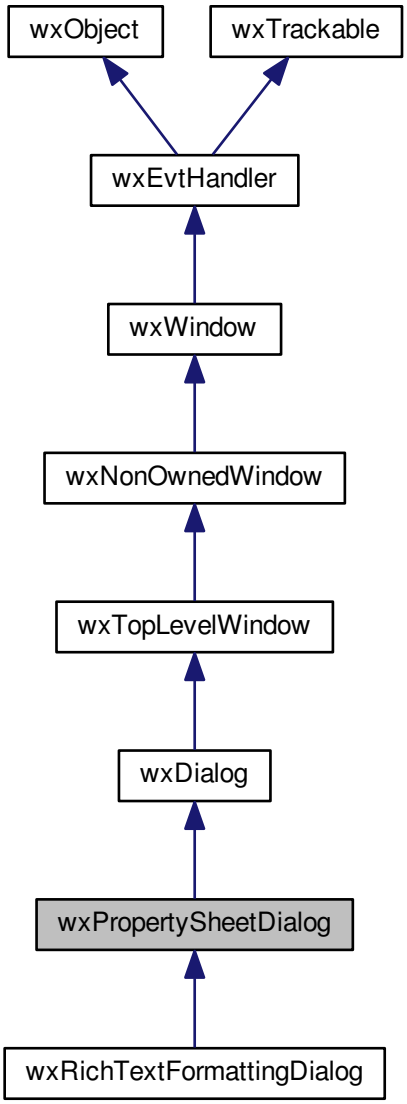
21.567.5 Friends And Related Function Documentation

friend class wxPropertyGridManager `[friend]`

21.568 wxPropertySheetDialog Class Reference

`#include <wx/propdlg.h>`

Inheritance diagram for wxPropertySheetDialog:



21.568.1 Detailed Description

This class represents a property sheet dialog: a tabbed dialog for showing settings.

It is optimized to show flat tabs on PocketPC devices, and can be customized to use different controllers instead of the default notebook style.

To use this class, call Create() from your own Create function. Then call CreateButtons(), and create pages, adding them to the book control. Finally call LayoutDialog().

For example:

```
bool MyPropertySheetDialog::Create(...)
{
    if (!wxPropertySheetDialog::Create(...))
        return false;

    CreateButtons(wxOK|wxCANCEL|wxHELP);

    // Add page
    wxPanel* panel = new wxPanel(GetBookCtrl(), ...);
    GetBookCtrl()->AddPage(panel, "General");

    LayoutDialog();
    return true;
}
```

If necessary, override CreateBookCtrl() and AddBookCtrl() to create and add a different kind of book control. You will then need to use two-step construction for the dialog or change the style of the book control by calling SetSheetStyle() before calling Create().

The Dialogs Sample shows this class being used with notebook and toolbook controllers (for Windows-style and Mac-style settings dialogs).

To make pages of the dialog scroll when the display is too small to fit the whole dialog, you can switch layout adaptation on globally with wxDialog::EnableLayoutAdaptation() or per dialog with wxDialog::SetLayoutAdaptationMode().

For more about layout adaptation, see Automatic Scrolled Dialogs.

Library: wxAdvanced

Category: Managed Windows

Public Member Functions

- wxPropertySheetDialog (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE, const wxString &name=wxDialogNameStr)

  Constructor.
- virtual void AddBookCtrl (wxSizer *sizer)

  Override this if you wish to add the book control in a way different from the standard way (for example, using different spacing).
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE, const wxString &name=wxDialogNameStr)

  Call this from your own Create function, before adding buttons and pages.
- virtual wxBookCtrlBase * CreateBookCtrl ()

  Override this if you wish to create a different kind of book control; by default, the value passed to SetSheetStyle() is used to determine the control.
- virtual void CreateButtons (int flags=wxOK|wxCANCEL)

Call this to create the buttons for the dialog.

- wxBookCtrlBase * GetBookCtrl () const

  Returns the book control that will contain your settings pages.
- wxSizer * GetInnerSizer () const

  Returns the inner sizer that contains the book control and button sizer.
- long GetSheetStyle () const

  Returns the sheet style.
- virtual void LayoutDialog (int centreFlags=wxBOTH)

  Call this to lay out the dialog.
- void SetBookCtrl (wxBookCtrlBase *bookCtrl)

  Sets the book control used for the dialog.
- void SetSheetStyle (long style)

  You can customize the look and feel of the dialog by setting the sheet style.

Additional Inherited Members

21.568.2 Constructor & Destructor Documentation

wxPropertySheetDialog::wxPropertySheetDialog (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE, const wxString & *name* = `wxDialogNameStr`)

Constructor.

21.568.3 Member Function Documentation

virtual void wxPropertySheetDialog::AddBookCtrl (wxSizer * *sizer*) `[virtual]`

Override this if you wish to add the book control in a way different from the standard way (for example, using different spacing).

bool wxPropertySheetDialog::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE, const wxString & *name* = `wxDialogNameStr`)

Call this from your own Create function, before adding buttons and pages.

virtual wxBookCtrlBase* wxPropertySheetDialog::CreateBookCtrl ( ) `[virtual]`

Override this if you wish to create a different kind of book control; by default, the value passed to SetSheetStyle() is used to determine the control.

The default behaviour is to create a notebook except on Smartphone, where a choicebook is used.

virtual void wxPropertySheetDialog::CreateButtons (int *flags* = wxOK|wxCANCEL) `[virtual]`

Call this to create the buttons for the dialog.

This calls wxDialog::CreateButtonSizer(), and the flags are the same.

Note

On PocketPC, no buttons are created.

wxBookCtrlBase* wxPropertySheetDialog::GetBookCtrl ( ) const

Returns the book control that will contain your settings pages.

wxSizer* wxPropertySheetDialog::GetInnerSizer ( ) const

Returns the inner sizer that contains the book control and button sizer.

long wxPropertySheetDialog::GetSheetStyle ( ) const

Returns the sheet style.

See SetSheetStyle() for allowed values.

virtual void wxPropertySheetDialog::LayoutDialog (int *centreFlags* = wxBOTH) [virtual]

Call this to lay out the dialog.

Note

On PocketPC, this does nothing, since the dialog will be shown full-screen, and the layout will be done when the dialog receives a size event.

void wxPropertySheetDialog::SetBookCtrl (wxBookCtrlBase * *bookCtrl*)

Sets the book control used for the dialog.

You will normally not need to use this.

void wxPropertySheetDialog::SetSheetStyle (long *style*)

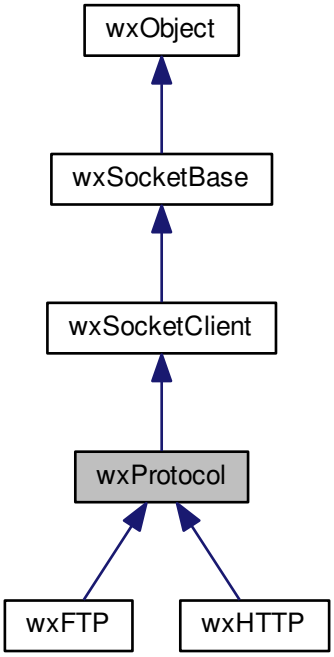
You can customize the look and feel of the dialog by setting the sheet style.

It is a bit list of the wxPropertySheetDialogFlags values.

21.569 wxProtocol Class Reference

```
#include <wx/protocol/protocol.h>
```

Inheritance diagram for wxProtocol:



21.569.1 Detailed Description

Represents a protocol for use with wxURL.

Note that you may want to change the default time-out for HTTP/FTP connections and network operations (using SetDefaultTimeout()) since the default time-out value is quite long (60 seconds).

Library: wxNet

Category: Networking

See also

wxSocketBase, wxURL

Public Member Functions

- virtual bool Abort ()=0

  Abort the current stream.
- virtual wxString GetContentType () const

  Returns the type of the content of the last opened stream.
- virtual wxProtocolError GetError () const

  Returns the last occurred error.

- virtual wxInputStream * GetInputStream (const wxString &path)=0

  Creates a new input stream on the specified path.
- bool Reconnect ()

  Tries to reestablish a previous opened connection (close and renegotiate connection).
- virtual void SetPassword (const wxString &user)

  Sets the authentication password.
- virtual void SetUser (const wxString &user)

  Sets the authentication user.
- void SetDefaultTimeout (wxUint32 Value)

  Sets a new default timeout for the network operations.

Logging support.

Each wxProtocol object may have the associated logger (by default there is none) which is used to log network requests and responses.

See also

wxProtocolLog

- void SetLog (wxProtocolLog *log)

  Set the logger, deleting the old one and taking ownership of this one.
- wxProtocolLog * GetLog () const

  Return the current logger, may be NULL.
- wxProtocolLog * DetachLog ()

  Detach the existing logger without deleting it.
- void LogRequest (const wxString &str)

  Call wxProtocolLog::LogRequest() if we have a valid logger or do nothing otherwise.
- void LogResponse (const wxString &str)

  Call wxProtocolLog::LogResponse() if we have a valid logger or do nothing otherwise.

Additional Inherited Members

21.569.2 Member Function Documentation

virtual bool wxProtocol::Abort ( ) [pure virtual]

Abort the current stream.

Warning

It is advised to destroy the input stream instead of aborting the stream this way.

Returns

Returns true, if successful, else false.

Implemented in wxFTP.

wxProtocolLog* wxProtocol::DetachLog ( )

Detach the existing logger without deleting it.

The caller is responsible for deleting the returned pointer if it's non-NULL.

virtual **wxString** wxProtocol::GetContentType ( ) const [virtual]

Returns the type of the content of the last opened stream.

It is a mime-type. May be an empty string if the content-type is unknown.

virtual **wxProtocolError** wxProtocol::GetError ( ) const [virtual]

Returns the last occurred error.

See also

wxProtocolError

virtual **wxInputStream*** wxProtocol::GetInputStream (const **wxString** & *path*) [pure virtual]

Creates a new input stream on the specified path.

You can use all but seek() functionality of wxStream. Seek() isn't available on all streams. For example, HTTP or FTP streams don't deal with it. Other functions like StreamSize() and Tell() aren't available for the moment for this sort of stream. You will be notified when the EOF is reached by an error.

Returns

Returns the initialized stream. You will have to delete it yourself once you don't use it anymore. The destructor closes the network connection.

See also

wxInputStream

Implemented in wxFTP, and wxHTTP.

wxProtocolLog* wxProtocol::GetLog ( ) const [inline]

Return the current logger, may be NULL.

void wxProtocol::LogRequest (const **wxString** & *str*)

Call wxProtocolLog::LogRequest() if we have a valid logger or do nothing otherwise.

void wxProtocol::LogResponse (const **wxString** & *str*)

Call wxProtocolLog::LogResponse() if we have a valid logger or do nothing otherwise.

bool wxProtocol::Reconnect ( )

Tries to reestablish a previous opened connection (close and renegotiate connection).

Returns

true, if the connection is established, else false.

void wxProtocol::SetDefaultTimeout (wxUint32 *Value*)

Sets a new default timeout for the network operations.

The default timeout is 60 seconds.

See also

wxSocketBase::SetTimeout

void wxProtocol::SetLog (wxProtocolLog * *log*)

Set the logger, deleting the old one and taking ownership of this one.

Parameters

log	New logger allocated on the heap or NULL.

virtual void wxProtocol::SetPassword (const wxString & *user*) `[virtual]`

Sets the authentication password.

Reimplemented in wxFTP.

virtual void wxProtocol::SetUser (const wxString & *user*) `[virtual]`

Sets the authentication user.

Reimplemented in wxFTP.

21.570 wxProtocolLog Class Reference

```
#include <wx/protocol/log.h>
```

21.570.1 Detailed Description

Class allowing to log network operations performed by wxProtocol.

Library: wxNet

Category: Networking

See also

wxProtocol

Public Member Functions

- wxProtocolLog (const wxString &traceMask)

  Create object doing the logging using wxLogTrace() with the specified trace mask.
- virtual void LogRequest (const wxString &str)

  Called by wxProtocol-derived objects to log strings sent to the server.

- virtual void LogResponse (const wxString &str)

  Called by wxProtocol-derived objects to log strings received from the server.

Protected Member Functions

- virtual void DoLogString (const wxString &str)

  Log the given string.

21.570.2 Constructor & Destructor Documentation

wxProtocolLog::wxProtocolLog (const wxString & *traceMask*)

Create object doing the logging using wxLogTrace() with the specified trace mask.

If you override DoLogString() in your class the *traceMask* may be left empty but it must have a valid value if you rely on the default DoLogString() implementation.

21.570.3 Member Function Documentation

virtual void wxProtocolLog::DoLogString (const wxString & *str*) `[protected],[virtual]`

Log the given string.

This function is called from LogRequest() and LogResponse() and by default uses wxLogTrace() with the trace mask specified in the constructor but can be overridden to do something different by the derived classes.

virtual void wxProtocolLog::LogRequest (const wxString & *str*) `[virtual]`

Called by wxProtocol-derived objects to log strings sent to the server.

Default implementation prepends a client-to-server marker to *str* and calls DoLogString().

virtual void wxProtocolLog::LogResponse (const wxString & *str*) `[virtual]`

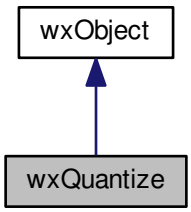
Called by wxProtocol-derived objects to log strings received from the server.

Default implementation prepends a server-to-client marker to *str* and calls DoLogString().

21.571 wxQuantize Class Reference

```
#include <wx/quantize.h>
```

Inheritance diagram for wxQuantize:



21.571.1 Detailed Description

Performs quantization, or colour reduction, on a wxImage.

Functions in this class are static and so a wxQuantize object need not be created.

Library: wxCore

Category: Miscellaneous

Public Member Functions

- wxQuantize ()

  Constructor.

Static Public Member Functions

- static void DoQuantize (unsigned int w, unsigned int h, unsigned char **in_rows, unsigned char **out_rows, unsigned char *palette, int desiredNoColours)

  Converts input bitmap(s) into 8bit representation with custom palette.

- static bool Quantize (const wxImage &src, wxImage &dest, wxPalette **pPalette, int desiredNoColours=236, unsigned char **eightBitData=0, int flags=wxQUANTIZE_INCLUDE_WINDOWS_COLOURS|wxQUANTIZE_FILL_DESTINATION_IMAGE|wxQUANTIZE_RETURN_8BIT_DATA)

  Reduce the colours in the source image and put the result into the destination image.

- static bool Quantize (const wxImage &src, wxImage &dest, int desiredNoColours=236, unsigned char **eightBitData=0, int flags=wxQUANTIZE_INCLUDE_WINDOWS_COLOURS|wxQUANTIZE_FILL_DESTINATION_IMAGE|wxQUANTIZE_RETURN_8BIT_DATA)

  This version sets a palette in the destination image so you don't have to manage it yourself.

Additional Inherited Members

21.571.2 Constructor & Destructor Documentation

wxQuantize::wxQuantize ( )

Constructor.

You do not need to construct a wxQuantize object since its functions are static.

21.571.3 Member Function Documentation

static void wxQuantize::DoQuantize (unsigned int *w,* unsigned int *h,* unsigned char ** *in_rows,* unsigned char ** *out_rows,* unsigned char * *palette,* int *desiredNoColours*) `[static]`

Converts input bitmap(s) into 8bit representation with custom palette.

in_rows and *out_rows* are arrays [0..h-1] of pointer to rows (*in_rows* contains *w* * 3 bytes per row, *out_rows w* bytes per row). Fills *out_rows* with indexes into palette (which is also stored into *palette* variable).

static bool wxQuantize::Quantize (const wxImage & *src,* wxImage & *dest,* wxPalette ** *pPalette,* int *desiredNoColours* = `236`, unsigned char ** *eightBitData* = `0`, int *flags* = `wxQUANTIZE_INCLUDE_WINDOWS_COLOURS|wxQUANTIZE_FILL_DESTINATION_IMAGE|wxQUANTIZE_RETURN_8BIT_DATA`) `[static]`

Reduce the colours in the source image and put the result into the destination image.

Both images may be the same, to overwrite the source image.

Specify an optional palette pointer to receive the resulting palette. This palette may be passed to ConvertImageToBitmap, for example.

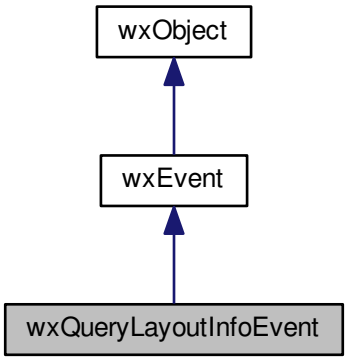
static bool wxQuantize::Quantize (const wxImage & *src,* wxImage & *dest,* int *desiredNoColours* = `236`, unsigned char ** *eightBitData* = `0`, int *flags* = `wxQUANTIZE_INCLUDE_WINDOWS_COLOURS|wxQUANTIZE_FILL_DESTINATION_IMAGE|wxQUANTIZE_RETURN_8BIT_DATA`) `[static]`

This version sets a palette in the destination image so you don't have to manage it yourself.

21.572 wxQueryLayoutInfoEvent Class Reference

`#include <wx/laywin.h>`

Inheritance diagram for wxQueryLayoutInfoEvent:



21.572.1 Detailed Description

This event is sent when wxLayoutAlgorithm wishes to get the size, orientation and alignment of a window.

More precisely, the event is sent by the OnCalculateLayout handler which is itself invoked by wxLayoutAlgorithm.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxQueryLayoutInfoEvent& event)

Event macros:

- EVT_QUERY_LAYOUT_INFO(func): Process a `wxEVT_QUERY_LAYOUT_INFO` event, to get size, orientation and alignment from a window.

Library: wxAdvanced

Category: Events

See also

wxCalculateLayoutEvent, wxSashLayoutWindow, wxLayoutAlgorithm.

Public Member Functions

- wxQueryLayoutInfoEvent (wxWindowID id=0)

  Constructor.
- wxLayoutAlignment GetAlignment () const

  Specifies the alignment of the window (which side of the remaining parent client area the window sticks to).
- int GetFlags () const

  Returns the flags associated with this event.

- wxLayoutOrientation GetOrientation () const

  Returns the orientation that the event handler specified to the event object.
- int GetRequestedLength () const

  Returns the requested length of the window in the direction of the window orientation.
- wxSize GetSize () const

  Returns the size that the event handler specified to the event object as being the requested size of the window.
- void SetAlignment (wxLayoutAlignment alignment)

  Call this to specify the alignment of the window (which side of the remaining parent client area the window sticks to).
- void SetFlags (int flags)

  Sets the flags associated with this event.
- void SetOrientation (wxLayoutOrientation orientation)

  Call this to specify the orientation of the window.
- void SetRequestedLength (int length)

  Sets the requested length of the window in the direction of the window orientation.
- void SetSize (const wxSize &size)

  Call this to let the calling code know what the size of the window is.

Additional Inherited Members

21.572.2 Constructor & Destructor Documentation

wxQueryLayoutInfoEvent::wxQueryLayoutInfoEvent (wxWindowID *id* = 0)

Constructor.

21.572.3 Member Function Documentation

wxLayoutAlignment wxQueryLayoutInfoEvent::GetAlignment ( ) const

Specifies the alignment of the window (which side of the remaining parent client area the window sticks to).

One of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.

int wxQueryLayoutInfoEvent::GetFlags ( ) const

Returns the flags associated with this event.

Not currently used.

wxLayoutOrientation wxQueryLayoutInfoEvent::GetOrientation ( ) const

Returns the orientation that the event handler specified to the event object.

May be one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.

int wxQueryLayoutInfoEvent::GetRequestedLength ( ) const

Returns the requested length of the window in the direction of the window orientation.

This information is not yet used.

wxSize wxQueryLayoutInfoEvent::GetSize ( ) const

Returns the size that the event handler specified to the event object as being the requested size of the window.

void wxQueryLayoutInfoEvent::SetAlignment (wxLayoutAlignment *alignment*)

Call this to specify the alignment of the window (which side of the remaining parent client area the window sticks to).

May be one of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.

void wxQueryLayoutInfoEvent::SetFlags (int *flags*)

Sets the flags associated with this event.

Not currently used.

void wxQueryLayoutInfoEvent::SetOrientation (wxLayoutOrientation *orientation*)

Call this to specify the orientation of the window.

May be one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.

void wxQueryLayoutInfoEvent::SetRequestedLength (int *length*)

Sets the requested length of the window in the direction of the window orientation.

This information is not yet used.

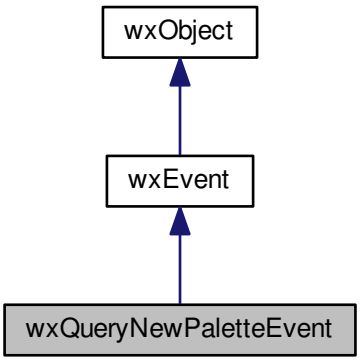
void wxQueryLayoutInfoEvent::SetSize (const wxSize & *size*)

Call this to let the calling code know what the size of the window is.

21.573 wxQueryNewPaletteEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxQueryNewPaletteEvent:



Public Member Functions

- wxQueryNewPaletteEvent (wxWindowID winid=0)
- void SetPaletteRealized (bool realized)
- bool GetPaletteRealized ()

Additional Inherited Members

21.573.1 Constructor & Destructor Documentation

wxQueryNewPaletteEvent::wxQueryNewPaletteEvent (wxWindowID *winid* = 0)

21.573.2 Member Function Documentation

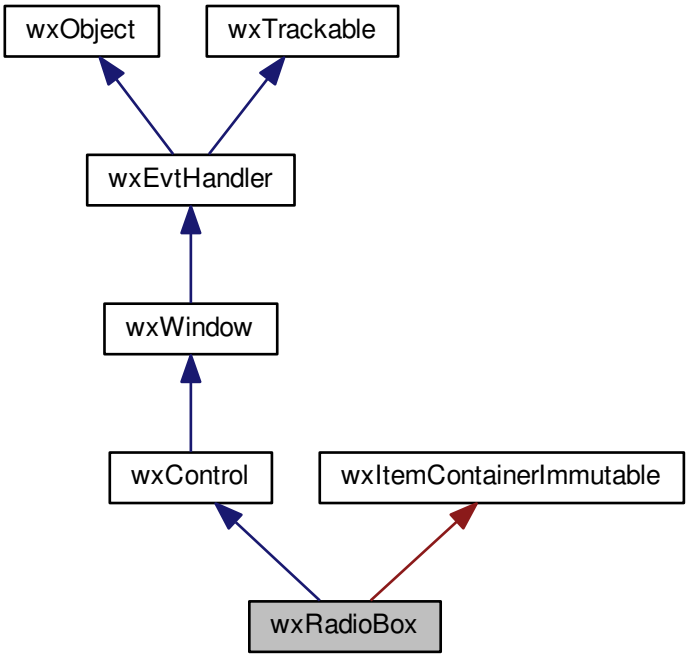
bool wxQueryNewPaletteEvent::GetPaletteRealized ( )

void wxQueryNewPaletteEvent::SetPaletteRealized (bool *realized*)

21.574 wxRadioBox Class Reference

```
#include <wx/radiobox.h>
```

Inheritance diagram for wxRadioBox:



21.574.1 Detailed Description

A radio box item is used to select one of number of mutually exclusive choices.

It is displayed as a vertical column or horizontal row of labelled buttons.

Styles

This class supports the following styles:

- wxRA_SPECIFY_ROWS: The major dimension parameter refers to the maximum number of rows.
- wxRA_SPECIFY_COLS: The major dimension parameter refers to the maximum number of columns.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_RADIOBOX(id, func): Process a `wxEVT_RADIOBOX` event, when a radiobutton is clicked.

Library: wxCore

Category: Controls

See also

Events and Event Handling, wxRadioButton, wxCheckBox

Public Member Functions

- wxRadioBox ()

  Default constructor.
- wxRadioBox (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, int majorDimension=0, long style=wxRA_SPECIFY_COLS, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRadioBoxNameStr)

  Constructor, creating and showing a radiobox.
- wxRadioBox (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, int majorDimension=0, long style=wxRA_SPECIFY_COLS, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRadioBoxNameStr)

  Constructor, creating and showing a radiobox.
- virtual ~wxRadioBox ()

  Destructor, destroying the radiobox item.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, int majorDimension=0, long style=wxRA_SPECIFY_COLS, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRadioBoxNameStr)

  Creates the radiobox for two-step construction.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, int majorDimension=0, long style=wxRA_SPECIFY_COLS, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRadioBoxNameStr)

  Creates the radiobox for two-step construction.
- virtual bool Enable (unsigned int n, bool enable=true)

  Enables or disables an individual button in the radiobox.
- virtual int FindString (const wxString &string, bool bCase=false) const

  Finds a button matching the given string, returning the position if found, or `wxNOT_FOUND` if not found.
- unsigned int GetColumnCount () const

  Returns the number of columns in the radiobox.
- virtual int GetItemFromPoint (const wxPoint &pt) const

  Returns a radio box item under the point, a zero-based item index, or `wxNOT_FOUND` if no item is under the point.
- wxString GetItemHelpText (unsigned int item) const

  Returns the helptext associated with the specified item if any or `wxEmptyString`.
- wxToolTip * GetItemToolTip (unsigned int item) const

  Returns the tooltip associated with the specified item if any or NULL.
- unsigned int GetRowCount () const

  Returns the number of rows in the radiobox.
- virtual bool IsItemEnabled (unsigned int n) const

  Returns true if the item is enabled or false if it was disabled using Enable(n, false).
- virtual bool IsItemShown (unsigned int n) const

  Returns true if the item is currently shown or false if it was hidden using Show(n, false).
- void SetItemHelpText (unsigned int item, const wxString &helptext)

  Sets the helptext for an item.
- void SetItemToolTip (unsigned int item, const wxString &text)

  Sets the tooltip text for the specified item in the radio group.
- virtual void SetSelection (int n)

Sets the selection to the given item.

- virtual bool Show (unsigned int item, bool show=true)

  Shows or hides individual buttons.

- virtual unsigned int GetCount () const

  Returns the number of items in the control.

- virtual wxString GetString (unsigned int n) const

  Returns the label of the item with the given index.

- virtual void SetString (unsigned int n, const wxString &string)

  Sets the label for the given item.

- virtual int GetSelection () const

  Returns the index of the selected item or `wxNOT_FOUND` *if no item is selected.*

Additional Inherited Members

21.574.2 Constructor & Destructor Documentation

wxRadioBox::wxRadioBox ( )

Default constructor.

See also

Create(), wxValidator

wxRadioBox::wxRadioBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = `0`, const wxString *choices[]* = `NULL`, int *majorDimension* = `0`, long *style* = wxRA_SPECIFY_COLS, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxRadioBoxNameStr`)

Constructor, creating and showing a radiobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value `wxID_ANY` indicates a default value.
label	Label for the static box surrounding the radio buttons.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
n	Number of choices with which to initialize the radiobox.
choices	An array of choices with which to initialize the radiobox.
majorDimension	Specifies the maximum number of rows (if style contains `wxRA_SPECIFY_ROWS`) or columns (if style contains `wxRA_SPECIFY_COLS`) for a two-dimensional radiobox. The default value of 0 means to use the number of items, i.e. *n*.
style	Window style. See wxRadioBox.
validator	Window validator.
name	Window name.

wxPerl Note: Not supported by wxPerl.

See also

Create(), wxValidator

wxRadioBox::wxRadioBox (wxWindow ∗ ***parent*****, wxWindowID** ***id*****,** const **wxString** & ***label*****,** const **wxPoint** & ***pos*****,** const **wxSize** & ***size*****,** const **wxArrayString** & ***choices*****,** int ***majorDimension*** **=** `0`**,** long ***style*** **= wxRA_SPECIFY_COLS,** const **wxValidator** & ***validator*** **= wxDefaultValidator,** const **wxString** & ***name*** **=** `wxRadioBoxNameStr` **)**

Constructor, creating and showing a radiobox.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value `wxID_ANY` indicates a default value.
label	Label for the static box surrounding the radio buttons.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
choices	An array of choices with which to initialize the radiobox.
majorDimension	Specifies the maximum number of rows (if style contains `wxRA_SPECIFY_ROWS`) or columns (if style contains `wxRA_SPECIFY_COLS`) for a two-dimensional radiobox. The default value of 0 means to use the number of items, i.e. number of elements in *choices*.
style	Window style. See wxRadioBox.
validator	Window validator.
name	Window name.

wxPerl Note: Use an array reference for the *choices* parameter.

See also

Create(), wxValidator

virtual wxRadioBox::~wxRadioBox ( ) [virtual]

Destructor, destroying the radiobox item.

21.574.3 Member Function Documentation

bool wxRadioBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, int *majorDimension* = 0, long *style* = wxRA_SPECIFY_COLS, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxRadioBoxNameStr)

Creates the radiobox for two-step construction.

See wxRadioBox() for further details.

bool wxRadioBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, int *majorDimension* = 0, long *style* = wxRA_SPECIFY_COLS, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxRadioBoxNameStr)

Creates the radiobox for two-step construction.

See wxRadioBox() for further details.

virtual bool wxRadioBox::Enable (unsigned int *n*, bool *enable* = true) [virtual]

Enables or disables an individual button in the radiobox.

Parameters

enable	true to enable, false to disable.
n	The zero-based button to enable or disable.

See also

wxWindow::Enable()

virtual int wxRadioBox::FindString (const wxString & *string,* bool *bCase* = `false`) const `[virtual]`

Finds a button matching the given string, returning the position if found, or `wxNOT_FOUND` if not found.

Parameters

string	The string to find.
bCase	Should the search be case-sensitive?

Reimplemented from wxItemContainerImmutable.

unsigned int wxRadioBox::GetColumnCount ( ) const

Returns the number of columns in the radiobox.

virtual unsigned int wxRadioBox::GetCount ( ) const `[virtual]`

Returns the number of items in the control.

See also

IsEmpty()

Implements wxItemContainerImmutable.

virtual int wxRadioBox::GetItemFromPoint (const wxPoint & *pt*) const `[virtual]`

Returns a radio box item under the point, a zero-based item index, or `wxNOT_FOUND` if no item is under the point.

Parameters

pt	Point in client coordinates.

wxString wxRadioBox::GetItemHelpText (unsigned int *item*) const

Returns the helptext associated with the specified *item* if any or `wxEmptyString`.

Parameters

item	The zero-based item index.

See also

SetItemHelpText()

wxToolTip* wxRadioBox::GetItemToolTip (unsigned int *item*) const

Returns the tooltip associated with the specified *item* if any or NULL.

See also

SetItemToolTip(), wxWindow::GetToolTip()

unsigned int wxRadioBox::GetRowCount ( ) const

Returns the number of rows in the radiobox.

virtual int wxRadioBox::GetSelection ( ) const `[virtual]`

Returns the index of the selected item or `wxNOT_FOUND` if no item is selected.

Returns

The position of the current selection.

Remarks

This method can be used with single selection list boxes only, you should use wxListBox::GetSelections() for the list boxes with wxLB_MULTIPLE style.

See also

SetSelection(), GetStringSelection()

Implements wxItemContainerImmutable.

virtual wxString wxRadioBox::GetString (unsigned int *n*) const `[virtual]`

Returns the label of the item with the given index.

Parameters

n	The zero-based index.

Returns

The label of the item or an empty string if the position was invalid.

Implements wxItemContainerImmutable.

virtual bool wxRadioBox::IsItemEnabled (unsigned int *n*) const `[virtual]`

Returns true if the item is enabled or false if it was disabled using Enable(n, false).

This function is currently only implemented in wxMSW, wxGTK, wxQT and wxUniversal and always returns true in the other ports.

Parameters

n	The zero-based button position.

virtual bool wxRadioBox::IsItemShown (unsigned int *n*) const `[virtual]`

Returns true if the item is currently shown or false if it was hidden using Show(n, false).

Note that this function returns true for an item which hadn't been hidden even if the entire radiobox is not currently shown.

This function is currently only implemented in wxMSW, wxGTK, wxQT and wxUniversal and always returns true in the other ports.

Parameters

n	The zero-based button position.

void wxRadioBox::SetItemHelpText (unsigned int *item*, const wxString & *helptext*)

Sets the helptext for an item.

Empty string erases any existing helptext.

Parameters

item	The zero-based item index.
helptext	The help text to set for the item.

See also

GetItemHelpText()

void wxRadioBox::SetItemToolTip (unsigned int *item*, const wxString & *text*)

Sets the tooltip text for the specified item in the radio group.

This function is currently only implemented in wxMSW and wxGTK2 and does nothing in the other ports.

Parameters

item	Index of the item the tooltip will be shown for.
text	Tooltip text for the item, the tooltip is removed if empty.

See also

GetItemToolTip(), wxWindow::SetToolTip()

virtual void wxRadioBox::SetSelection (int *n*) [virtual]

Sets the selection to the given item.

Notice that a radio box always has selection, so *n* must be valid here and passing `wxNOT_FOUND` is not allowed.

Implements wxItemContainerImmutable.

virtual void wxRadioBox::SetString (unsigned int *n*, const wxString & *string*) [virtual]

Sets the label for the given item.

Parameters

n	The zero-based item index.
string	The label to set.

Implements wxItemContainerImmutable.

virtual bool wxRadioBox::Show (unsigned int *item*, bool *show* = `true`) [virtual]

Shows or hides individual buttons.

Parameters

show	true to show, false to hide.
item	The zero-based position of the button to show or hide.

Returns

true if the item has been shown or hidden or false if nothing was done because it already was in the requested state.

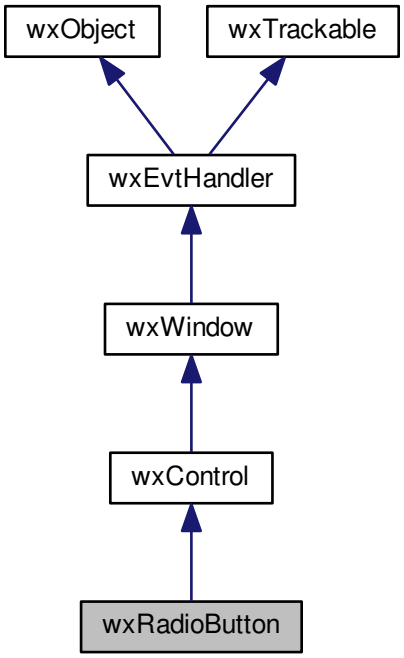
See also

wxWindow::Show()

21.575 wxRadioButton Class Reference

`#include <wx/radiobut.h>`

Inheritance diagram for wxRadioButton:



21.575.1 Detailed Description

A radio button item is a button which usually denotes one of several mutually exclusive options.

It has a text label next to a (usually) round button.

You can create a group of mutually-exclusive radio buttons by specifying `wxRB_GROUP` for the first in the group. The group ends when another radio button group is created, or there are no more radio buttons.

Styles

This class supports the following styles:

- wxRB_GROUP: Marks the beginning of a new group of radio buttons.
- wxRB_SINGLE: In some circumstances, radio buttons that are not consecutive siblings trigger a hang bug in Windows (only). If this happens, add this style to mark the button as not belonging to a group, and implement the mutually-exclusive group behaviour yourself.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_RADIOBUTTON(id, func): Process a `wxEVT_RADIOBUTTON` event, when the radiobutton is clicked.

Library: wxCore

Category: Controls

See also

Events and Event Handling, wxRadioBox, wxCheckBox

Public Member Functions

- wxRadioButton ()

  Default constructor.
- wxRadioButton (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRadioButtonNameStr)

  Constructor, creating and showing a radio button.
- virtual ~wxRadioButton ()

  Destructor, destroying the radio button item.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRadioButtonNameStr)

  Creates the choice for two-step construction.
- virtual bool GetValue () const

  Returns true if the radio button is checked, false otherwise.
- virtual void SetValue (bool value)

  Sets the radio button to checked or unchecked status.

Additional Inherited Members

21.575.2 Constructor & Destructor Documentation

wxRadioButton::wxRadioButton ( )

Default constructor.

See also

Create(), wxValidator

wxRadioButton::wxRadioButton (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxRadioButtonNameStr)

Constructor, creating and showing a radio button.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
label	Label for the radio button.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxRadioButton.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxRadioButton::~wxRadioButton ( ) [virtual]

Destructor, destroying the radio button item.

21.575.3 Member Function Documentation

bool wxRadioButton::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxRadioButtonNameStr)

Creates the choice for two-step construction.

See wxRadioButton() for further details.

virtual bool wxRadioButton::GetValue ( ) const [virtual]

Returns true if the radio button is checked, false otherwise.

virtual void wxRadioButton::SetValue (bool *value*) [virtual]

Sets the radio button to checked or unchecked status.

This does not cause a wxEVT_RADIOBUTTON event to get emitted.

If the radio button belongs to a radio group exactly one button in the group may be checked and so this method can be only called with *value* set to true. To uncheck a radio button in a group you must check another button in the same group.

Parameters

value	true to check, false to uncheck.

21.576 wxRealPoint Class Reference

`#include <wx/gdicmn.h>`

21.576.1 Detailed Description

A wxRealPoint is a useful data structure for graphics operations.

It contains floating point *x* and *y* members. See wxPoint for an integer version.

Note that the coordinates stored inside a wxRealPoint object may be negative and that wxRealPoint functions do not perform any check against negative values.

Library: wxCore

Category: Data Structures

See also

wxPoint

Public Member Functions

- wxRealPoint ()

  Initializes to zero the x and y members.
- wxRealPoint (double x, double y)

  Initializes the point with the given coordinates.
- wxRealPoint (const wxPoint &pt)

  Converts the given wxPoint (with integer coordinates) to a wxRealPoint.

Miscellaneous operators

Note that these operators are documented as class members (to make them easier to find) but, as their prototype shows, they are implemented as global operators; note that this is transparent to the user but it helps to understand why the following functions are documented to take the wxPoint they operate on as an explicit argument.

- wxRealPoint & operator= (const wxRealPoint &pt)
- bool operator== (const wxRealPoint &p1, const wxRealPoint &p2)
- bool operator!= (const wxRealPoint &p1, const wxRealPoint &p2)
- wxRealPoint operator+ (const wxRealPoint &p1, const wxRealPoint &p2)
- wxRealPoint operator- (const wxRealPoint &p1, const wxRealPoint &p2)
- wxRealPoint & operator+= (const wxRealPoint &pt)
- wxRealPoint & operator-= (const wxRealPoint &pt)
- wxRealPoint operator+ (const wxRealPoint &pt, const wxSize &sz)
- wxRealPoint operator- (const wxRealPoint &pt, const wxSize &sz)
- wxRealPoint operator+ (const wxSize &sz, const wxRealPoint &pt)
- wxRealPoint operator- (const wxSize &sz, const wxRealPoint &pt)
- wxRealPoint & operator+= (const wxSize &sz)
- wxRealPoint & operator-= (const wxSize &sz)
- wxSize operator/ (const wxRealPoint &sz, int factor)

- wxSize operator∗ (const wxRealPoint &sz, int factor)
- wxSize operator∗ (int factor, const wxSize &sz)
- wxSize & operator/= (int factor)
- wxSize & operator∗= (int factor)

Public Attributes

- double x

  X coordinate of this point.
- double y

  Y coordinate of this point.

21.576.2 Constructor & Destructor Documentation

wxRealPoint::wxRealPoint ( )

Initializes to zero the x and y members.

wxRealPoint::wxRealPoint (double *x*, double *y*)

Initializes the point with the given coordinates.

wxRealPoint::wxRealPoint (const wxPoint & *pt*)

Converts the given wxPoint (with integer coordinates) to a wxRealPoint.

21.576.3 Member Function Documentation

bool wxRealPoint::operator!= (const wxRealPoint & *p1*, const wxRealPoint & *p2*)

wxSize wxRealPoint::operator∗ (const wxRealPoint & *sz*, int *factor*)

wxSize wxRealPoint::operator∗ (int *factor*, const wxSize & *sz*)

wxSize& wxRealPoint::operator∗= (int *factor*)

wxRealPoint wxRealPoint::operator+ (const wxRealPoint & *p1*, const wxRealPoint & *p2*)

wxRealPoint wxRealPoint::operator+ (const wxRealPoint & *pt*, const wxSize & *sz*)

wxRealPoint wxRealPoint::operator+ (const wxSize & *sz*, const wxRealPoint & *pt*)

wxRealPoint& wxRealPoint::operator+= (const wxRealPoint & *pt*)

wxRealPoint& wxRealPoint::operator+= (const wxSize & *sz*)

wxRealPoint wxRealPoint::operator- (const wxRealPoint & *p1*, const wxRealPoint & *p2*)

wxRealPoint wxRealPoint::operator- (const wxRealPoint & *pt*, const wxSize & *sz*)

wxRealPoint wxRealPoint::operator- (const wxSize & *sz*, const wxRealPoint & *pt*)

wxRealPoint& wxRealPoint::operator-= (const wxRealPoint & *pt*)

wxRealPoint& wxRealPoint::operator-= (const wxSize & *sz*)

wxSize wxRealPoint::operator/ (const wxRealPoint & *sz*, int *factor*)

wxSize& wxRealPoint::operator/= (int *factor*)

wxRealPoint& wxRealPoint::operator= (const wxRealPoint & *pt*)

bool wxRealPoint::operator== (const wxRealPoint & *p1*, const wxRealPoint & *p2*)

21.576.4 Member Data Documentation

double wxRealPoint::x

X coordinate of this point.

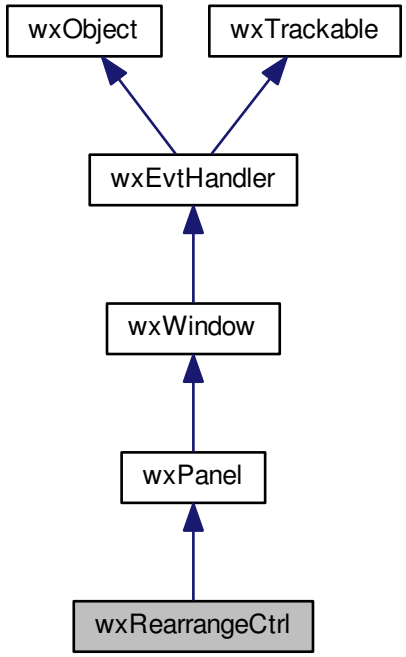
double wxRealPoint::y

Y coordinate of this point.

21.577 wxRearrangeCtrl Class Reference

`#include <wx/rearrangectrl.h>`

Inheritance diagram for wxRearrangeCtrl:



21.577.1 Detailed Description

A composite control containing a wxRearrangeList and the buttons allowing to move the items in it.

This control is in fact a panel containing the wxRearrangeList control and the "Up" and "Down" buttons to move the currently selected item up or down. It is used as the main part of a wxRearrangeDialog.

Since

2.9.0

Library: wxCore

Category: Controls

Public Member Functions

- wxRearrangeCtrl ()

  Default constructor.
- wxRearrangeCtrl (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayInt &order, const wxArrayString &items, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRearrangeListNameStr)

  Constructor really creating the control.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayInt &order, const wxArrayString &items, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRearrangeListNameStr)

  Effectively creates the window for an object created using the default constructor.
- wxRearrangeList * GetList () const

  Return the listbox which is the main part of this control.

Additional Inherited Members

21.577.2 Constructor & Destructor Documentation

wxRearrangeCtrl::wxRearrangeCtrl ( )

Default constructor.

Create() must be called later to effectively create the control.

wxRearrangeCtrl::wxRearrangeCtrl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayInt & *order*, const wxArrayString & *items*, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxRearrangeListNameStr`)

Constructor really creating the control.

Please see Create() for the parameters description.

21.577.3 Member Function Documentation

bool wxRearrangeCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayInt & *order*, const wxArrayString & *items*, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxRearrangeListNameStr`)

Effectively creates the window for an object created using the default constructor.

The parameters of this method are the same as for wxRearrangeList::Create().

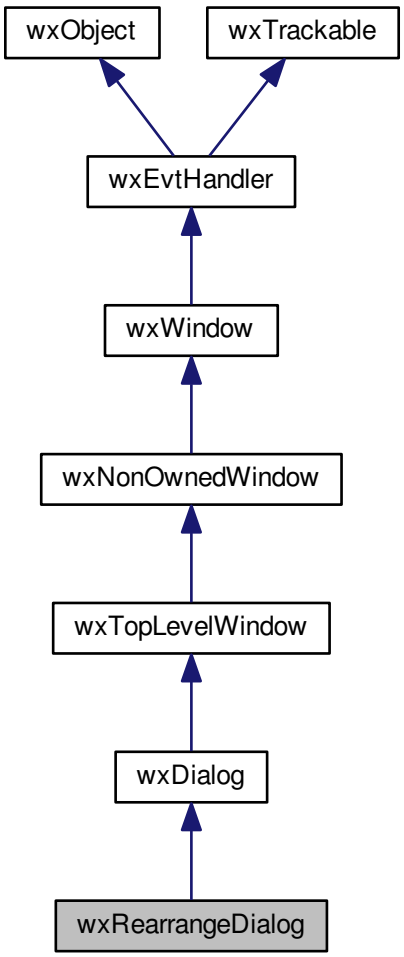
wxRearrangeList* wxRearrangeCtrl::GetList ( ) const

Return the listbox which is the main part of this control.

21.578 wxRearrangeDialog Class Reference

`#include <wx/rearrangectrl.h>`

Inheritance diagram for wxRearrangeDialog:



21.578.1 Detailed Description

A dialog allowing the user to rearrange the specified items.

This dialog can be used to allow the user to modify the order of the items and to enable or disable them individually. For example:

```
wxArrayString items;
items.push_back("meat");
items.push_back("fish");
items.push_back("fruits");
items.push_back("beer");
wxArrayInt order;
order.push_back(3);
order.push_back(0);
order.push_back(1);
order.push_back(2);

wxRearrangeDialog dlg(NULL,
                      "You can also uncheck the items you don't like "
                      "at all.",
                      "Sort the items in order of preference",
                      order, items);
if ( dlg.ShowModal() == wxID_OK ) {
    order = dlg.GetOrder();
    for ( size_t n = 0; n < order.size(); n++ ) {
        if ( order[n] >= 0 ) {
            wxLogMessage("Your most preferred item is \"%s\"",
                         items[order[n]]);
            break;
        }
    }
}
```

Since

2.9.0

Library: wxCore

Category: Common Dialogs

Public Member Functions

- wxRearrangeDialog ()

  Default constructor.

- wxRearrangeDialog (wxWindow *parent, const wxString &message, const wxString &title, const wxArrayInt &order, const wxArrayString &items, const wxPoint &pos=wxDefaultPosition, const wxString &name=wxRearrangeDialogNameStr)

  Constructor creating the dialog.

- bool Create (wxWindow *parent, const wxString &message, const wxString &title, const wxArrayInt &order, const wxArrayString &items, const wxPoint &pos=wxDefaultPosition, const wxString &name=wxRearrangeDialogNameStr)

  Effectively creates the dialog for an object created using the default constructor.

- void AddExtraControls (wxWindow *win)

  Customize the dialog by adding extra controls to it.

- wxRearrangeList * GetList () const

  Return the list control used by the dialog.

- wxArrayInt GetOrder () const

  Return the array describing the order of items after it was modified by the user.

Additional Inherited Members

21.578.2 Constructor & Destructor Documentation

wxRearrangeDialog::wxRearrangeDialog ( )

Default constructor.

Create() must be called later to effectively create the control.

wxRearrangeDialog::wxRearrangeDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *title*, const wxArrayInt & *order*, const wxArrayString & *items*, const wxPoint & *pos* = wxDefaultPosition, const wxString & *name* = `wxRearrangeDialogNameStr`)

Constructor creating the dialog.

Please see Create() for the parameters description.

21.578.3 Member Function Documentation

void wxRearrangeDialog::AddExtraControls (wxWindow * *win*)

Customize the dialog by adding extra controls to it.

This function adds the given *win* to the dialog, putting it just below the part occupied by wxRearrangeCtrl. It must be called after creating the dialog and you will typically need to process the events generated by the extra controls for them to do something useful.

For example:

```
class MyRearrangeDialog : public wxRearrangeDialog
{
public:
    MyRearrangeDialog(wxWindow *parent, ...)
        : wxRearrangeDialog(parent, ...)
    {
        wxPanel *panel = new wxPanel(this);
        wxSizer *sizer = new wxBoxSizer(wxHORIZONTAL);
        sizer->Add(new wxStaticText(panel, wxID_ANY,
                                    "Column width in pixels:"));
        sizer->Add(new wxTextCtrl(panel, wxID_ANY, ""));
        panel->SetSizer(sizer);
        AddExtraControls(panel);
    }

    ... code to update the text control with the currently selected
        item width and to react to its changes omitted ...
};
```

See also the complete example of a custom rearrange dialog in the dialogs sample.

Parameters

win	The window containing the extra controls. It must have this dialog as its parent.

bool wxRearrangeDialog::Create (wxWindow * *parent*, const wxString & *message*, const wxString & *title*, const wxArrayInt & *order*, const wxArrayString & *items*, const wxPoint & *pos* = wxDefaultPosition, const wxString & *name* = `wxRearrangeDialogNameStr`)

Effectively creates the dialog for an object created using the default constructor.

Parameters

parent	The dialog parent, possibly NULL.
message	The message shown inside the dialog itself, above the items list.
title	The title of the dialog.
order	The initial order of the items in the convention used by wxRearrangeList.
items	The items to show in the dialog.
pos	Optional dialog position.
name	Optional dialog name.

Returns

true if the dialog was successfully created or false if creation failed.

wxRearrangeList* wxRearrangeDialog::GetList ( ) const

Return the list control used by the dialog.

See also

wxRearrangeCtrl::GetList()

wxArrayInt wxRearrangeDialog::GetOrder ( ) const

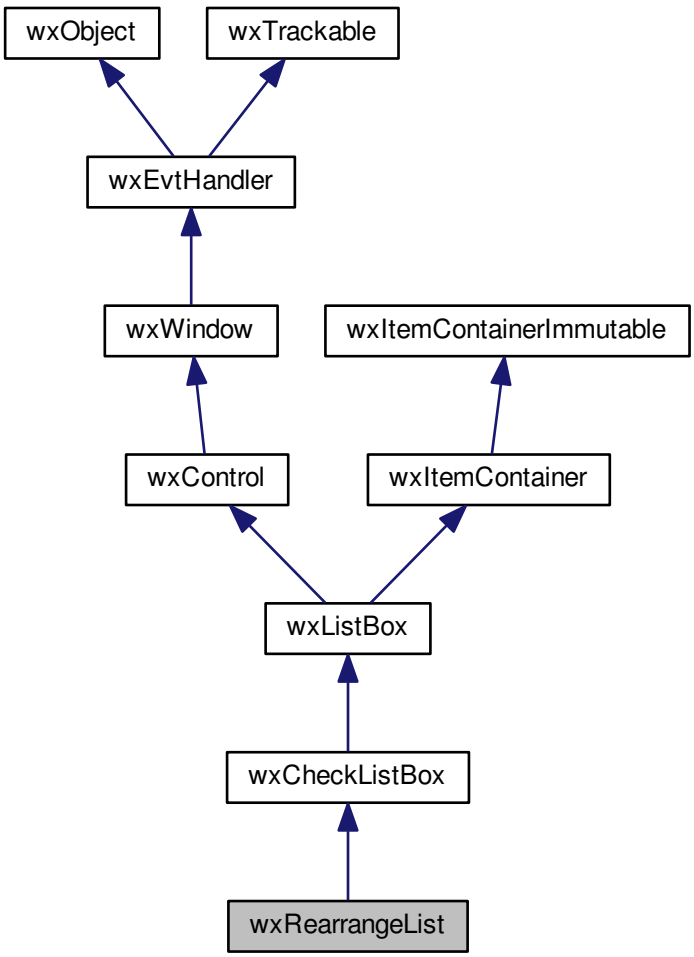
Return the array describing the order of items after it was modified by the user.

Please notice that the array will contain negative items if any items were unchecked. See wxRearrangeList for more information about the convention used for this array.

21.579 wxRearrangeList Class Reference

```
#include <wx/rearrangectrl.h>
```

Inheritance diagram for wxRearrangeList:



21.579.1 Detailed Description

A listbox-like control allowing the user to rearrange the items and to enable or disable them.

This class allows to change the order of the items shown in it as well as to check or uncheck them individually. The data structure used to allow this is the order array which contains the items indices indexed by their position with an added twist that the unchecked items are represented by the bitwise complement of the corresponding index (for any architecture using two's complement for negative numbers representation (i.e. just about any at all) this means that a checked item N is represented by -N-1 in unchecked state). In practice this means that you must apply the C bitwise complement operator when constructing the order array, e.g.

```
wxArrayInt order;
order.push_back(0); // checked item #0
order.push_back(~1); // unchecked item #1
```

So, for example, the array order [1 -3 0] used in conjunction with the items array ["first", "second", "third"] means that the items order is "second", "third", "first" and the "third" item is unchecked while the other two are checked.

This convention is used both for the order argument of the control ctor or Create() and for the array returned from GetCurrentOrder().

Usually this control will be used together with other controls allowing to move the items around in it interactively. The simplest possible solution is to use wxRearrangeCtrl which combines it with two standard buttons to move the current item up or down.

Since

2.9.0

Library: wxCore

Category: Controls

Public Member Functions

- wxRearrangeList ()

  Default constructor.

- wxRearrangeList (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayInt &order, const wxArrayString &items, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRearrangeListNameStr)

  Constructor really creating the control.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayInt &order, const wxArrayString &items, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxRearrangeListNameStr)

  Effectively creates the window for an object created using the default constructor.

- const wxArrayInt & GetCurrentOrder () const

  Return the current order of the items.

- bool CanMoveCurrentUp () const

  Return true if the currently selected item can be moved up.

- bool CanMoveCurrentDown () const

  Return true if the currently selected item can be moved down.

- bool MoveCurrentUp ()

  Move the currently selected item one position above.

- bool MoveCurrentDown ()

  Move the currently selected item one position below.

Additional Inherited Members

21.579.2 Constructor & Destructor Documentation

wxRearrangeList::wxRearrangeList ( )

Default constructor.

Create() must be called later to effectively create the control.

wxRearrangeList::wxRearrangeList (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayInt & *order*, const wxArrayString & *items*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxRearrangeListNameStr)

Constructor really creating the control.

Please see Create() for the parameters description.

21.579.3 Member Function Documentation

bool wxRearrangeList::CanMoveCurrentDown ( ) const

Return true if the currently selected item can be moved down.

See also

CanMoveCurrentUp()

bool wxRearrangeList::CanMoveCurrentUp ( ) const

Return true if the currently selected item can be moved up.

This function is useful for EVT_UPDATE_UI handler for the standard "Up" button often used together with this control and wxRearrangeCtrl uses it in this way.

Returns

true if the currently selected item can be moved up in the listbox, false if there is no selection or the current item is the first one.

See also

CanMoveCurrentDown()

bool wxRearrangeList::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayInt & *order*, const wxArrayString & *items*, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxRearrangeListNameStr)

Effectively creates the window for an object created using the default constructor.

This function is very similar to wxCheckListBox::Create() except that it has an additional parameter specifying the initial order of the items. Please see the class documentation for the explanation of the conventions used by the *order* argument.

Parameters

parent	The parent window, must be non-NULL.
id	The window identifier.
pos	The initial window position.
size	The initial window size.
order	Array specifying the initial order of the items in *items* array.
items	The items to display in the list.

style	The control style, there are no special styles for this class but the base class styles can be used here.
validator	Optional window validator.
name	Optional window name.

const wxArrayInt& wxRearrangeList::GetCurrentOrder ( ) const

Return the current order of the items.

The order may be different from the one passed to the constructor if MoveCurrentUp() or MoveCurrentDown() were called.

bool wxRearrangeList::MoveCurrentDown ( )

Move the currently selected item one position below.

See also

MoveCurrentUp()

bool wxRearrangeList::MoveCurrentUp ( )

Move the currently selected item one position above.

This method is useful to implement the standard "Up" button behaviour and wxRearrangeCtrl uses it for this.

Returns

true if the item was moved or false if this couldn't be done.

See also

MoveCurrentDown()

21.580 wxRect Class Reference

```
#include <wx/gdicmn.h>
```

21.580.1 Detailed Description

A class for manipulating rectangles.

Note that the x, y coordinates and the width and height stored inside a wxRect object may be negative and that wxRect functions do not perform any check against negative values.

Library: wxCore

Category: Data Structures

See also

wxPoint, wxSize

Public Member Functions

- wxRect ()

  Default constructor.
- wxRect (int x, int y, int width, int height)

  Creates a wxRect object from x, y, width and height values.
- wxRect (const wxPoint &topLeft, const wxPoint &bottomRight)

  Creates a wxRect object from top-left and bottom-right points.
- wxRect (const wxPoint &pos, const wxSize &size)

  Creates a wxRect object from position pos and size values.
- wxRect (const wxSize &size)

  Creates a wxRect object from size values at the origin.
- bool Contains (int x, int y) const

  Returns true if the given point is inside the rectangle (or on its boundary) and false otherwise.
- bool Contains (const wxPoint &pt) const

  Returns true if the given point is inside the rectangle (or on its boundary) and false otherwise.
- bool Contains (const wxRect &rect) const

  Returns true if the given rectangle is completely inside this rectangle (or touches its boundary) and false otherwise.
- int GetBottom () const

  Gets the bottom point of the rectangle.
- wxPoint GetBottomLeft () const

  Gets the position of the bottom left corner.
- wxPoint GetBottomRight () const

  Gets the position of the bottom right corner.
- int GetHeight () const

  Gets the height member.
- int GetLeft () const

  Gets the left point of the rectangle (the same as GetX()).
- wxPoint GetPosition () const

  Gets the position.
- int GetRight () const

  Gets the right point of the rectangle.
- wxSize GetSize () const

  Gets the size.
- int GetTop () const

  Gets the top point of the rectangle (the same as GetY()).
- wxPoint GetTopLeft () const

  Gets the position of the top left corner of the rectangle, same as GetPosition().
- wxPoint GetTopRight () const

  Gets the position of the top right corner.
- int GetWidth () const

  Gets the width member.
- int GetX () const

  Gets the x member.
- int GetY () const

  Gets the y member.
- wxRect & Intersect (const wxRect &rect)

  Modifies this rectangle to contain the overlapping portion of this rectangle and the one passed in as parameter.
- wxRect Intersect (const wxRect &rect) const

  Returns the overlapping portion of this rectangle and the one passed in as parameter.
- bool Intersects (const wxRect &rect) const

Returns true if this rectangle has a non-empty intersection with the rectangle rect and false otherwise.

- bool IsEmpty () const

  Returns true if this rectangle has a width or height less than or equal to 0 and false otherwise.
- void SetHeight (int height)

  Sets the height.
- void SetPosition (const wxPoint &pos)

  Sets the position.
- void SetSize (const wxSize &s)

  Sets the size.
- void SetWidth (int width)

  Sets the width.
- void SetX (int x)

  Sets the x position.
- void SetY (int y)

  Sets the y position.
- void SetLeft (int left)

  Set the left side of the rectangle.
- void SetRight (int right)

  Set the right side of the rectangle.
- void SetTop (int top)

  Set the top edge of the rectangle.
- void SetBottom (int bottom)

  Set the bottom edge of the rectangle.
- void SetTopLeft (const wxPoint &p)

  Set the top-left point of the rectangle.
- void SetBottomRight (const wxPoint &p)

  Set the bottom-right point of the rectangle.
- void SetTopRight (const wxPoint &p)

  Set the top-right point of the rectangle.
- void SetBottomLeft (const wxPoint &p)

  Set the bottom-left point of the rectangle.
- bool operator!= (const wxRect &r1, const wxRect &r2)

  Inequality operator.
- wxRect & operator= (const wxRect &rect)

  Assignment operator.
- bool operator== (const wxRect &r1, const wxRect &r2)

  Equality operator.

- wxRect CentreIn (const wxRect &r, int dir=wxBOTH) const

  Returns the rectangle having the same size as this one but centered relatively to the given rectangle r.
- wxRect CenterIn (const wxRect &r, int dir=wxBOTH) const

  Returns the rectangle having the same size as this one but centered relatively to the given rectangle r.

- wxRect & Deflate (wxCoord dx, wxCoord dy)

  Decrease the rectangle size.
- wxRect & Deflate (const wxSize &diff)

  Decrease the rectangle size.
- wxRect & Deflate (wxCoord diff)

  Decrease the rectangle size.
- wxRect Deflate (wxCoord dx, wxCoord dy) const

  Decrease the rectangle size.

- wxRect & Inflate (wxCoord dx, wxCoord dy)

  Increases the size of the rectangle.
- wxRect & Inflate (const wxSize &diff)

  Increases the size of the rectangle.
- wxRect & Inflate (wxCoord diff)

  Increases the size of the rectangle.
- wxRect Inflate (wxCoord dx, wxCoord dy) const

  Increases the size of the rectangle.

- void Offset (wxCoord dx, wxCoord dy)

  Moves the rectangle by the specified offset.
- void Offset (const wxPoint &pt)

  Moves the rectangle by the specified offset.

- wxRect Union (const wxRect &rect) const

  Modifies the rectangle to contain the bounding box of this rectangle and the one passed in as parameter.
- wxRect & Union (const wxRect &rect)

  Modifies the rectangle to contain the bounding box of this rectangle and the one passed in as parameter.

- wxRect operator+ (const wxRect &r1, const wxRect &r2)

  Like Union(), but doesn't treat empty rectangles specially.
- wxRect & operator+= (const wxRect &r)

  Like Union(), but doesn't treat empty rectangles specially.

- wxRect operator∗ (const wxRect &r1, const wxRect &r2)

  Returns the intersection of two rectangles (which may be empty).
- wxRect & operator∗= (const wxRect &r)

  Returns the intersection of two rectangles (which may be empty).

Public Attributes

- int height

  Height member.
- int width

  Width member.
- int x

  x coordinate of the top-level corner of the rectangle.
- int y

  y coordinate of the top-level corner of the rectangle.

21.580.2 Constructor & Destructor Documentation

wxRect::wxRect ( )

Default constructor.

Initializes to zero the internal *x*, *y*, *width* and *height* members.

wxRect::wxRect (int *x*, int *y*, int *width*, int *height*)

Creates a wxRect object from *x*, *y*, *width* and *height* values.

wxRect::wxRect (const wxPoint & *topLeft,* const wxPoint & *bottomRight*)

Creates a wxRect object from top-left and bottom-right points.

wxRect::wxRect (const wxPoint & *pos,* const wxSize & *size*)

Creates a wxRect object from position *pos* and *size* values.

wxRect::wxRect (const wxSize & *size*)

Creates a wxRect object from *size* values at the origin.

21.580.3 Member Function Documentation

wxRect wxRect::CenterIn (const wxRect & *r,* int *dir* = wxBOTH) const

Returns the rectangle having the same size as this one but centered relatively to the given rectangle *r*.

By default, rectangle is centred in both directions but if *dir* includes only `wxVERTICAL` or only `wxHORIZONTAL`, then it is only centered in this direction while the other component of its position remains unchanged.

wxRect wxRect::CentreIn (const wxRect & *r,* int *dir* = wxBOTH) const

Returns the rectangle having the same size as this one but centered relatively to the given rectangle *r*.

By default, rectangle is centred in both directions but if *dir* includes only `wxVERTICAL` or only `wxHORIZONTAL`, then it is only centered in this direction while the other component of its position remains unchanged.

bool wxRect::Contains (int *x,* int *y*) const

Returns true if the given point is inside the rectangle (or on its boundary) and false otherwise.

bool wxRect::Contains (const wxPoint & *pt*) const

Returns true if the given point is inside the rectangle (or on its boundary) and false otherwise.

bool wxRect::Contains (const wxRect & *rect*) const

Returns true if the given rectangle is completely inside this rectangle (or touches its boundary) and false otherwise.

wxRect& wxRect::Deflate (wxCoord *dx,* wxCoord *dy*)

Decrease the rectangle size.

This method is the opposite from Inflate(): Deflate(a, b) is equivalent to Inflate(-a, -b). Please refer to Inflate() for full description.

wxRect& wxRect::Deflate (const wxSize & *diff*)

Decrease the rectangle size.

This method is the opposite from Inflate(): Deflate(a, b) is equivalent to Inflate(-a, -b). Please refer to Inflate() for full description.

wxRect& wxRect::Deflate (wxCoord *diff*)

Decrease the rectangle size.

This method is the opposite from Inflate(): Deflate(a, b) is equivalent to Inflate(-a, -b). Please refer to Inflate() for full description.

wxRect wxRect::Deflate (wxCoord *dx*, wxCoord *dy*) const

Decrease the rectangle size.

This method is the opposite from Inflate(): Deflate(a, b) is equivalent to Inflate(-a, -b). Please refer to Inflate() for full description.

int wxRect::GetBottom ( ) const

Gets the bottom point of the rectangle.

wxPoint wxRect::GetBottomLeft ( ) const

Gets the position of the bottom left corner.

wxPoint wxRect::GetBottomRight ( ) const

Gets the position of the bottom right corner.

int wxRect::GetHeight ( ) const

Gets the height member.

int wxRect::GetLeft ( ) const

Gets the left point of the rectangle (the same as GetX()).

wxPoint wxRect::GetPosition ( ) const

Gets the position.

int wxRect::GetRight ( ) const

Gets the right point of the rectangle.

wxSize wxRect::GetSize ( ) const

Gets the size.

See also

SetSize()

int wxRect::GetTop ( ) const

Gets the top point of the rectangle (the same as GetY()).

wxPoint wxRect::GetTopLeft ( ) const

Gets the position of the top left corner of the rectangle, same as GetPosition().

wxPoint wxRect::GetTopRight ( ) const

Gets the position of the top right corner.

int wxRect::GetWidth ( ) const

Gets the width member.

int wxRect::GetX ( ) const

Gets the x member.

int wxRect::GetY ( ) const

Gets the y member.

wxRect& wxRect::Inflate (wxCoord *dx*, wxCoord *dy*)

Increases the size of the rectangle.

The left border is moved farther left and the right border is moved farther right by *dx*. The upper border is moved farther up and the bottom border is moved farther down by *dy*. (Note that the width and height of the rectangle thus change by 2∗dx and 2∗dy, respectively.) If one or both of *dx* and *dy* are negative, the opposite happens: the rectangle size decreases in the respective direction.

Inflating and deflating behaves "naturally". Defined more precisely, that means:

1. "Real" inflates (that is, *dx* and/or *dy* = 0) are not constrained. Thus inflating a rectangle can cause its upper left corner to move into the negative numbers. (2.5.4 and older forced the top left coordinate to not fall below (0, 0), which implied a forced move of the rectangle.)

2. Deflates are clamped to not reduce the width or height of the rectangle below zero. In such cases, the top-left corner is nonetheless handled properly. For example, a rectangle at (10, 10) with size (20, 40) that is inflated by (-15, -15) will become located at (20, 25) at size (0, 10). Finally, observe that the width and height are treated independently. In the above example, the width is reduced by 20, whereas the height is reduced by the full 30 (rather than also stopping at 20, when the width reached zero).

See also

Deflate()

wxRect& wxRect::Inflate (const wxSize & *diff*)

Increases the size of the rectangle.

The left border is moved farther left and the right border is moved farther right by *dx*. The upper border is moved farther up and the bottom border is moved farther down by *dy*. (Note that the width and height of the rectangle thus change by 2∗dx and 2∗dy, respectively.) If one or both of *dx* and *dy* are negative, the opposite happens: the rectangle size decreases in the respective direction.

Inflating and deflating behaves "naturally". Defined more precisely, that means:

1. "Real" inflates (that is, *dx* and/or *dy* = 0) are not constrained. Thus inflating a rectangle can cause its upper left corner to move into the negative numbers. (2.5.4 and older forced the top left coordinate to not fall below (0, 0), which implied a forced move of the rectangle.)
2. Deflates are clamped to not reduce the width or height of the rectangle below zero. In such cases, the top-left corner is nonetheless handled properly. For example, a rectangle at (10, 10) with size (20, 40) that is inflated by (-15, -15) will become located at (20, 25) at size (0, 10). Finally, observe that the width and height are treated independently. In the above example, the width is reduced by 20, whereas the height is reduced by the full 30 (rather than also stopping at 20, when the width reached zero).

See also

Deflate()

wxRect& wxRect::Inflate (wxCoord *diff*)

Increases the size of the rectangle.

The left border is moved farther left and the right border is moved farther right by *dx*. The upper border is moved farther up and the bottom border is moved farther down by *dy*. (Note that the width and height of the rectangle thus change by 2∗dx and 2∗dy, respectively.) If one or both of *dx* and *dy* are negative, the opposite happens: the rectangle size decreases in the respective direction.

Inflating and deflating behaves "naturally". Defined more precisely, that means:

1. "Real" inflates (that is, *dx* and/or *dy* = 0) are not constrained. Thus inflating a rectangle can cause its upper left corner to move into the negative numbers. (2.5.4 and older forced the top left coordinate to not fall below (0, 0), which implied a forced move of the rectangle.)
2. Deflates are clamped to not reduce the width or height of the rectangle below zero. In such cases, the top-left corner is nonetheless handled properly. For example, a rectangle at (10, 10) with size (20, 40) that is inflated by (-15, -15) will become located at (20, 25) at size (0, 10). Finally, observe that the width and height are treated independently. In the above example, the width is reduced by 20, whereas the height is reduced by the full 30 (rather than also stopping at 20, when the width reached zero).

See also

Deflate()

wxRect wxRect::Inflate (wxCoord *dx*, wxCoord *dy*) const

Increases the size of the rectangle.

The left border is moved farther left and the right border is moved farther right by *dx*. The upper border is moved farther up and the bottom border is moved farther down by *dy*. (Note that the width and height of the rectangle thus change by 2∗dx and 2∗dy, respectively.) If one or both of *dx* and *dy* are negative, the opposite happens: the rectangle size decreases in the respective direction.

Inflating and deflating behaves "naturally". Defined more precisely, that means:

1. "Real" inflates (that is, *dx* and/or *dy* = 0) are not constrained. Thus inflating a rectangle can cause its upper left corner to move into the negative numbers. (2.5.4 and older forced the top left coordinate to not fall below (0, 0), which implied a forced move of the rectangle.)

2. Deflates are clamped to not reduce the width or height of the rectangle below zero. In such cases, the top-left corner is nonetheless handled properly. For example, a rectangle at (10, 10) with size (20, 40) that is inflated by (-15, -15) will become located at (20, 25) at size (0, 10). Finally, observe that the width and height are treated independently. In the above example, the width is reduced by 20, whereas the height is reduced by the full 30 (rather than also stopping at 20, when the width reached zero).

See also

Deflate()

wxRect& wxRect::Intersect (const wxRect & *rect*)

Modifies this rectangle to contain the overlapping portion of this rectangle and the one passed in as parameter.

Returns

This rectangle, modified.

wxRect wxRect::Intersect (const wxRect & *rect*) const

Returns the overlapping portion of this rectangle and the one passed in as parameter.

bool wxRect::Intersects (const wxRect & *rect*) const

Returns true if this rectangle has a non-empty intersection with the rectangle *rect* and false otherwise.

bool wxRect::IsEmpty ( ) const

Returns true if this rectangle has a width or height less than or equal to 0 and false otherwise.

void wxRect::Offset (wxCoord *dx,* wxCoord *dy*)

Moves the rectangle by the specified offset.

If *dx* is positive, the rectangle is moved to the right, if *dy* is positive, it is moved to the bottom, otherwise it is moved to the left or top respectively.

void wxRect::Offset (const wxPoint & *pt*)

Moves the rectangle by the specified offset.

If *dx* is positive, the rectangle is moved to the right, if *dy* is positive, it is moved to the bottom, otherwise it is moved to the left or top respectively.

bool wxRect::operator!= (const wxRect & *r1,* const wxRect & *r2*)

Inequality operator.

wxRect wxRect::operator∗ (const wxRect & *r1*, const wxRect & *r2*)

Returns the intersection of two rectangles (which may be empty).

wxRect& wxRect::operator∗= (const wxRect & *r*)

Returns the intersection of two rectangles (which may be empty).

wxRect wxRect::operator+ (const wxRect & *r1*, const wxRect & *r2*)

Like Union(), but doesn't treat empty rectangles specially.

wxRect& wxRect::operator+= (const wxRect & *r*)

Like Union(), but doesn't treat empty rectangles specially.

wxRect& wxRect::operator= (const wxRect & *rect*)

Assignment operator.

bool wxRect::operator== (const wxRect & *r1*, const wxRect & *r2*)

Equality operator.

void wxRect::SetBottom (int *bottom*)

Set the bottom edge of the rectangle.

Notice that this doesn't affect GetTop() return value but changes the rectangle height to set its bottom side to the given position.

void wxRect::SetBottomLeft (const wxPoint & *p*)

Set the bottom-left point of the rectangle.

void wxRect::SetBottomRight (const wxPoint & *p*)

Set the bottom-right point of the rectangle.

void wxRect::SetHeight (int *height*)

Sets the height.

void wxRect::SetLeft (int *left*)

Set the left side of the rectangle.

Notice that because the rectangle stores its left side and width, calling SetLeft() changes the right side position too – but does preserve the width.

void wxRect::SetPosition (const wxPoint & *pos*)

Sets the position.

void wxRect::SetRight (int *right*)

Set the right side of the rectangle.

Notice that this doesn't affect GetLeft() return value but changes the rectangle width to set its right side to the given position.

void wxRect::SetSize (const wxSize & *s*)

Sets the size.

See also

GetSize()

void wxRect::SetTop (int *top*)

Set the top edge of the rectangle.

Notice that because the rectangle stores its top side and height, calling SetTop() changes the bottom side position too – but does preserve the height.

void wxRect::SetTopLeft (const wxPoint & *p*)

Set the top-left point of the rectangle.

void wxRect::SetTopRight (const wxPoint & *p*)

Set the top-right point of the rectangle.

void wxRect::SetWidth (int *width*)

Sets the width.

void wxRect::SetX (int *x*)

Sets the x position.

void wxRect::SetY (int *y*)

Sets the y position.

wxRect wxRect::Union (const wxRect & *rect*) const

Modifies the rectangle to contain the bounding box of this rectangle and the one passed in as parameter.

wxRect& wxRect::Union (const wxRect & *rect*)

Modifies the rectangle to contain the bounding box of this rectangle and the one passed in as parameter.

21.580.4 Member Data Documentation

int wxRect::height

Height member.

int wxRect::width

Width member.

int wxRect::x

x coordinate of the top-level corner of the rectangle.

int wxRect::y

y coordinate of the top-level corner of the rectangle.

21.581 wxRect2DDouble Class Reference

`#include <wx/geometry.h>`

Public Member Functions

- wxRect2DDouble ()
- wxRect2DDouble (wxDouble x, wxDouble y, wxDouble w, wxDouble h)
- wxPoint2DDouble GetPosition () const
- wxSize GetSize () const
- wxDouble GetLeft () const
- void SetLeft (wxDouble n)
- void MoveLeftTo (wxDouble n)
- wxDouble GetTop () const
- void SetTop (wxDouble n)
- void MoveTopTo (wxDouble n)
- wxDouble GetBottom () const
- void SetBottom (wxDouble n)
- void MoveBottomTo (wxDouble n)
- wxDouble GetRight () const
- void SetRight (wxDouble n)
- void MoveRightTo (wxDouble n)
- wxPoint2DDouble GetLeftTop () const
- void SetLeftTop (const wxPoint2DDouble &pt)
- void MoveLeftTopTo (const wxPoint2DDouble &pt)
- wxPoint2DDouble GetLeftBottom () const
- void SetLeftBottom (const wxPoint2DDouble &pt)
- void MoveLeftBottomTo (const wxPoint2DDouble &pt)
- wxPoint2DDouble GetRightTop () const

- void SetRightTop (const wxPoint2DDouble &pt)
- void MoveRightTopTo (const wxPoint2DDouble &pt)
- wxPoint2DDouble GetRightBottom () const
- void SetRightBottom (const wxPoint2DDouble &pt)
- void MoveRightBottomTo (const wxPoint2DDouble &pt)
- wxPoint2DDouble GetCentre () const
- void SetCentre (const wxPoint2DDouble &pt)
- void MoveCentreTo (const wxPoint2DDouble &pt)
- wxOutCode GetOutCode (const wxPoint2DDouble &pt) const
- wxOutCode GetOutcode (const wxPoint2DDouble &pt) const
- bool Contains (const wxPoint2DDouble &pt) const
- bool Contains (const wxRect2DDouble &rect) const
- bool IsEmpty () const
- bool HaveEqualSize (const wxRect2DDouble &rect) const
- void Inset (wxDouble x, wxDouble y)
- void Inset (wxDouble left, wxDouble top, wxDouble right, wxDouble bottom)
- void Offset (const wxPoint2DDouble &pt)
- void ConstrainTo (const wxRect2DDouble &rect)
- wxPoint2DDouble Interpolate (wxInt32 widthfactor, wxInt32 heightfactor)
- void Intersect (const wxRect2DDouble &otherRect)
- wxRect2DDouble CreateIntersection (const wxRect2DDouble &otherRect) const
- bool Intersects (const wxRect2DDouble &rect) const
- void Union (const wxRect2DDouble &otherRect)
- void Union (const wxPoint2DDouble &pt)
- wxRect2DDouble CreateUnion (const wxRect2DDouble &otherRect) const
- void Scale (wxDouble f)
- void Scale (wxInt32 num, wxInt32 denum)
- wxRect2DDouble & operator= (const wxRect2DDouble &rect)
- bool operator== (const wxRect2DDouble &rect) const
- bool operator!= (const wxRect2DDouble &rect) const

Static Public Member Functions

- static void Intersect (const wxRect2DDouble &src1, const wxRect2DDouble &src2, wxRect2DDouble *dest)
- static void Union (const wxRect2DDouble &src1, const wxRect2DDouble &src2, wxRect2DDouble *dest)

Public Attributes

- wxDouble m_x
- wxDouble m_y
- wxDouble m_width
- wxDouble m_height

21.581.1 Constructor & Destructor Documentation

wxRect2DDouble::wxRect2DDouble ( )

wxRect2DDouble::wxRect2DDouble (wxDouble *x*, wxDouble *y*, wxDouble *w*, wxDouble *h*)

21.581.2 Member Function Documentation

void wxRect2DDouble::ConstrainTo (const wxRect2DDouble & *rect*)

bool wxRect2DDouble::Contains (const **wxPoint2DDouble** & *pt*) const

bool wxRect2DDouble::Contains (const **wxRect2DDouble** & *rect*) const

wxRect2DDouble wxRect2DDouble::CreateIntersection (const **wxRect2DDouble** & *otherRect*) const

wxRect2DDouble wxRect2DDouble::CreateUnion (const **wxRect2DDouble** & *otherRect*) const

wxDouble wxRect2DDouble::GetBottom ( ) const

wxPoint2DDouble wxRect2DDouble::GetCentre ( ) const

wxDouble wxRect2DDouble::GetLeft ( ) const

wxPoint2DDouble wxRect2DDouble::GetLeftBottom ( ) const

wxPoint2DDouble wxRect2DDouble::GetLeftTop ( ) const

wxOutCode wxRect2DDouble::GetOutCode (const **wxPoint2DDouble** & *pt*) const

wxOutCode wxRect2DDouble::GetOutcode (const **wxPoint2DDouble** & *pt*) const

wxPoint2DDouble wxRect2DDouble::GetPosition ( ) const

wxDouble wxRect2DDouble::GetRight ( ) const

wxPoint2DDouble wxRect2DDouble::GetRightBottom ( ) const

wxPoint2DDouble wxRect2DDouble::GetRightTop ( ) const

wxSize wxRect2DDouble::GetSize ( ) const

wxDouble wxRect2DDouble::GetTop ( ) const

bool wxRect2DDouble::HaveEqualSize (const **wxRect2DDouble** & *rect*) const

void wxRect2DDouble::Inset (**wxDouble** *x*, **wxDouble** *y*)

void wxRect2DDouble::Inset (**wxDouble** *left*, **wxDouble** *top*, **wxDouble** *right*, **wxDouble** *bottom*)

wxPoint2DDouble wxRect2DDouble::Interpolate (**wxInt32** *widthfactor*, **wxInt32** *heightfactor*)

static void wxRect2DDouble::Intersect (const **wxRect2DDouble** & *src1*, const **wxRect2DDouble** & *src2*, **wxRect2DDouble** * *dest*) [static]

void wxRect2DDouble::Intersect (const **wxRect2DDouble** & *otherRect*)

bool wxRect2DDouble::Intersects (const **wxRect2DDouble** & *rect*) const

bool wxRect2DDouble::IsEmpty ( ) const

void wxRect2DDouble::MoveBottomTo (**wxDouble** *n*)

void wxRect2DDouble::MoveCentreTo (const **wxPoint2DDouble** & *pt*)

void wxRect2DDouble::MoveLeftBottomTo (const **wxPoint2DDouble** & *pt*)

void wxRect2DDouble::MoveLeftTo (wxDouble *n*)

void wxRect2DDouble::MoveLeftTopTo (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::MoveRightBottomTo (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::MoveRightTo (wxDouble *n*)

void wxRect2DDouble::MoveRightTopTo (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::MoveTopTo (wxDouble *n*)

void wxRect2DDouble::Offset (const wxPoint2DDouble & *pt*)

bool wxRect2DDouble::operator!= (const wxRect2DDouble & *rect*) const

wxRect2DDouble& wxRect2DDouble::operator= (const wxRect2DDouble & *rect*)

bool wxRect2DDouble::operator== (const wxRect2DDouble & *rect*) const

void wxRect2DDouble::Scale (wxDouble *f*)

void wxRect2DDouble::Scale (wxInt32 *num,* wxInt32 *denum*)

void wxRect2DDouble::SetBottom (wxDouble *n*)

void wxRect2DDouble::SetCentre (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::SetLeft (wxDouble *n*)

void wxRect2DDouble::SetLeftBottom (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::SetLeftTop (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::SetRight (wxDouble *n*)

void wxRect2DDouble::SetRightBottom (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::SetRightTop (const wxPoint2DDouble & *pt*)

void wxRect2DDouble::SetTop (wxDouble *n*)

static void wxRect2DDouble::Union (const wxRect2DDouble & *src1,* const wxRect2DDouble & *src2,* wxRect2DDouble * ***dest***) [static]

void wxRect2DDouble::Union (const wxRect2DDouble & *otherRect*)

void wxRect2DDouble::Union (const wxPoint2DDouble & *pt*)

21.581.3 Member Data Documentation

wxDouble wxRect2DDouble::m_height

wxDouble wxRect2DDouble::m_width

wxDouble wxRect2DDouble::m_x

wxDouble wxRect2DDouble::m_y

21.582 wxRect2DInt Class Reference

`#include <wx/geometry.h>`

Public Member Functions

- wxRect2DInt ()
- wxRect2DInt (const wxRect &r)
- wxRect2DInt (wxInt32 x, wxInt32 y, wxInt32 w, wxInt32 h)
- wxRect2DInt (const wxPoint2DInt &topLeft, const wxPoint2DInt &bottomRight)
- wxRect2DInt (const wxPoint2DInt &pos, const wxSize &size)
- wxRect2DInt (const wxRect2DInt &rect)
- wxPoint2DInt GetPosition () const
- wxSize GetSize () const
- wxInt32 GetLeft () const
- void SetLeft (wxInt32 n)
- void MoveLeftTo (wxInt32 n)
- wxInt32 GetTop () const
- void SetTop (wxInt32 n)
- void MoveTopTo (wxInt32 n)
- wxInt32 GetBottom () const
- void SetBottom (wxInt32 n)
- void MoveBottomTo (wxInt32 n)
- wxInt32 GetRight () const
- void SetRight (wxInt32 n)
- void MoveRightTo (wxInt32 n)
- wxPoint2DInt GetLeftTop () const
- void SetLeftTop (const wxPoint2DInt &pt)
- void MoveLeftTopTo (const wxPoint2DInt &pt)
- wxPoint2DInt GetLeftBottom () const
- void SetLeftBottom (const wxPoint2DInt &pt)
- void MoveLeftBottomTo (const wxPoint2DInt &pt)
- wxPoint2DInt GetRightTop () const
- void SetRightTop (const wxPoint2DInt &pt)
- void MoveRightTopTo (const wxPoint2DInt &pt)
- wxPoint2DInt GetRightBottom () const
- void SetRightBottom (const wxPoint2DInt &pt)
- void MoveRightBottomTo (const wxPoint2DInt &pt)
- wxPoint2DInt GetCentre () const
- void SetCentre (const wxPoint2DInt &pt)
- void MoveCentreTo (const wxPoint2DInt &pt)
- wxOutCode GetOutCode (const wxPoint2DInt &pt) const
- wxOutCode GetOutcode (const wxPoint2DInt &pt) const
- bool Contains (const wxPoint2DInt &pt) const
- bool Contains (const wxRect2DInt &rect) const
- bool IsEmpty () const
- bool HaveEqualSize (const wxRect2DInt &rect) const
- void Inset (wxInt32 x, wxInt32 y)
- void Inset (wxInt32 left, wxInt32 top, wxInt32 right, wxInt32 bottom)
- void Offset (const wxPoint2DInt &pt)
- void ConstrainTo (const wxRect2DInt &rect)

- wxPoint2DInt Interpolate (wxInt32 widthfactor, wxInt32 heightfactor)
- void Intersect (const wxRect2DInt &otherRect)
- wxRect2DInt CreateIntersection (const wxRect2DInt &otherRect) const
- bool Intersects (const wxRect2DInt &rect) const
- void Union (const wxRect2DInt &otherRect)
- void Union (const wxPoint2DInt &pt)
- wxRect2DInt CreateUnion (const wxRect2DInt &otherRect) const
- void Scale (wxInt32 f)
- void Scale (wxInt32 num, wxInt32 denum)
- wxRect2DInt & operator= (const wxRect2DInt &rect)
- bool operator== (const wxRect2DInt &rect) const
- bool operator!= (const wxRect2DInt &rect) const

Static Public Member Functions

- static void Intersect (const wxRect2DInt &src1, const wxRect2DInt &src2, wxRect2DInt ∗dest)
- static void Union (const wxRect2DInt &src1, const wxRect2DInt &src2, wxRect2DInt ∗dest)

Public Attributes

- wxInt32 m_x
- wxInt32 m_y
- wxInt32 m_width
- wxInt32 m_height

21.582.1 Constructor & Destructor Documentation

wxRect2DInt::wxRect2DInt ( )

wxRect2DInt::wxRect2DInt (const wxRect & *r*)

wxRect2DInt::wxRect2DInt (wxInt32 *x,* wxInt32 *y,* wxInt32 *w,* wxInt32 *h*)

wxRect2DInt::wxRect2DInt (const wxPoint2DInt & *topLeft,* const wxPoint2DInt & *bottomRight*)

wxRect2DInt::wxRect2DInt (const wxPoint2DInt & *pos,* const wxSize & *size*)

wxRect2DInt::wxRect2DInt (const wxRect2DInt & *rect*)

21.582.2 Member Function Documentation

void wxRect2DInt::ConstrainTo (const wxRect2DInt & *rect*)

bool wxRect2DInt::Contains (const wxPoint2DInt & *pt*) const

bool wxRect2DInt::Contains (const wxRect2DInt & *rect*) const

wxRect2DInt wxRect2DInt::CreateIntersection (const wxRect2DInt & *otherRect*) const

wxRect2DInt wxRect2DInt::CreateUnion (const wxRect2DInt & *otherRect*) const

wxInt32 wxRect2DInt::GetBottom ( ) const

wxPoint2DInt wxRect2DInt::GetCentre ( ) const

wxInt32 wxRect2DInt::GetLeft ( ) const

wxPoint2DInt wxRect2DInt::GetLeftBottom ( ) const

wxPoint2DInt wxRect2DInt::GetLeftTop ( ) const

wxOutCode wxRect2DInt::GetOutCode (const **wxPoint2DInt** & *pt*) const

wxOutCode wxRect2DInt::GetOutcode (const **wxPoint2DInt** & *pt*) const

wxPoint2DInt wxRect2DInt::GetPosition ( ) const

wxInt32 wxRect2DInt::GetRight ( ) const

wxPoint2DInt wxRect2DInt::GetRightBottom ( ) const

wxPoint2DInt wxRect2DInt::GetRightTop ( ) const

wxSize wxRect2DInt::GetSize ( ) const

wxInt32 wxRect2DInt::GetTop ( ) const

bool wxRect2DInt::HaveEqualSize (const **wxRect2DInt** & *rect*) const

void wxRect2DInt::Inset (**wxInt32** *x*, **wxInt32** *y*)

void wxRect2DInt::Inset (**wxInt32** *left*, **wxInt32** *top*, **wxInt32** *right*, **wxInt32** *bottom*)

wxPoint2DInt wxRect2DInt::Interpolate (**wxInt32** *widthfactor*, **wxInt32** *heightfactor*)

static void wxRect2DInt::Intersect (const **wxRect2DInt** & *src1*, const **wxRect2DInt** & *src2*, **wxRect2DInt** * *dest*) [static]

void wxRect2DInt::Intersect (const **wxRect2DInt** & *otherRect*)

bool wxRect2DInt::Intersects (const **wxRect2DInt** & *rect*) const

bool wxRect2DInt::IsEmpty ( ) const

void wxRect2DInt::MoveBottomTo (**wxInt32** *n*)

void wxRect2DInt::MoveCentreTo (const **wxPoint2DInt** & *pt*)

void wxRect2DInt::MoveLeftBottomTo (const **wxPoint2DInt** & *pt*)

void wxRect2DInt::MoveLeftTo (**wxInt32** *n*)

void wxRect2DInt::MoveLeftTopTo (const **wxPoint2DInt** & *pt*)

void wxRect2DInt::MoveRightBottomTo (const **wxPoint2DInt** & *pt*)

void wxRect2DInt::MoveRightTo (**wxInt32** *n*)

void wxRect2DInt::MoveRightTopTo (const **wxPoint2DInt** & *pt*)

void wxRect2DInt::MoveTopTo (wxInt32 *n*)

void wxRect2DInt::Offset (const wxPoint2DInt & *pt*)

bool wxRect2DInt::operator!= (const wxRect2DInt & *rect*) const

wxRect2DInt& wxRect2DInt::operator= (const wxRect2DInt & *rect*)

bool wxRect2DInt::operator== (const wxRect2DInt & *rect*) const

void wxRect2DInt::Scale (wxInt32 *f*)

void wxRect2DInt::Scale (wxInt32 *num,* wxInt32 *denum*)

void wxRect2DInt::SetBottom (wxInt32 *n*)

void wxRect2DInt::SetCentre (const wxPoint2DInt & *pt*)

void wxRect2DInt::SetLeft (wxInt32 *n*)

void wxRect2DInt::SetLeftBottom (const wxPoint2DInt & *pt*)

void wxRect2DInt::SetLeftTop (const wxPoint2DInt & *pt*)

void wxRect2DInt::SetRight (wxInt32 *n*)

void wxRect2DInt::SetRightBottom (const wxPoint2DInt & *pt*)

void wxRect2DInt::SetRightTop (const wxPoint2DInt & *pt*)

void wxRect2DInt::SetTop (wxInt32 *n*)

static void wxRect2DInt::Union (const wxRect2DInt & *src1,* const wxRect2DInt & *src2,* wxRect2DInt * *dest*) [static]

void wxRect2DInt::Union (const wxRect2DInt & *otherRect*)

void wxRect2DInt::Union (const wxPoint2DInt & *pt*)

21.582.3 Member Data Documentation

wxInt32 wxRect2DInt::m_height

wxInt32 wxRect2DInt::m_width

wxInt32 wxRect2DInt::m_x

wxInt32 wxRect2DInt::m_y

21.583 wxRecursionGuard Class Reference

#include <wx/recguard.h>

21.583.1 Detailed Description

wxRecursionGuard is a very simple class which can be used to prevent reentrancy problems in a function.

It is not thread-safe and so should be used only in single-threaded programs or in combination with some thread synchronization mechanisms.

wxRecursionGuard is always used together with the wxRecursionGuardFlag like in this example:

```
void Foo()
{
    static wxRecursionGuardFlag s_flag;
    wxRecursionGuard guard(s_flag);
    if ( guard.IsInside() )
    {
        // don't allow reentrancy
        return;
    }

    ...
}
```

As you can see, wxRecursionGuard simply tests the flag value and sets it to true if it hadn't been already set. IsInside() allows testing the old flag value. The advantage of using this class compared to directly manipulating the flag is that the flag is always reset in the wxRecursionGuard destructor and so you don't risk to forget to do it even if the function returns in an unexpected way (for example because an exception has been thrown).

Library: wxBase

Category: Miscellaneous

Public Member Functions

- wxRecursionGuard (wxRecursionGuardFlag &flag)

  A wxRecursionGuard object must always be initialized with a `static` wxRecursionGuardFlag.
- ~wxRecursionGuard ()

  The destructor resets the flag value so that the function can be entered again the next time.
- bool IsInside () const

  Returns true if we're already inside the code block "protected" by this wxRecursionGuard (i.e.

21.583.2 Constructor & Destructor Documentation

wxRecursionGuard::wxRecursionGuard (wxRecursionGuardFlag & *flag*)

A wxRecursionGuard object must always be initialized with a `static` wxRecursionGuardFlag.

The constructor saves the value of the flag to be able to return the correct value from IsInside().

wxRecursionGuard::~wxRecursionGuard ( )

The destructor resets the flag value so that the function can be entered again the next time.

Note

This is not virtual, so this class is not meant to be derived from (besides, there is absolutely no reason to do it anyhow).

21.583.3 Member Function Documentation

bool wxRecursionGuard::IsInside ( ) const

Returns true if we're already inside the code block "protected" by this wxRecursionGuard (i.e.

between this line and the end of current scope). Usually the function using wxRecursionGuard takes some specific actions in such case (may be simply returning) to prevent reentrant calls to itself.

If this method returns false, it is safe to continue.

21.584 wxRecursionGuardFlag Class Reference

`#include <wx/recguard.h>`

21.584.1 Detailed Description

This is a completely opaque class which exists only to be used with wxRecursionGuard, please see the example in that class' documentation.

Remarks

wxRecursionGuardFlag object must be declared `static` or the recursion would never be detected.

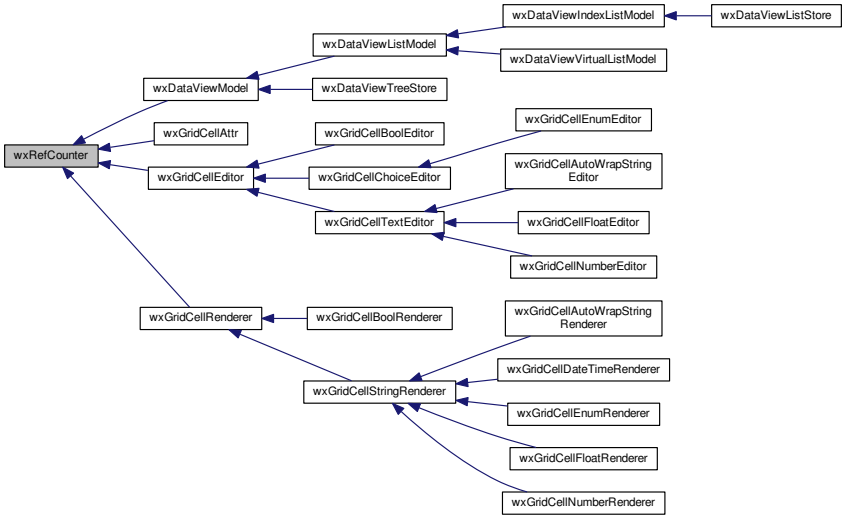
Library: wxBase

Category: Miscellaneous

21.585 wxRefCounter Class Reference

`#include <wx/object.h>`

Inheritance diagram for wxRefCounter:



21.585.1 Detailed Description

This class is used to manage reference-counting providing a simple interface and a counter.

wxRefCounter can be easily used together with wxObjectDataPtr<T> to ensure that no calls to wxRefCounter::↩ DecRef() are missed - thus avoiding memory leaks.

wxObjectRefData is a typedef to wxRefCounter and is used as the built-in reference counted storage for wxObject-derived classes.

Library: wxBase

Category: Runtime Type Information (RTTI)

See also

wxObject, wxObjectRefData, wxObjectDataPtr<T>, Reference Counting

Public Member Functions

- wxRefCounter ()

  Default constructor.

- void DecRef ()

  Decrements the reference count associated with this shared data and, if it reaches zero, destroys this instance of wxRefCounter releasing its memory.

- int GetRefCount () const

  Returns the reference count associated with this shared data.

- void IncRef ()

  Increments the reference count associated with this shared data.

Protected Member Functions

- virtual ~wxRefCounter ()

  Destructor.

21.585.2 Constructor & Destructor Documentation

virtual wxRefCounter::~wxRefCounter ( ) [protected], [virtual]

Destructor.

It's declared `protected` so that wxRefCounter instances will never be destroyed directly but only as result of a DecRef() call.

wxRefCounter::wxRefCounter ( )

Default constructor.

Initialises the internal reference count to 1.

21.585.3 Member Function Documentation

void wxRefCounter::DecRef ( )

Decrements the reference count associated with this shared data and, if it reaches zero, destroys this instance of wxRefCounter releasing its memory.

Please note that after calling this function, the caller should absolutely avoid to use the pointer to this instance since it may not be valid anymore.

int wxRefCounter::GetRefCount ( ) const

Returns the reference count associated with this shared data.

When this goes to zero during a DecRef() call, the object will auto-free itself.

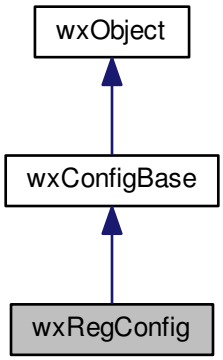
void wxRefCounter::IncRef ( )

Increments the reference count associated with this shared data.

21.586 wxRegConfig Class Reference

`#include <wx/msw/regconf.h>`

Inheritance diagram for wxRegConfig:



21.586.1 Detailed Description

wxRegConfig implements the wxConfigBase interface for storing and retrieving configuration information using Windows registry.

This class is used by default for wxConfig on Windows platforms; see wxFileConfig for an alternative you may want to use (also on Windows).

Library: wxBase

Category: Application and System configuration

See also

wxConfigBase

Public Member Functions

- wxRegConfig (const wxString &appName=wxEmptyString, const wxString &vendorName=wxEmptyString, const wxString &localFilename=wxEmptyString, const wxString &globalFilename=wxEmptyString, long style=wxCONFIG_USE_GLOBAL_FILE)

  The wxRegConfig constructor.

Additional Inherited Members

21.586.2 Constructor & Destructor Documentation

wxRegConfig::wxRegConfig (const wxString & *appName* = wxEmptyString, const wxString & *vendorName* = wxEmptyString, const wxString & *localFilename* = wxEmptyString, const wxString & *globalFilename* = wxEmptyString, long *style* = wxCONFIG_USE_GLOBAL_FILE)

The wxRegConfig constructor.

For more info see the docs for the wxConfigBase::wxConfigBase() constructor.

Note that wxRegConfig's *style* argument defaults to `wxCONFIG_USE_GLOBAL_FILE`, i.e. to the use of the `HKLM` key (also known as "HKEY_LOCAL_MACHINE").

21.587 wxRegEx Class Reference

`#include <wx/regex.h>`

21.587.1 Detailed Description

wxRegEx represents a regular expression.

This class provides support for regular expressions matching and also replacement.

It is built on top of either the system library (if it has support for POSIX regular expressions - which is the case of the most modern Unices) or uses the built in Henry Spencer's library. Henry Spencer would appreciate being given credit in the documentation of software which uses his library, but that is not a requirement.

Regular expressions, as defined by POSIX, come in two flavours: *extended* and *basic*. The builtin library also adds a third flavour of expression advanced, which is not available when using the system library.

Unicode is fully supported only when using the builtin library. When using the system library in Unicode mode, the expressions and data are translated to the default 8-bit encoding before being passed to the library.

On platforms where a system library is available, the default is to use the builtin library for Unicode builds, and the system library otherwise. It is possible to use the other if preferred by selecting it when building the wxWidgets.

Library: wxBase

Category: Data Structures

Examples:

A bad example of processing some text containing email addresses (the example is bad because the real email addresses can have more complicated form than `user@host.net`):

```
wxString text;
...
wxRegEx reEmail = "([^@]+)@([[:alnum:].-_].)+([[:alnum:]]+)";
if ( reEmail.Matches(text) )
{
    wxString text = reEmail.GetMatch(email);
    wxString username = reEmail.GetMatch(email, 1);
    if ( reEmail.GetMatch(email, 3) == "com" ) // .com TLD?
    {
        ...
    }
}

// or we could do this to hide the email address
size_t count = reEmail.ReplaceAll(text, "HIDDEN@\\2\\3");
printf("text now contains %u hidden addresses", count);
```

Public Member Functions

- wxRegEx ()

  Default constructor: use Compile() later.
- wxRegEx (const wxString &expr, int flags=wxRE_DEFAULT)

  Create and compile the regular expression, use IsValid() to test for compilation errors.
- ~wxRegEx ()

Destructor.

- bool Compile (const wxString &pattern, int flags=wxRE_DEFAULT)

  Compile the string into regular expression, return true if ok or false if string has a syntax error.
- bool GetMatch (size_t *start, size_t *len, size_t index=0) const

  Get the start index and the length of the match of the expression (if index is 0) or a bracketed subexpression (index different from 0).
- wxString GetMatch (const wxString &text, size_t index=0) const

  Returns the part of string corresponding to the match where index is interpreted as above.
- size_t GetMatchCount () const

  Returns the size of the array of matches, i.e. the number of bracketed subexpressions plus one for the expression itself, or 0 on error.
- bool IsValid () const

  Return true if this is a valid compiled regular expression, false otherwise.
- bool Matches (const wxString &text, int flags=0) const

  Matches the precompiled regular expression against the string text, returns true if matches and false otherwise.
- int Replace (wxString *text, const wxString &replacement, size_t maxMatches=0) const

  Replaces the current regular expression in the string pointed to by text, with the text in replacement and return number of matches replaced (maybe 0 if none found) or -1 on error.
- int ReplaceAll (wxString *text, const wxString &replacement) const

  Replace all occurrences: this is actually a synonym for Replace().
- int ReplaceFirst (wxString *text, const wxString &replacement) const

  Replace the first occurrence.

- bool Matches (const wxChar *text, int flags=0) const

  Matches the precompiled regular expression against the string text, returns true if matches and false otherwise.
- bool Matches (const wxChar *text, int flags, size_t len) const

  Matches the precompiled regular expression against the string text, returns true if matches and false otherwise.

21.587.2 Constructor & Destructor Documentation

wxRegEx::wxRegEx ( )

Default constructor: use Compile() later.

wxRegEx::wxRegEx (const wxString & *expr,* int *flags* = wxRE_DEFAULT)

Create and compile the regular expression, use IsValid() to test for compilation errors.

As for the flags, please see wxRE_FLAGS.

wxRegEx::~wxRegEx ( )

Destructor.

It's not virtual, don't derive from this class.

21.587.3 Member Function Documentation

bool wxRegEx::Compile (const wxString & *pattern,* int *flags* = wxRE_DEFAULT)

Compile the string into regular expression, return true if ok or false if string has a syntax error.

As for the flags, please see wxRE_FLAGS.

bool wxRegEx::GetMatch (size_t * *start*, size_t * *len*, size_t *index* = 0) const

Get the start index and the length of the match of the expression (if *index* is 0) or a bracketed subexpression (*index* different from 0).

May only be called after successful call to Matches() and only if `wxRE_NOSUB` was **not** used in Compile().

Returns false if no match or if an error occurred.

wxString wxRegEx::GetMatch (const wxString & *text*, size_t *index* = 0) const

Returns the part of string corresponding to the match where index is interpreted as above.

Empty string is returned if match failed.

May only be called after successful call to Matches() and only if `wxRE_NOSUB` was **not** used in Compile().

size_t wxRegEx::GetMatchCount ( ) const

Returns the size of the array of matches, i.e. the number of bracketed subexpressions plus one for the expression itself, or 0 on error.

May only be called after successful call to Compile(). and only if `wxRE_NOSUB` was **not** used.

bool wxRegEx::IsValid ( ) const

Return true if this is a valid compiled regular expression, false otherwise.

bool wxRegEx::Matches (const wxChar * *text*, int *flags* = 0) const

Matches the precompiled regular expression against the string *text*, returns true if matches and false otherwise.

Flags may be combination of `wxRE_NOTBOL` and `wxRE_NOTEOL`, see wxRE_NOT_FLAGS.

Some regex libraries assume that the text given is null terminated, while others require the length be given as a separate parameter. Therefore for maximum portability assume that *text* cannot contain embedded nulls.

When the **Matches(const wxChar *text, int flags = 0)** form is used, a wxStrlen() will be done internally if the regex library requires the length. When using Matches() in a loop the **Matches(text, flags, len)** form can be used instead, making it possible to avoid a wxStrlen() inside the loop.

May only be called after successful call to Compile().

bool wxRegEx::Matches (const wxChar * *text*, int *flags*, size_t *len*) const

Matches the precompiled regular expression against the string *text*, returns true if matches and false otherwise.

Flags may be combination of `wxRE_NOTBOL` and `wxRE_NOTEOL`, see wxRE_NOT_FLAGS.

Some regex libraries assume that the text given is null terminated, while others require the length be given as a separate parameter. Therefore for maximum portability assume that *text* cannot contain embedded nulls.

When the **Matches(const wxChar *text, int flags = 0)** form is used, a wxStrlen() will be done internally if the regex library requires the length. When using Matches() in a loop the **Matches(text, flags, len)** form can be used instead, making it possible to avoid a wxStrlen() inside the loop.

May only be called after successful call to Compile().

bool wxRegEx::Matches (const wxString & *text*, int *flags* = 0) const

Matches the precompiled regular expression against the string *text*, returns true if matches and false otherwise.

Flags may be combination of `wxRE_NOTBOL` and `wxRE_NOTEOL`, see wxRE_NOT_FLAGS.

May only be called after successful call to Compile().

int wxRegEx::Replace (wxString * *text*, const wxString & *replacement*, size_t *maxMatches* = `0`) const

Replaces the current regular expression in the string pointed to by *text*, with the text in *replacement* and return number of matches replaced (maybe 0 if none found) or -1 on error.

The replacement text may contain back references `\number` which will be replaced with the value of the corresponding subexpression in the pattern match. `\0` corresponds to the entire match and `&` is a synonym for it. Backslash may be used to quote itself or `&` character.

maxMatches may be used to limit the number of replacements made, setting it to 1, for example, will only replace first occurrence (if any) of the pattern in the text while default value of 0 means replace all.

int wxRegEx::ReplaceAll (wxString * *text*, const wxString & *replacement*) const

Replace all occurrences: this is actually a synonym for Replace().

See also

ReplaceFirst()

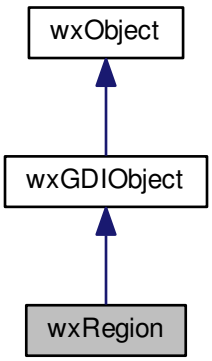
int wxRegEx::ReplaceFirst (wxString * *text*, const wxString & *replacement*) const

Replace the first occurrence.

21.588 wxRegion Class Reference

`#include <wx/region.h>`

Inheritance diagram for wxRegion:



21.588.1 Detailed Description

A wxRegion represents a simple or complex region on a device context or window.

This class uses reference counting and copy-on-write internally so that assignments between two instances of this class are very cheap. You can therefore use actual objects instead of pointers without efficiency problems. If an instance of this class is changed it will create its own data internally so that other instances, which previously shared the data using the reference counting, are not affected.

Predefined objects/pointers:

- wxNullRegion

Library: wxCore

Category: Data Structures, Graphics Device Interface (GDI)

See also

wxRegionIterator

Public Member Functions

- wxRegion ()

  Default constructor.
- wxRegion (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Constructs a rectangular region with the given position and size.
- wxRegion (const wxPoint &topLeft, const wxPoint &bottomRight)

  Constructs a rectangular region from the top left point and the bottom right point.
- wxRegion (const wxRect &rect)

  Constructs a rectangular region a wxRect object.
- wxRegion (const wxRegion &region)

  Copy constructor, uses Reference Counting.
- wxRegion (size_t n, const wxPoint *points, wxPolygonFillMode fillStyle=wxODDEVEN_RULE)

  Constructs a region corresponding to the polygon made of n points in the provided array.
- wxRegion (const wxBitmap &bmp)

  Constructs a region using a bitmap.
- wxRegion (const wxBitmap &bmp, const wxColour &transColour, int tolerance=0)

  Constructs a region using the non-transparent pixels of a bitmap.
- virtual ~wxRegion ()

  Destructor.
- virtual void Clear ()

  Clears the current region.
- wxRegionContain Contains (wxCoord x, wxCoord y) const

  Returns a value indicating whether the given point is contained within the region.
- wxRegionContain Contains (const wxPoint &pt) const

  Returns a value indicating whether the given point is contained within the region.
- wxRegionContain Contains (wxCoord x, wxCoord y, wxCoord width, wxCoord height) const

  Returns a value indicating whether the given rectangle is contained within the region.
- wxRegionContain Contains (const wxRect &rect) const

  Returns a value indicating whether the given rectangle is contained within the region.
- wxBitmap ConvertToBitmap () const

Convert the region to a black and white bitmap with the white pixels being inside the region.

- bool Intersect (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Finds the intersection of this region and another, rectangular region, specified using position and size.
- bool Intersect (const wxRect &rect)

  Finds the intersection of this region and another, rectangular region.
- bool Intersect (const wxRegion &region)

  Finds the intersection of this region and another region.
- virtual bool IsEmpty () const

  Returns true if the region is empty, false otherwise.
- bool IsEqual (const wxRegion &region) const

  Returns true if the region is equal to, i.e. covers the same area as, another one.
- bool Subtract (const wxRect &rect)

  Subtracts a rectangular region from this region.
- bool Subtract (const wxRegion &region)

  Subtracts a region from this region.
- bool Union (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Finds the union of this region and another, rectangular region, specified using position and size.
- bool Union (const wxRect &rect)

  Finds the union of this region and another, rectangular region.
- bool Union (const wxRegion &region)

  Finds the union of this region and another region.
- bool Union (const wxBitmap &bmp)

  Finds the union of this region and the non-transparent pixels of a bitmap.
- bool Union (const wxBitmap &bmp, const wxColour &transColour, int tolerance=0)

  Finds the union of this region and the non-transparent pixels of a bitmap.
- bool Xor (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Finds the Xor of this region and another, rectangular region, specified using position and size.
- bool Xor (const wxRect &rect)

  Finds the Xor of this region and another, rectangular region.
- bool Xor (const wxRegion &region)

  Finds the Xor of this region and another region.
- wxRegion & operator= (const wxRegion &region)

  Assignment operator, using Reference Counting.

- void GetBox (wxCoord &x, wxCoord &y, wxCoord &width, wxCoord &height) const

  Returns the outer bounds of the region.
- wxRect GetBox () const

  Returns the outer bounds of the region.

- bool Offset (wxCoord x, wxCoord y)

  Moves the region by the specified offsets in horizontal and vertical directions.
- bool Offset (const wxPoint &pt)

  Moves the region by the specified offsets in horizontal and vertical directions.

Additional Inherited Members

21.588.2 Constructor & Destructor Documentation

wxRegion::wxRegion ( )

Default constructor.

This constructor creates an invalid, or null, object, i.e. calling IsOk() on it returns false and IsEmpty() returns true.

wxRegion::wxRegion (wxCoord *x*, wxCoord *y*, wxCoord *width*, wxCoord *height*)

Constructs a rectangular region with the given position and size.

wxRegion::wxRegion (const wxPoint & *topLeft*, const wxPoint & *bottomRight*)

Constructs a rectangular region from the top left point and the bottom right point.

wxRegion::wxRegion (const wxRect & *rect*)

Constructs a rectangular region a wxRect object.

wxRegion::wxRegion (const wxRegion & *region*)

Copy constructor, uses Reference Counting.

wxRegion::wxRegion (size_t *n*, const wxPoint * *points*, wxPolygonFillMode *fillStyle* = wxODDEVEN_RULE)

Constructs a region corresponding to the polygon made of *n* points in the provided array.

fillStyle parameter may have values `wxWINDING_RULE` or `wxODDEVEN_RULE`.

wxRegion::wxRegion (const wxBitmap & *bmp*)

Constructs a region using a bitmap.

See Union() for more details.

wxRegion::wxRegion (const wxBitmap & *bmp*, const wxColour & *transColour*, int *tolerance* = 0)

Constructs a region using the non-transparent pixels of a bitmap.

See Union() for more details.

virtual wxRegion::~wxRegion ( ) [virtual]

Destructor.

See reference-counted object destruction for more info.

21.588.3 Member Function Documentation

virtual void wxRegion::Clear ( ) [virtual]

Clears the current region.

The object becomes invalid, or null, after being cleared.

wxRegionContain wxRegion::Contains (wxCoord *x*, wxCoord *y*) const

Returns a value indicating whether the given point is contained within the region.

This method always returns `wxOutRegion` for an invalid region but may, nevertheless, be safely called in this case.

Returns

The return value is one of `wxOutRegion` and `wxInRegion`.

wxRegionContain wxRegion::Contains (const wxPoint & *pt*) const

Returns a value indicating whether the given point is contained within the region.

This method always returns `wxOutRegion` for an invalid region but may, nevertheless, be safely called in this case.

Returns

The return value is one of `wxOutRegion` and `wxInRegion`.

wxRegionContain wxRegion::Contains (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*) const

Returns a value indicating whether the given rectangle is contained within the region.

This method always returns `wxOutRegion` for an invalid region but may, nevertheless, be safely called in this case.

Returns

One of wxOutRegion, wxPartRegion or wxInRegion.

Note

On Windows, only wxOutRegion and wxInRegion are returned; a value wxInRegion then indicates that all or some part of the region is contained in this region.

wxRegionContain wxRegion::Contains (const wxRect & *rect*) const

Returns a value indicating whether the given rectangle is contained within the region.

This method always returns `wxOutRegion` for an invalid region but may, nevertheless, be safely called in this case.

Returns

One of wxOutRegion, wxPartRegion or wxInRegion.

Note

On Windows, only wxOutRegion and wxInRegion are returned; a value wxInRegion then indicates that all or some part of the region is contained in this region.

wxBitmap wxRegion::ConvertToBitmap ( ) const

Convert the region to a black and white bitmap with the white pixels being inside the region.

This method can't be used for invalid region.

void wxRegion::GetBox (wxCoord & *x,* wxCoord & *y,* wxCoord & *width,* wxCoord & *height*) const

Returns the outer bounds of the region.

This method returns 0-sized bounding box for invalid regions.

wxRect wxRegion::GetBox ( ) const

Returns the outer bounds of the region.

This method returns 0-sized bounding box for invalid regions.

bool wxRegion::Intersect (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Finds the intersection of this region and another, rectangular region, specified using position and size.

This method always fails, i.e. returns false, if this region is invalid but may nevertheless be safely used even in this case.

Returns

true if successful, false otherwise.

Remarks

Creates the intersection of the two regions, that is, the parts which are in both regions. The result is stored in this region.

bool wxRegion::Intersect (const wxRect & *rect*)

Finds the intersection of this region and another, rectangular region.

This method always fails, i.e. returns false, if this region is invalid but may nevertheless be safely used even in this case.

Returns

true if successful, false otherwise.

Remarks

Creates the intersection of the two regions, that is, the parts which are in both regions. The result is stored in this region.

bool wxRegion::Intersect (const wxRegion & *region*)

Finds the intersection of this region and another region.

This method always fails, i.e. returns false, if this region is invalid but may nevertheless be safely used even in this case.

Returns

true if successful, false otherwise.

Remarks

Creates the intersection of the two regions, that is, the parts which are in both regions. The result is stored in this region.

virtual bool wxRegion::IsEmpty ( ) const `[virtual]`

Returns true if the region is empty, false otherwise.

Always returns true if the region is invalid.

bool wxRegion::IsEqual (const wxRegion & *region*) const

Returns true if the region is equal to, i.e. covers the same area as, another one.

If both this region and *region* are both invalid, they are considered to be equal.

bool wxRegion::Offset (wxCoord *x*, wxCoord *y*)

Moves the region by the specified offsets in horizontal and vertical directions.

This method can't be called if the region is invalid as it doesn't make sense to offset it then. Attempts to do it will result in assert failure.

Returns

true if successful, false otherwise (the region is unchanged then).

bool wxRegion::Offset (const wxPoint & *pt*)

Moves the region by the specified offsets in horizontal and vertical directions.

This method can't be called if the region is invalid as it doesn't make sense to offset it then. Attempts to do it will result in assert failure.

Returns

true if successful, false otherwise (the region is unchanged then).

wxRegion& wxRegion::operator= (const wxRegion & *region*)

Assignment operator, using Reference Counting.

bool wxRegion::Subtract (const wxRect & *rect*)

Subtracts a rectangular region from this region.

This method always fails, i.e. returns false, if this region is invalid but may nevertheless be safely used even in this case.

Returns

true if successful, false otherwise.

Remarks

This operation combines the parts of 'this' region that are not part of the second region. The result is stored in this region.

bool wxRegion::Subtract (const wxRegion & *region*)

Subtracts a region from this region.

This method always fails, i.e. returns false, if this region is invalid but may nevertheless be safely used even in this case.

Returns

true if successful, false otherwise.

Remarks

This operation combines the parts of 'this' region that are not part of the second region. The result is stored in this region.

bool wxRegion::Union (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Finds the union of this region and another, rectangular region, specified using position and size.

This method can be used even if this region is invalid and has the natural behaviour in this case, i.e. makes this region equal to the given rectangle.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

bool wxRegion::Union (const wxRect & *rect*)

Finds the union of this region and another, rectangular region.

This method can be used even if this region is invalid and has the natural behaviour in this case, i.e. makes this region equal to the given rectangle.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

bool wxRegion::Union (const wxRegion & *region*)

Finds the union of this region and another region.

This method can be used even if this region is invalid and has the natural behaviour in this case, i.e. makes this region equal to the given *region*.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

bool wxRegion::Union (const wxBitmap & *bmp*)

Finds the union of this region and the non-transparent pixels of a bitmap.

The bitmap's mask is used to determine transparency. If the bitmap doesn't have a mask, the bitmap's full dimensions are used.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

bool wxRegion::Union (const wxBitmap & *bmp,* const wxColour & *transColour,* int *tolerance* = 0)

Finds the union of this region and the non-transparent pixels of a bitmap.

Colour to be treated as transparent is specified in the *transColour* argument, along with an optional colour tolerance value.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region. The result is stored in this region.

bool wxRegion::Xor (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Finds the Xor of this region and another, rectangular region, specified using position and size.

This method can be used even if this region is invalid and has the natural behaviour in this case, i.e. makes this region equal to the given rectangle.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region, except for any overlapping areas. The result is stored in this region.

bool wxRegion::Xor (const wxRect & *rect*)

Finds the Xor of this region and another, rectangular region.

This method can be used even if this region is invalid and has the natural behaviour in this case, i.e. makes this region equal to the given rectangle.

Returns

true if successful, false otherwise.

Remarks

This operation creates a region that combines all of this region and the second region, except for any overlapping areas. The result is stored in this region.

bool wxRegion::Xor (const wxRegion & *region*)

Finds the Xor of this region and another region.

This method can be used even if this region is invalid and has the natural behaviour in this case, i.e. makes this region equal to the given *region*.

Returns

true if successful, false otherwise.

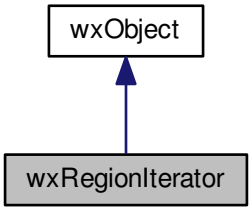
Remarks

This operation creates a region that combines all of this region and the second region, except for any overlapping areas. The result is stored in this region.

21.589 wxRegionIterator Class Reference

`#include <wx/region.h>`

Inheritance diagram for wxRegionIterator:



21.589.1 Detailed Description

This class is used to iterate through the rectangles in a region, typically when examining the damaged regions of a window within an OnPaint call.

To use it, construct an iterator object on the stack and loop through the regions, testing the object and incrementing the iterator at the end of the loop.

See wxPaintEvent for an example of use.

Library: wxCore

Category: Graphics Device Interface (GDI)

Predefined objects/pointers: wxNullRegion

See also

wxPaintEvent

Public Member Functions

- wxRegionIterator ()
  Default constructor.
- wxRegionIterator (const wxRegion &region)
  Creates an iterator object given a region.
- wxCoord GetH () const
  An alias for GetHeight().
- wxCoord GetHeight () const
  Returns the height value for the current region.
- wxRect GetRect () const
  Returns the current rectangle.
- wxCoord GetW () const
  An alias for GetWidth().
- wxCoord GetWidth () const
  Returns the width value for the current region.
- wxCoord GetX () const
  Returns the x value for the current region.
- wxCoord GetY () const
  Returns the y value for the current region.
- bool HaveRects () const
  Returns true if there are still some rectangles; otherwise returns false.
- void Reset ()
  Resets the iterator to the beginning of the rectangles.
- void Reset (const wxRegion &region)
  Resets the iterator to the given region.
- wxRegionIterator & operator++ ()
  Increment operator.
- operator bool () const
  Returns true if there are still some rectangles; otherwise returns false.

Additional Inherited Members

21.589.2 Constructor & Destructor Documentation

wxRegionIterator::wxRegionIterator ( )

Default constructor.

wxRegionIterator::wxRegionIterator (const wxRegion & *region*)

Creates an iterator object given a region.

21.589.3 Member Function Documentation

wxCoord wxRegionIterator::GetH ( ) const

An alias for GetHeight().

wxCoord wxRegionIterator::GetHeight ( ) const

Returns the height value for the current region.

wxRect wxRegionIterator::GetRect ( ) const

Returns the current rectangle.

wxCoord wxRegionIterator::GetW ( ) const

An alias for GetWidth().

wxCoord wxRegionIterator::GetWidth ( ) const

Returns the width value for the current region.

wxCoord wxRegionIterator::GetX ( ) const

Returns the x value for the current region.

wxCoord wxRegionIterator::GetY ( ) const

Returns the y value for the current region.

bool wxRegionIterator::HaveRects ( ) const

Returns true if there are still some rectangles; otherwise returns false.

wxRegionIterator::operator bool ( ) const

Returns true if there are still some rectangles; otherwise returns false.

You can use this to test the iterator object as if it were of type `bool`.

wxRegionIterator& wxRegionIterator::operator++ ( )

Increment operator.

Increments the iterator to the next region.

void wxRegionIterator::Reset ( )

Resets the iterator to the beginning of the rectangles.

void wxRegionIterator::Reset (const wxRegion & *region*)

Resets the iterator to the given region.

21.590 wxRegKey Class Reference

`#include <wx/msw/registry.h>`

21.590.1 Detailed Description

wxRegKey is a class representing the Windows registry (it is only available under Windows).

One can create, query and delete registry keys using this class.

The Windows registry is easy to understand. There are five registry keys, namely:

- `HKEY_CLASSES_ROOT` (HKCR)
- `HKEY_CURRENT_USER` (HKCU)
- `HKEY_LOCAL_MACHINE` (HKLM)
- `HKEY_CURRENT_CONFIG` (HKCC)
- `HKEY_USERS` (HKU)

After creating a key, it can hold a value. The values can be:

- String Value
- Binary Value
- DWORD Value
- Multi String Value
- Expandable String Value

Availability: only available for the wxMSW port.

Example:

```
// This assume that the key already exists, use HasSubKey() to check
// for the key existence if necessary.
wxRegKey key(wxRegKey::HKLM, "Software\\MyKey");

// Create a new value "MyValue" and set it to 12.
key.SetValue("MyValue", 12);

// Read the value back.
long value;
key.QueryValue("MyValue", &value);
wxMessageBox(wxString::Format("%d", value), "Registry Value",
      wxOK);

// Get the number of subkeys and enumerate them.
size_t subkeys;
key.GetKeyInfo(&subkeys, NULL, NULL, NULL);

wxString key_name;
key.GetFirstKey(key_name, 1);
for(int i = 0; i < subkeys; i++)
{
    wxMessageBox(key_name, "Subkey Name", wxOK);
    key.GetNextKey(key_name, 1);
}
```

Library: wxBase

Category: Application and System configuration

Public Types

- enum AccessMode {
  Read,
  Write }

  Access modes for wxRegKey.
- enum StdKey {
  HKCR,
  HKCU,
  HKLM,
  HKUSR,
  HKPD,
  HKCC,
  HKDD,
  HKMAX }

  The standard registry key enumerator.
- enum ValueType {
  Type_None,
  Type_String,
  Type_Expand_String,
  Type_Binary,
  Type_Dword,
  Type_Dword_little_endian,
  Type_Dword_big_endian,
  Type_Link,
  Type_Multi_String,
  Type_Resource_list,
  Type_Full_resource_descriptor,
  Type_Resource_requirements_list }

  The value type enumerator.
- enum WOW64ViewMode {
  WOW64ViewMode_Default,
  WOW64ViewMode_32,
  WOW64ViewMode_64 }

  Used to determine how the registry will be viewed, either as 32-bit or 64-bit.

Public Member Functions

- wxRegKey (WOW64ViewMode viewMode=WOW64ViewMode_Default)

  Default constructor, initializes to `HKEY_CLASSES_ROOT`.
- wxRegKey (const wxString &strKey, WOW64ViewMode viewMode=WOW64ViewMode_Default)

  The constructor to set the full name of the key.
- wxRegKey (StdKey keyParent, const wxString &strKey, WOW64ViewMode viewMode=WOW64ViewMode_Default)

  The constructor to set the full name of the key using one of the standard keys, that is, HKCR, HKCU, HKLM, HKUSR, HKPD, HKCC or HKDD.
- wxRegKey (const wxRegKey &keyParent, const wxString &strKey)

  The constructor to set the full name of the key under a previously created parent.
- void Close ()

Closes the key.

- bool Copy (const wxString &szNewName)

  Copy the entire contents of the key recursively to another location using the name.
- bool Copy (wxRegKey &keyDst)

  Copy the entire contents of the key recursively to another location using the key.
- bool CopyValue (const wxString &szValue, wxRegKey &keyDst, const wxString &szNewName=wxEmptyString)

  Copy the value to another key, possibly changing its name.
- bool Create (bool bOkIfExists=true)

  Creates the key.
- void DeleteKey (const wxString &szKey)

  Deletes the subkey with all its subkeys and values recursively.
- void DeleteSelf ()

  Deletes this key and all its subkeys and values recursively.
- void DeleteValue (const wxString &szKey)

  Deletes the named value or use an empty string argument to remove the default value of the key.
- bool Exists () const

  Returns true if the key exists.
- bool Export (const wxString &filename) const

  Write the contents of this key and all its subkeys to the given file.
- bool Export (wxOutputStream &ostr) const

  Write the contents of this key and all its subkeys to the opened stream.
- bool GetFirstKey (wxString &strKeyName, long &lIndex)

  Gets the first key.
- bool GetFirstValue (wxString &strValueName, long &lIndex)

  Gets the first value of this key.
- bool GetKeyInfo (size_t *pnSubKeys, size_t *pnMaxKeyLen, size_t *pnValues, size_t *pnMaxValueLen) const

  Gets information about the key.
- wxString GetName (bool bShortPrefix=true) const

  Gets the name of the registry key.
- WOW64ViewMode GetView () const

  Retrieves the registry view used by this key.
- bool GetNextKey (wxString &strKeyName, long &lIndex) const

  Gets the next key.
- bool GetNextValue (wxString &strValueName, long &lIndex) const

  Gets the next key value for this key.
- ValueType GetValueType (const wxString &szValue) const

  Gets the value type.
- bool HasSubKey (const wxString &szKey) const

  Returns true if given subkey exists.
- bool HasSubkeys () const

  Returns true if any subkeys exist.
- bool HasValue (const wxString &szValue) const

  Returns true if the value exists.
- bool HasValues () const

  Returns true if any values exist.
- bool IsEmpty () const

  Returns true if this key is empty, nothing under this key.
- bool IsNumericValue (const wxString &szValue) const

  Returns true if the value contains a number.

- bool IsOpened () const

  Returns true if the key is opened.
- bool Open (AccessMode mode=Write)

  Explicitly opens the key.
- wxRegKey & operator= (const wxString &strValue)

  Assignment operator to set the default value of the key.
- wxString QueryDefaultValue () const

  Return the default value of the key.
- bool QueryRawValue (const wxString &szValue, wxString &strValue) const

  Retrieves the raw string value.
- bool QueryValue (const wxString &szValue, wxString &strValue, bool raw) const

  Retrieves the raw or expanded string value.
- bool QueryValue (const wxString &szValue, long *plValue) const

  Retrieves the numeric value.
- bool QueryValue (const wxString &szValue, wxMemoryBuffer &buf) const

  Retrieves the binary structure.
- bool Rename (const wxString &szNewName)

  Renames the key.
- bool RenameValue (const wxString &szValueOld, const wxString &szValueNew)

  Renames a value.
- void ReserveMemoryForName (size_t bytes)

  Preallocate some memory for the name.
- void SetHkey (WXHKEY hKey)

  Set or change the HKEY handle.
- void SetName (const wxString &strKey)

  Set the full key name.
- void SetName (StdKey keyParent, const wxString &strKey)

  Set the name relative to the parent key.
- void SetName (const wxRegKey &keyParent, const wxString &strKey)

  Set the name relative to the parent key.
- bool SetValue (const wxString &szValue, long lValue)

  Sets the given szValue which must be numeric.
- bool SetValue (const wxString &szValue, const wxString &strValue)

  Sets the given szValue which must be string.
- bool SetValue (const wxString &szValue, const wxMemoryBuffer &buf)

  Sets the given szValue which must be binary.

21.590.2 Member Enumeration Documentation

enum wxRegKey::AccessMode

Access modes for wxRegKey.

Enumerator

Read Read-only.

Write Read and Write.

enum wxRegKey::StdKey

The standard registry key enumerator.

Enumerator

HKCR HKEY_CLASSES_ROOT.

HKCU HKEY_CURRENT_USER.

HKLM HKEY_LOCAL_MACHINE.

HKUSR HKEY_USERS.

HKPD HKEY_PERFORMANCE_DATA (Windows NT and 2K only)

HKCC HKEY_CURRENT_CONFIG.

HKDD HKEY_DYN_DATA (Windows 95 and 98 only)

HKMAX

enum wxRegKey::ValueType

The value type enumerator.

Enumerator

Type_None No value type.

Type_String Unicode null-terminated string.

Type_Expand_String Unicode null-terminated string (with environment variable references)

Type_Binary Free form binary.

Type_Dword 32-bit number

Type_Dword_little_endian 32-bit number (same as Type_Dword)

Type_Dword_big_endian 32-bit number

Type_Link Symbolic Link (Unicode)

Type_Multi_String Multiple Unicode strings.

Type_Resource_list Resource list in the resource map.

Type_Full_resource_descriptor Resource list in the hardware description.

Type_Resource_requirements_list

enum wxRegKey::WOW64ViewMode

Used to determine how the registry will be viewed, either as 32-bit or 64-bit.

Since

2.9.2

Enumerator

WOW64ViewMode_Default Uses 32-bit registry for 32-bit applications and 64-bit registry for 64-bit ones.

WOW64ViewMode_32 Can be used in 64-bit apps to access the 32-bit registry, has no effect (i.e. treated as default) in 32-bit apps.

WOW64ViewMode_64 Can be used in 32-bit apps to access the 64-bit registry, has no effect (i.e. treated as default) in 64-bit apps.

21.590.3 Constructor & Destructor Documentation

wxRegKey::wxRegKey (WOW64ViewMode *viewMode* = WOW64ViewMode_Default)

Default constructor, initializes to `HKEY_CLASSES_ROOT`.

The *viewMode* parameter is new since wxWidgets 2.9.2.

wxRegKey::wxRegKey (const wxString & *strKey,* WOW64ViewMode *viewMode* = WOW64ViewMode_Default)

The constructor to set the full name of the key.

The *viewMode* parameter is new since wxWidgets 2.9.2.

wxRegKey::wxRegKey (StdKey *keyParent,* const wxString & *strKey,* WOW64ViewMode *viewMode* = WOW64ViewMode_Default)

The constructor to set the full name of the key using one of the standard keys, that is, HKCR, HKCU, HKLM, HKUSR, HKPD, HKCC or HKDD.

The *viewMode* parameter is new since wxWidgets 2.9.2.

wxRegKey::wxRegKey (const wxRegKey & *keyParent,* const wxString & *strKey*)

The constructor to set the full name of the key under a previously created parent.

The registry view is inherited from the parent.

21.590.4 Member Function Documentation

void wxRegKey::Close ( )

Closes the key.

bool wxRegKey::Copy (const wxString & *szNewName*)

Copy the entire contents of the key recursively to another location using the name.

Returns true if successful.

bool wxRegKey::Copy (wxRegKey & *keyDst*)

Copy the entire contents of the key recursively to another location using the key.

Returns true if successful.

bool wxRegKey::CopyValue (const wxString & *szValue,* wxRegKey & *keyDst,* const wxString & *szNewName* = wxEmptyString)

Copy the value to another key, possibly changing its name.

By default it will remain the same. Returns true if successful.

bool wxRegKey::Create (bool *bOkIfExists* = `true`)

Creates the key.

Will fail if the key already exists and *bOkIfExists* is false. Returns true if successful.

void wxRegKey::DeleteKey (const wxString & *szKey*)

Deletes the subkey with all its subkeys and values recursively.

void wxRegKey::DeleteSelf ( )

Deletes this key and all its subkeys and values recursively.

void wxRegKey::DeleteValue (const wxString & *szKey*)

Deletes the named value or use an empty string argument to remove the default value of the key.

bool wxRegKey::Exists ( ) const

Returns true if the key exists.

bool wxRegKey::Export (const wxString & *filename*) const

Write the contents of this key and all its subkeys to the given file.

(The file will not be overwritten; it's an error if it already exists.) Note that we export the key in REGEDIT4 format, not RegSaveKey() binary format nor the newer REGEDIT5. Returns true if successful.

bool wxRegKey::Export (wxOutputStream & *ostr*) const

Write the contents of this key and all its subkeys to the opened stream.

Returns true if successful.

bool wxRegKey::GetFirstKey (wxString & *strKeyName,* long & *lIndex*)

Gets the first key.

Returns true if successful.

bool wxRegKey::GetFirstValue (wxString & *strValueName,* long & *lIndex*)

Gets the first value of this key.

Returns true if successful.

bool wxRegKey::GetKeyInfo (size_t * *pnSubKeys,* size_t * *pnMaxKeyLen,* size_t * *pnValues,* size_t * *pnMaxValueLen*) const

Gets information about the key.

Returns true if successful.

Parameters

pnSubKeys	The number of subkeys.
pnMaxKeyLen	The maximum length of the subkey name.
pnValues	The number of values.
pnMaxValueLen	The maximum length of a value.

wxString wxRegKey::GetName (bool *bShortPrefix* = `true`) const

Gets the name of the registry key.

bool wxRegKey::GetNextKey (wxString & *strKeyName,* long & *lIndex*) const

Gets the next key.

Returns true if successful.

bool wxRegKey::GetNextValue (wxString & *strValueName,* long & *lIndex*) const

Gets the next key value for this key.

Returns true if successful.

ValueType wxRegKey::GetValueType (const wxString & *szValue*) const

Gets the value type.

WOW64ViewMode wxRegKey::GetView ( ) const `[inline]`

Retrieves the registry view used by this key.

Since

2.9.2

Returns

The registry view given at the object's construction.

bool wxRegKey::HasSubKey (const wxString & *szKey*) const

Returns true if given subkey exists.

bool wxRegKey::HasSubkeys ( ) const

Returns true if any subkeys exist.

bool wxRegKey::HasValue (const wxString & *szValue*) const

Returns true if the value exists.

bool wxRegKey::HasValues ( ) const

Returns true if any values exist.

bool wxRegKey::IsEmpty ( ) const

Returns true if this key is empty, nothing under this key.

bool wxRegKey::IsNumericValue (const wxString & *szValue*) const

Returns true if the value contains a number.

bool wxRegKey::IsOpened ( ) const

Returns true if the key is opened.

bool wxRegKey::Open (AccessMode *mode* = Write)

Explicitly opens the key.

This method also allows the key to be opened in read-only mode by passing wxRegKey::Read instead of default wxRegKey::Write parameter. Returns true if successful.

wxRegKey& wxRegKey::operator= (const wxString & *strValue*)

Assignment operator to set the default value of the key.

wxString wxRegKey::QueryDefaultValue ( ) const

Return the default value of the key.

bool wxRegKey::QueryRawValue (const wxString & *szValue,* wxString & *strValue*) const

Retrieves the raw string value.

Returns true if successful. An empty *szValue* queries the default/unnamed key value.

bool wxRegKey::QueryValue (const wxString & *szValue,* wxString & *strValue,* bool *raw*) const

Retrieves the raw or expanded string value.

Returns true if successful. An empty *szValue* queries the default/unnamed key value.

bool wxRegKey::QueryValue (const wxString & *szValue,* long * *plValue*) const

Retrieves the numeric value.

Returns true if successful. An empty *szValue* queries the default/unnamed key value.

bool wxRegKey::QueryValue (const wxString & *szValue,* wxMemoryBuffer & *buf*) const

Retrieves the binary structure.

Returns true if successful. An empty *szValue* queries the default/unnamed key value.

bool wxRegKey::Rename (const wxString & *szNewName*)

Renames the key.

Returns true if successful.

bool wxRegKey::RenameValue (const wxString & *szValueOld,* const wxString & *szValueNew*)

Renames a value.

Returns true if successful.

void wxRegKey::ReserveMemoryForName (size_t *bytes*)

Preallocate some memory for the name.

For wxRegConfig usage only.

void wxRegKey::SetHkey (WXHKEY *hKey*)

Set or change the HKEY handle.

void wxRegKey::SetName (const wxString & *strKey*)

Set the full key name.

The name is absolute. It should start with HKEY_xxx.

void wxRegKey::SetName (StdKey *keyParent,* const wxString & *strKey*)

Set the name relative to the parent key.

void wxRegKey::SetName (const wxRegKey & *keyParent,* const wxString & *strKey*)

Set the name relative to the parent key.

bool wxRegKey::SetValue (const wxString & *szValue,* long *lValue*)

Sets the given *szValue* which must be numeric.

If the value doesn't exist, it is created. Returns true if successful. An empty *szValue* sets the default/unnamed key value.

bool wxRegKey::SetValue (const wxString & *szValue,* const wxString & *strValue*)

Sets the given *szValue* which must be string.

If the value doesn't exist, it is created. Returns true if successful. An empty *szValue* sets the default/unnamed key value.

bool wxRegKey::SetValue (const wxString & *szValue,* const wxMemoryBuffer & *buf*)

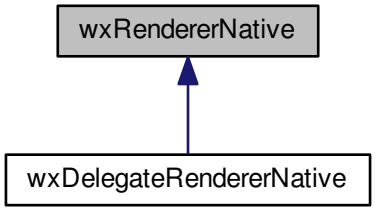
Sets the given *szValue* which must be binary.

If the value doesn't exist, it is created. Returns true if successful. An empty *szValue* sets the default/unnamed key value.

21.591 wxRendererNative Class Reference

`#include <wx/renderer.h>`

Inheritance diagram for wxRendererNative:



21.591.1 Detailed Description

First, a brief introduction to wxRendererNative and why it is needed.

Usually wxWidgets uses the underlying low level GUI system to draw all the controls - this is what we mean when we say that it is a "native" framework. However not all controls exist under all (or even any) platforms and in this case wxWidgets provides a default, generic, implementation of them written in wxWidgets itself.

These controls don't have the native appearance if only the standard line drawing and other graphics primitives are used, because the native appearance is different under different platforms while the lines are always drawn in the same way.

This is why we have renderers: wxRendererNative is a class which virtualizes the drawing, i.e. it abstracts the drawing operations and allows you to draw say, a button, without caring about exactly how this is done. Of course, as we can draw the button differently in different renderers, this also allows us to emulate the native look and feel.

So the renderers work by exposing a large set of high-level drawing functions which are used by the generic controls. There is always a default global renderer but it may be changed or extended by the user, see Render Sample.

All drawing functions take some standard parameters:

- *win* - The window being drawn. It is normally not used and when it is it should only be used as a generic wxWindow (in order to get its low level handle, for example), but you should not assume that it is of some given type as the same renderer function may be reused for drawing different kinds of control.
- *dc* - The wxDC to draw on. Only this device context should be used for drawing. It is not necessary to restore pens and brushes for it on function exit but, on the other hand, you shouldn't assume that it is in any specific state on function entry: the rendering functions should always prepare it.
- *rect* - The bounding rectangle for the element to be drawn.
- *flags* - The optional flags (none by default) which can be a combination of the wxCONTROL_FLAGS.

Note that each drawing function restores the wxDC attributes if it changes them, so it is safe to assume that the same pen, brush and colours that were active before the call to this function are still in effect after it.

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- virtual ~wxRendererNative ()

  Virtual destructor as for any base class.
- virtual void DrawCheckBox (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a check box.
- virtual void DrawComboBoxDropButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a button like the one used by wxComboBox to show a drop down window.
- virtual void DrawDropArrow (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a drop down arrow that is suitable for use outside a combo box.
- virtual void DrawFocusRect (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a focus rectangle using the specified rectangle.
- virtual void DrawGauge (wxWindow *win, wxDC &dc, const wxRect &rect, int value, int max, int flags=0)=0

  Draw a progress bar in the specified rectangle.
- virtual int DrawHeaderButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0, wxHeaderSort↩IconType sortArrow=wxHDR_SORT_ICON_NONE, wxHeaderButtonParams *params=NULL)=0

  Draw the header control button (used, for example, by wxListCtrl).
- virtual int DrawHeaderButtonContents (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0, wx↩HeaderSortIconType sortArrow=wxHDR_SORT_ICON_NONE, wxHeaderButtonParams *params=NULL)=0

  Draw the contents of a header control button (label, sort arrows, etc.).
- virtual void DrawItemSelectionRect (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a selection rectangle underneath the text as used e.g.
- virtual void DrawPushButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a blank push button that looks very similar to wxButton.
- virtual void DrawSplitterBorder (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw the border for sash window: this border must be such that the sash drawn by DrawSplitterSash() blends into it well.
- virtual void DrawSplitterSash (wxWindow *win, wxDC &dc, const wxSize &size, wxCoord position, wx↩Orientation orient, int flags=0)=0

  Draw a sash.
- virtual void DrawTreeItemButton (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw the expanded/collapsed icon for a tree control item.
- virtual void DrawChoice (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a native wxChoice.
- virtual void DrawComboBox (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a native wxComboBox.
- virtual void DrawTextCtrl (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a native wxTextCtrl frame.
- virtual void DrawRadioBitmap (wxWindow *win, wxDC &dc, const wxRect &rect, int flags=0)=0

  Draw a native wxRadioButton bitmap.
- virtual void DrawTitleBarBitmap (wxWindow *win, wxDC &dc, const wxRect &rect, wxTitleBarButton button, int flags=0)=0

  Draw a title bar button in the given state.

- virtual wxSize GetCheckBoxSize (wxWindow *win)=0

  Returns the size of a check box.
- virtual int GetHeaderButtonHeight (wxWindow *win)=0

  Returns the height of a header button, either a fixed platform height if available, or a generic height based on the win window's font.
- virtual int GetHeaderButtonMargin (wxWindow *win)=0

  Returns the horizontal margin on the left and right sides of header button's label.
- virtual wxSplitterRenderParams GetSplitterParams (const wxWindow *win)=0

  Get the splitter parameters, see wxSplitterRenderParams.
- virtual wxRendererVersion GetVersion () const =0

  This function is used for version checking: Load() refuses to load any shared libraries implementing an older or incompatible version.

Static Public Member Functions

- static wxRendererNative & Get ()

  Return the currently used renderer.
- static wxRendererNative & GetDefault ()

  Return the default (native) implementation for this platform – this is also the one used by default but this may be changed by calling Set() in which case the return value of this method may be different from the return value of Get().
- static wxRendererNative & GetGeneric ()

  Return the generic implementation of the renderer.
- static wxRendererNative * Load (const wxString &name)

  Load the renderer from the specified DLL, the returned pointer must be deleted by caller if not NULL when it is not used any more.
- static wxRendererNative * Set (wxRendererNative *renderer)

  Set the renderer to use, passing NULL reverts to using the default renderer (the global renderer must always exist).

21.591.2 Constructor & Destructor Documentation

virtual wxRendererNative::~wxRendererNative ( ) `[virtual]`

Virtual destructor as for any base class.

21.591.3 Member Function Documentation

virtual void wxRendererNative::DrawCheckBox (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = `0`) `[pure virtual]`

Draw a check box.

flags may have the `wxCONTROL_CHECKED`, `wxCONTROL_CURRENT` or `wxCONTROL_UNDETERMINED` bit set, see wxCONTROL_FLAGS.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawChoice (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = `0`) `[pure virtual]`

Draw a native wxChoice.

virtual void wxRendererNative::DrawComboBox (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a native wxComboBox.

virtual void wxRendererNative::DrawComboBoxDropButton (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a button like the one used by wxComboBox to show a drop down window.

The usual appearance is a downwards pointing arrow.

flags may have the wxCONTROL_PRESSED or wxCONTROL_CURRENT bit set, see wxCONTROL_FLAGS.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawDropArrow (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a drop down arrow that is suitable for use outside a combo box.

Arrow will have transparent background.

rect is not entirely filled by the arrow. Instead, you should use bounding rectangle of a drop down button which arrow matches the size you need.

flags may have the wxCONTROL_PRESSED or wxCONTROL_CURRENT bit set, see wxCONTROL_FLAGS.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawFocusRect (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a focus rectangle using the specified rectangle.

wxListCtrl.

The only supported flags is wxCONTROL_SELECTED for items which are selected. see wxCONTROL_FLAGS.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawGauge (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *value*, int *max*, int *flags* = 0) [pure virtual]

Draw a progress bar in the specified rectangle.

The *value* and *max* arguments determine the part of the progress bar that is drawn as being filled in, *max* must be strictly positive and *value* must be between 0 and *max*.

Since

3.1.0

virtual int wxRendererNative::DrawHeaderButton (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0, wxHeaderSortIconType *sortArrow* = wxHDR_SORT_ICON_NONE, wxHeaderButtonParams * *params* = NULL) [pure virtual]

Draw the header control button (used, for example, by wxListCtrl).

Depending on platforms the *flags* parameter may support the wxCONTROL_SELECTED wxCONTROL_DISABLED and wxCONTROL_CURRENT bits, see wxCONTROL_FLAGS.

Returns

The optimal width to contain the unabbreviated label text or bitmap, the sort arrow if present, and internal margins.

Implemented in wxDelegateRendererNative.

virtual int wxRendererNative::DrawHeaderButtonContents (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0, wxHeaderSortIconType *sortArrow* = wxHDR_SORT_ICON_NONE, wxHeaderButtonParams * *params* = NULL) [pure virtual]

Draw the contents of a header control button (label, sort arrows, etc.).

This function is normally only called by DrawHeaderButton().

Depending on platforms the *flags* parameter may support the `wxCONTROL_SELECTED` `wxCONTROL_DISABLED` and `wxCONTROL_CURRENT` bits, see wxCONTROL_FLAGS.

Returns

The optimal width to contain the unabbreviated label text or bitmap, the sort arrow if present, and internal margins.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawItemSelectionRect (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a selection rectangle underneath the text as used e.g.

in a wxListCtrl.

The supported *flags* are `wxCONTROL_SELECTED` for items which are selected (e.g. often a blue rectangle) and `wxCONTROL_CURRENT` for the item that has the focus (often a dotted line around the item's text). `wxCONTROL_FOCUSED` may be used to indicate if the control has the focus (otherwise the selection rectangle is e.g. often grey and not blue). This may be ignored by the renderer or deduced by the code directly from the *win*.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawPushButton (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a blank push button that looks very similar to wxButton.

flags may have the `wxCONTROL_PRESSED`, `wxCONTROL_CURRENT` or `wxCONTROL_ISDEFAULT` bit set, see wxCONTROL_FLAGS.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawRadioBitmap (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw a native wxRadioButton bitmap.

virtual void wxRendererNative::DrawSplitterBorder (wxWindow * *win*, wxDC & *dc*, const wxRect & *rect*, int *flags* = 0) [pure virtual]

Draw the border for sash window: this border must be such that the sash drawn by DrawSplitterSash() blends into it well.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawSplitterSash (wxWindow * *win,* wxDC & *dc,* const wxSize & *size,* wxCoord *position,* wxOrientation *orient,* int *flags* = 0) [pure virtual]

Draw a sash.

The *orient* parameter defines whether the sash should be vertical or horizontal and how the *position* should be interpreted.

Implemented in wxDelegateRendererNative.

virtual void wxRendererNative::DrawTextCtrl (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = 0) [pure virtual]

Draw a native wxTextCtrl frame.

virtual void wxRendererNative::DrawTitleBarBitmap (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* wxTitleBarButton *button,* int *flags* = 0) [pure virtual]

Draw a title bar button in the given state.

This function is currently only available under MSW and OS X (and only for wxTITLEBAR_BUTTON_CLOSE under the latter), its best replacement for the other platforms is to use wxArtProvider to retrieve the bitmaps for `wxART_HELP` and `wxART_CLOSE` (but not any other title bar buttons and not for any state but normal, i.e. not pressed and not current one).

The presence of this function is indicated by `wxHAS_DRAW_TITLE_BAR_BITMAP` symbol being defined.

Also notice that PNG handler must be enabled using wxImage::AddHandler() to use this function under OS X currently as the bitmaps are embedded in the library itself in PNG format.

Since

2.9.1

virtual void wxRendererNative::DrawTreeItemButton (wxWindow * *win,* wxDC & *dc,* const wxRect & *rect,* int *flags* = 0) [pure virtual]

Draw the expanded/collapsed icon for a tree control item.

To draw an expanded button the *flags* parameter must contain `wxCONTROL_EXPANDED` bit, see wxCONTROL_FLAGS.

Implemented in wxDelegateRendererNative.

static wxRendererNative& wxRendererNative::Get ( ) [static]

Return the currently used renderer.

virtual wxSize wxRendererNative::GetCheckBoxSize (wxWindow * *win*) [pure virtual]

Returns the size of a check box.

The *win* parameter is not used currently and can be NULL.

Implemented in wxDelegateRendererNative.

static wxRendererNative& wxRendererNative::GetDefault ( ) [static]

Return the default (native) implementation for this platform – this is also the one used by default but this may be changed by calling Set() in which case the return value of this method may be different from the return value of Get().

static wxRendererNative& wxRendererNative::GetGeneric ( ) [static]

Return the generic implementation of the renderer.

Under some platforms, this is the default renderer implementation, others have platform-specific default renderer which can be retrieved by calling GetDefault().

virtual int wxRendererNative::GetHeaderButtonHeight (wxWindow * *win*) [pure virtual]

Returns the height of a header button, either a fixed platform height if available, or a generic height based on the *win* window's font.

Implemented in wxDelegateRendererNative.

virtual int wxRendererNative::GetHeaderButtonMargin (wxWindow * *win*) [pure virtual]

Returns the horizontal margin on the left and right sides of header button's label.

Since

2.9.2

Implemented in wxDelegateRendererNative.

virtual wxSplitterRenderParams wxRendererNative::GetSplitterParams (const wxWindow * *win*) [pure virtual]

Get the splitter parameters, see wxSplitterRenderParams.

The *win* parameter should be a wxSplitterWindow.

Implemented in wxDelegateRendererNative.

virtual wxRendererVersion wxRendererNative::GetVersion ( ) const [pure virtual]

This function is used for version checking: Load() refuses to load any shared libraries implementing an older or incompatible version.

Remarks

The implementation of this method is always the same in all renderers (simply construct wxRendererVersion using the `wxRendererVersion::Current_XXX` values), but it has to be in the derived, not base, class, to detect mismatches between the renderers versions and so you have to implement it anew in all renderers.

Implemented in wxDelegateRendererNative.

static wxRendererNative* wxRendererNative::Load (const wxString & *name*) [static]

Load the renderer from the specified DLL, the returned pointer must be deleted by caller if not NULL when it is not used any more.

The *name* should be just the base name of the renderer and not the full name of the DLL file which is constructed differently (using wxDynamicLibrary::CanonicalizePluginName()) on different systems.

static wxRendererNative* wxRendererNative::Set (wxRendererNative * *renderer*) `[static]`

Set the renderer to use, passing NULL reverts to using the default renderer (the global renderer must always exist).

Return the previous renderer used with Set() or NULL if none.

21.592 wxRendererVersion Struct Reference

`#include <wx/renderer.h>`

21.592.1 Detailed Description

This simple struct represents the wxRendererNative interface version and is only used as the return value of wxRendererNative::GetVersion().

The version has two components: the version itself and the age. If the main program and the renderer have different versions they are never compatible with each other because the version is only changed when an existing virtual function is modified or removed. The age, on the other hand, is incremented each time a new virtual method is added and so, at least for the compilers using a common C++ object model, the calling program is compatible with any renderer which has the age greater or equal to its age. This verification is done by IsCompatible() method.

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- wxRendererVersion (int version_, int age_)

Static Public Member Functions

- static bool IsCompatible (const wxRendererVersion &ver)

  Checks if the main program is compatible with the renderer having the version ver, returns true if it is and false otherwise.

Public Attributes

- const int age

  The age component.
- const int version

  The version component.

21.592.2 Constructor & Destructor Documentation

wxRendererVersion::wxRendererVersion (int *version_*, int *age_*)

21.592.3 Member Function Documentation

static bool wxRendererVersion::IsCompatible (const wxRendererVersion & *ver*) `[static]`

Checks if the main program is compatible with the renderer having the version *ver*, returns true if it is and false otherwise.

This method is used by wxRendererNative::Load() to determine whether a renderer can be used.

21.592.4 Member Data Documentation

const int wxRendererVersion::age

The age component.

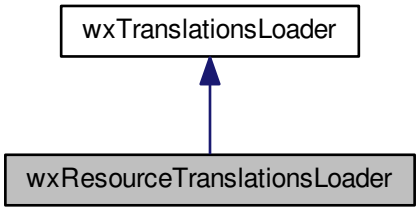
const int wxRendererVersion::version

The version component.

21.593 wxResourceTranslationsLoader Class Reference

`#include <wx/translation.h>`

Inheritance diagram for wxResourceTranslationsLoader:



21.593.1 Detailed Description

This loader makes it possible to load translations from Windows resources.

If you wish to store translation MO files in resources, you have to enable this loader before calling wxTranslations::AddCatalog() or wxLocale::AddCatalog():

```
wxTranslations::Get()->SetLoader(new
      wxResourceTranslationsLoader);
```

Translations are stored in resources as compiled MO files, with type set to "MOFILE" (unless you override GetResourceType()) and name consisting of the domain, followed by underscore, followed by language identification. For example, the relevant part of .rc file would look like this:

```
myapp_de      MOFILE   "catalogs/de/myapp.mo"
myapp_fr      MOFILE   "catalogs/fr/myapp.mo"
myapp_en_GB   MOFILE   "catalogs/en_GB/myapp.mo"
```

This class is only available on Windows.

Since

2.9.1

Protected Member Functions

- virtual wxString GetResourceType () const

  Returns resource type to use for translations.
- virtual WXHINSTANCE GetModule () const

  Returns handle of the module to load resources from.

Additional Inherited Members

21.593.2 Member Function Documentation

virtual WXHINSTANCE wxResourceTranslationsLoader::GetModule ( ) const `[protected], [virtual]`

Returns handle of the module to load resources from.

By default, the main executable is used.

virtual wxString wxResourceTranslationsLoader::GetResourceType ( ) const `[protected], [virtual]`

Returns resource type to use for translations.

Default type is "MOFILE".

21.594 wxRibbonArtProvider Class Reference

```
#include <wx/ribbon/art.h>
```

21.594.1 Detailed Description

wxRibbonArtProvider is responsible for drawing all the components of the ribbon interface.

This allows a ribbon bar to have a pluggable look-and-feel, while retaining the same underlying behaviour. As a single art provider is used for all ribbon components, a ribbon bar usually has a consistent (though unique) appearance.

By default, a wxRibbonBar uses an instance of this class called `wxRibbonDefaultArtProvider`, which resolves to `wxRibbonAUIArtProvider`, `wxRibbonMSWArtProvider`, or `wxRibbonOSXArtProvider` - whichever is most appropriate to the current platform. These art providers are all slightly configurable with regard to colours and fonts, but for larger modifications, you can derive from one of these classes, or write a completely new art provider class. Call wxRibbonBar::SetArtProvider to change the art provider being used.

Library: wxRibbon

Category: Ribbon User Interface

See also

wxRibbonBar

Public Member Functions

- wxRibbonArtProvider ()

  Constructor.
- virtual ~wxRibbonArtProvider ()

  Destructor.
- virtual wxRibbonArtProvider * Clone () const =0

  Create a new art provider which is a clone of this one.
- virtual void SetFlags (long flags)=0

  Set the style flags.
- virtual long GetFlags () const =0

  Get the previously set style flags.
- virtual int GetMetric (int id) const =0

  Get the value of a certain integer setting.
- virtual void SetMetric (int id, int new_val)=0

  Set the value of a certain integer setting to the value new_val.
- virtual void SetFont (int id, const wxFont &font)=0

  Set the value of a certain font setting to the value font.
- virtual wxFont GetFont (int id) const =0

  Get the value of a certain font setting.
- virtual wxColour GetColour (int id) const =0

  Get the value of a certain colour setting.
- virtual void SetColour (int id, const wxColor &colour)=0

  Set the value of a certain colour setting to the value colour.
- wxColour GetColor (int id) const
- void SetColor (int id, const wxColour &color)
- virtual void GetColourScheme (wxColour *primary, wxColour *secondary, wxColour *tertiary) const =0

  Get the current colour scheme.
- virtual void SetColourScheme (const wxColour &primary, const wxColour &secondary, const wxColour &tertiary)=0

  Set all applicable colour settings from a few base colours.
- virtual void DrawTabCtrlBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0

  Draw the background of the tab region of a ribbon bar.
- virtual void DrawTab (wxDC &dc, wxWindow *wnd, const wxRibbonPageTabInfo &tab)=0

  Draw a single tab in the tab region of a ribbon bar.
- virtual void DrawTabSeparator (wxDC &dc, wxWindow *wnd, const wxRect &rect, double visibility)=0

  Draw a separator between two tabs in a ribbon bar.
- virtual void DrawPageBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0

  Draw the background of a ribbon page.
- virtual void DrawScrollButton (wxDC &dc, wxWindow *wnd, const wxRect &rect, long style)=0

  Draw a ribbon-style scroll button.
- virtual void DrawPanelBackground (wxDC &dc, wxRibbonPanel *wnd, const wxRect &rect)=0

  Draw the background and chrome for a ribbon panel.
- virtual void DrawGalleryBackground (wxDC &dc, wxRibbonGallery *wnd, const wxRect &rect)=0

  Draw the background and chrome for a wxRibbonGallery control.
- virtual void DrawGalleryItemBackground (wxDC &dc, wxRibbonGallery *wnd, const wxRect &rect, wxRibbonGalleryItem *item)=0

  Draw the background of a single item in a wxRibbonGallery control.

- virtual void DrawMinimisedPanel (wxDC &dc, wxRibbonPanel *wnd, const wxRect &rect, wxBitmap &bitmap)=0

  Draw a minimised ribbon panel.
- virtual void DrawButtonBarBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0

  Draw the background for a wxRibbonButtonBar control.
- virtual void DrawButtonBarButton (wxDC &dc, wxWindow *wnd, const wxRect &rect, wxRibbonButtonBarButtonKind kind, long state, const wxString &label, const wxBitmap &bitmap_large, const wxBitmap &bitmap_small)=0

  Draw a single button for a wxRibbonButtonBar control.
- virtual void DrawToolBarBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0

  Draw the background for a wxRibbonToolBar control.
- virtual void DrawToolGroupBackground (wxDC &dc, wxWindow *wnd, const wxRect &rect)=0

  Draw the background for a group of tools on a wxRibbonToolBar control.
- virtual void DrawTool (wxDC &dc, wxWindow *wnd, const wxRect &rect, const wxBitmap &bitmap, wxRibbonButtonKind kind, long state)=0

  Draw a single tool (for a wxRibbonToolBar control).
- virtual void DrawToggleButton (wxDC &dc, wxRibbonBar *wnd, const wxRect &rect, wxRibbonDisplayMode mode)=0

  Draw toggle button on wxRibbonBar.
- virtual void DrawHelpButton (wxDC &dc, wxRibbonBar *wnd, const wxRect &rect)=0

  Draw help button on wxRibbonBar.
- virtual void GetBarTabWidth (wxDC &dc, wxWindow *wnd, const wxString &label, const wxBitmap &bitmap, int *ideal, int *small_begin_need_separator, int *small_must_have_separator, int *minimum)=0

  Calculate the ideal and minimum width (in pixels) of a tab in a ribbon bar.
- virtual int GetTabCtrlHeight (wxDC &dc, wxWindow *wnd, const wxRibbonPageTabInfoArray &pages)=0

  Calculate the height (in pixels) of the tab region of a ribbon bar.
- virtual wxSize GetScrollButtonMinimumSize (wxDC &dc, wxWindow *wnd, long style)=0

  Calculate the minimum size (in pixels) of a scroll button.
- virtual wxSize GetPanelSize (wxDC &dc, const wxRibbonPanel *wnd, wxSize client_size, wxPoint *client_offset)=0

  Calculate the size of a panel for a given client size.
- virtual wxSize GetPanelClientSize (wxDC &dc, const wxRibbonPanel *wnd, wxSize size, wxPoint *client_offset)=0

  Calculate the client size of a panel for a given overall size.
- virtual wxRect GetPanelExtButtonArea (wxDC &dc, const wxRibbonPanel *wnd, wxRect rect)=0

  Calculate the position and size of the panel extension button.
- virtual wxSize GetGallerySize (wxDC &dc, const wxRibbonGallery *wnd, wxSize client_size)=0

  Calculate the size of a wxRibbonGallery control for a given client size.
- virtual wxSize GetGalleryClientSize (wxDC &dc, const wxRibbonGallery *wnd, wxSize size, wxPoint *client_offset, wxRect *scroll_up_button, wxRect *scroll_down_button, wxRect *extension_button)=0

  Calculate the client size of a wxRibbonGallery control for a given size.
- virtual wxRect GetPageBackgroundRedrawArea (wxDC &dc, const wxRibbonPage *wnd, wxSize page_old_size, wxSize page_new_size)=0

  Calculate the portion of a page background which needs to be redrawn when a page is resized.
- virtual bool GetButtonBarButtonSize (wxDC &dc, wxWindow *wnd, wxRibbonButtonBarButtonKind kind, wxRibbonButtonBarButtonState size, const wxString &label, wxSize bitmap_size_large, wxSize bitmap_size_small, wxSize *button_size, wxRect *normal_region, wxRect *dropdown_region)=0

  Calculate the size of a button within a wxRibbonButtonBar.
- virtual wxSize GetMinimisedPanelMinimumSize (wxDC &dc, const wxRibbonPanel *wnd, wxSize *desired_bitmap_size, wxDirection *expanded_panel_direction)=0

  Calculate the size of a minimised ribbon panel.
- virtual wxSize GetToolSize (wxDC &dc, wxWindow *wnd, wxSize bitmap_size, wxRibbonButtonKind kind, bool is_first, bool is_last, wxRect *dropdown_region)=0

Calculate the size of a tool within a wxRibbonToolBar.

- virtual wxRect GetBarToggleButtonArea (const wxRect &rect)=0

  Calculate the position and size of the ribbon's toggle button.

- virtual wxRect GetRibbonHelpButtonArea (const wxRect &rect)=0

  Calculate the position and size of the ribbon's help button.

21.594.2 Constructor & Destructor Documentation

wxRibbonArtProvider::wxRibbonArtProvider ( )

Constructor.

virtual wxRibbonArtProvider::~wxRibbonArtProvider ( ) [virtual]

Destructor.

21.594.3 Member Function Documentation

virtual wxRibbonArtProvider* wxRibbonArtProvider::Clone ( ) const [pure virtual]

Create a new art provider which is a clone of this one.

virtual void wxRibbonArtProvider::DrawButtonBarBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) [pure virtual]

Draw the background for a wxRibbonButtonBar control.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto (which will typically be the button bar itself, though this is not guaranteed).
rect	The rectangle within which to draw.

virtual void wxRibbonArtProvider::DrawButtonBarButton (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*, wxRibbonButtonBarButtonKind *kind*, long *state*, const wxString & *label*, const wxBitmap & *bitmap_large*, const wxBitmap & *bitmap_small*) [pure virtual]

Draw a single button for a wxRibbonButtonBar control.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto.
rect	The rectangle within which to draw. The size of this rectangle will be a size previously returned by GetButtonBarButtonSize(), and the rectangle will be entirely within a rectangle on the same device context previously painted with DrawButtonBarBackground().
kind	The kind of button to draw (normal, dropdown or hybrid).
state	Combination of a size flag and state flags from the wxRibbonButtonBarButtonState enumeration.

label	The label of the button.
bitmap_large	The large bitmap of the button (or the large disabled bitmap when wxRIBBON_BUTTONBAR_BUTTON_DISABLED is set in *state*).
bitmap_small	The small bitmap of the button (or the small disabled bitmap when wxRIBBON_BUTTONBAR_BUTTON_DISABLED is set in *state*).

virtual void wxRibbonArtProvider::DrawGalleryBackground (wxDC & *dc*, wxRibbonGallery * *wnd*, const wxRect & *rect*) [pure virtual]

Draw the background and chrome for a wxRibbonGallery control.

This should draw the border, background, scroll buttons, extension button, and any other UI elements which are not attached to a specific gallery item.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto, which is always the gallery whose background and chrome is being drawn. Attributes used during drawing like the gallery hover state and individual button states can be queried from this parameter by wxRibbonGallery::IsHovered(), wxRibbonGallery::GetExtensionButtonState(), wxRibbonGallery::GetUpButtonState(), and wxRibbonGallery::GetDownButtonState().
rect	The rectangle within which to draw. This rectangle is the entire area of the gallery control, not just the client rectangle.

virtual void wxRibbonArtProvider::DrawGalleryItemBackground (wxDC & *dc*, wxRibbonGallery * *wnd*, const wxRect & *rect*, wxRibbonGalleryItem * *item*) [pure virtual]

Draw the background of a single item in a wxRibbonGallery control.

This is painted on top of a gallery background, and behind the items bitmap. Unlike DrawButtonBarButton() and DrawTool(), it is not expected to draw the item bitmap - that is done by the gallery control itself.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto, which is always the gallery which contains the item being drawn.
rect	The rectangle within which to draw. The size of this rectangle will be the size of the item's bitmap, expanded by gallery item padding values (wxRIBBON_ART_GALLERY_BITMAP_PADDING_LEFT_SIZE, wxRIBBON_ART_GALLERY_BITMAP_PADDING_RIGHT_SIZE, wxRIBBON_ART_GALLERY_BITMAP_PADDING_TOP_SIZE, and wxRIBBON_ART_GALLERY_BITMAP_PADDING_BOTTOM_SIZE). The drawing rectangle will be entirely within a rectangle on the same device context previously painted with DrawGalleryBackground().
item	The item whose background is being painted. Typically the background will vary if the item is hovered, active, or selected; wxRibbonGallery::GetSelection(), wxRibbonGallery::GetActiveItem(), and wxRibbonGallery::GetHoveredItem() can be called to test if the given item is in one of these states.

virtual void wxRibbonArtProvider::DrawHelpButton (wxDC & *dc*, wxRibbonBar * *wnd*, const wxRect & *rect*) [pure virtual]

Draw help button on wxRibbonBar.

This should draw a help button at top right corner of ribbon bar.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto, which is always the panel whose background and chrome is being drawn. The panel label and other panel attributes can be obtained by querying this.
rect	The rectangle within which to draw.

Since

2.9.5

virtual void wxRibbonArtProvider::DrawMinimisedPanel (wxDC & *dc*, wxRibbonPanel * *wnd*, const wxRect & *rect*, wxBitmap & *bitmap*) [pure virtual]

Draw a minimised ribbon panel.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto, which is always the panel which is minimised. The panel label can be obtained from this window. The minimised icon obtained from querying the window may not be the size requested by GetMinimisedPanelMinimumSize() - the *bitmap* argument contains the icon in the requested size.
rect	The rectangle within which to draw. The size of the rectangle will be at least the size returned by GetMinimisedPanelMinimumSize().
bitmap	A copy of the panel's minimised bitmap rescaled to the size returned by GetMinimisedPanelMinimumSize().

virtual void wxRibbonArtProvider::DrawPageBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) [pure virtual]

Draw the background of a ribbon page.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto (which is commonly the wxRibbonPage whose background is being drawn, but doesn't have to be).
rect	The rectangle within which to draw.

See also

GetPageBackgroundRedrawArea

virtual void wxRibbonArtProvider::DrawPanelBackground (wxDC & *dc*, wxRibbonPanel * *wnd*, const wxRect & *rect*) [pure virtual]

Draw the background and chrome for a ribbon panel.

This should draw the border, background, label, and any other items of a panel which are outside the client area of a panel.

Note that when a panel is minimised, this function is not called - only DrawMinimisedPanel() is called, so a background should be explicitly painted by that if required.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto, which is always the panel whose background and chrome is being drawn. The panel label and other panel attributes can be obtained by querying this.
rect	The rectangle within which to draw.

virtual void wxRibbonArtProvider::DrawScrollButton (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*, long *style*) [pure virtual]

Draw a ribbon-style scroll button.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto.
rect	The rectangle within which to draw. The size of this rectangle will be at least the size returned by GetScrollButtonMinimumSize() for a scroll button with the same style. For tab scroll buttons, this rectangle will be entirely within a rectangle on the same device context previously painted with DrawTabCtrlBackground(), but this is not guaranteed for other types of button (for example, page scroll buttons will not be painted on an area previously painted with DrawPageBackground()).
style	A combination of flags from wxRibbonScrollButtonStyle, including a direction, a for flag, and one or more states.

virtual void wxRibbonArtProvider::DrawTab (wxDC & *dc*, wxWindow * *wnd*, const wxRibbonPageTabInfo & *tab*) [pure virtual]

Draw a single tab in the tab region of a ribbon bar.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto (not the wxRibbonPage associated with the tab being drawn).
tab	The rectangle within which to draw, and also the tab label, icon, and state (active and/or hovered). The drawing rectangle will be entirely within a rectangle on the same device context previously painted with DrawTabCtrlBackground(). The rectangle's width will be at least the minimum value returned by GetBarTabWidth(), and height will be the value returned by GetTabCtrlHeight().

virtual void wxRibbonArtProvider::DrawTabCtrlBackground (wxDC & *dc*, wxWindow * *wnd*, const wxRect & *rect*) [pure virtual]

Draw the background of the tab region of a ribbon bar.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto.
rect	The rectangle within which to draw.

virtual void wxRibbonArtProvider::DrawTabSeparator (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect,* double *visibility*) `[pure virtual]`

Draw a separator between two tabs in a ribbon bar.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto.
rect	The rectangle within which to draw, which will be entirely within a rectangle on the same device context previously painted with DrawTabCtrlBackground().
visibility	The opacity with which to draw the separator. Values are in the range [0, 1], with 0 being totally transparent, and 1 being totally opaque.

virtual void wxRibbonArtProvider::DrawToggleButton (wxDC & *dc,* wxRibbonBar * *wnd,* const wxRect & *rect,* wxRibbonDisplayMode *mode*) [pure virtual]

Draw toggle button on wxRibbonBar.

This should draw a small toggle button at top right corner of ribbon bar.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto, which is always the panel whose background and chrome is being drawn. The panel label and other panel attributes can be obtained by querying this.
rect	The rectangle within which to draw.
mode	The wxRibbonDisplayMode which should be applied to display button

Since

2.9.5

virtual void wxRibbonArtProvider::DrawTool (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect,* const wxBitmap & *bitmap,* wxRibbonButtonKind *kind,* long *state*) [pure virtual]

Draw a single tool (for a wxRibbonToolBar control).

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto. In most cases this will be a wxRibbonToolBar, but it doesn't have to be.
rect	The rectangle within which to draw. The size of this rectangle will at least the size returned by GetToolSize(), and the height of it will be equal for all tools within the same group. The rectangle will be entirely within a rectangle on the same device context previously painted with DrawToolGroupBackground().
bitmap	The bitmap to use as the tool's foreground. If the tool is a hybrid or dropdown tool, then the foreground should also contain a standard dropdown button.
kind	The kind of tool to draw (normal, dropdown, or hybrid).
state	A combination of wxRibbonToolBarToolState flags giving the state of the tool and it's relative position within a tool group.

virtual void wxRibbonArtProvider::DrawToolBarBackground (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) [pure virtual]

Draw the background for a wxRibbonToolBar control.

Parameters

dc	The device context to draw onto.
wnd	The which is being drawn onto. In most cases this will be a wxRibbonToolBar, but it doesn't have to be.
rect	The rectangle within which to draw. Some of this rectangle will later be drawn over using DrawToolGroupBackground() and DrawTool(), but not all of it will (unless there is only a single group of tools).

virtual void wxRibbonArtProvider::DrawToolGroupBackground (wxDC & *dc,* wxWindow * *wnd,* const wxRect & *rect*) [pure virtual]

Draw the background for a group of tools on a wxRibbonToolBar control.

Parameters

dc	The device context to draw onto.
wnd	The window which is being drawn onto. In most cases this will be a wxRibbonToolBar, but it doesn't have to be.
rect	The rectangle within which to draw. This rectangle is a union of the individual tools' rectangles. As there are no gaps between tools, this rectangle will be painted over exactly once by calls to DrawTool(). The group background could therefore be painted by DrawTool(), though it can be conceptually easier and more efficient to draw it all at once here. The rectangle will be entirely within a rectangle on the same device context previously painted with DrawToolBarBackground().

virtual void wxRibbonArtProvider::GetBarTabWidth (wxDC & *dc,* wxWindow * *wnd,* const wxString & *label,* const wxBitmap & *bitmap,* int * *ideal,* int * *small_begin_need_separator,* int * *small_must_have_separator,* int * *minimum*) [pure virtual]

Calculate the ideal and minimum width (in pixels) of a tab in a ribbon bar.

Parameters

	dc	A device context to use when one is required for size calculations.
	wnd	The window onto which the tab will eventually be drawn.
	label	The tab's label (or wxEmptyString if it has none).
	bitmap	The tab's icon (or wxNullBitmap if it has none).
out	*ideal*	The ideal width (in pixels) of the tab.
out	*small_begin_need_separator*	A size less than the *ideal* size, at which a tab separator should begin to be drawn (i.e. drawn, but still fairly transparent).
out	*small_must_have_separator*	A size less than the *small_begin_need_separator* size, at which a tab separator must be drawn (i.e. drawn at full opacity).
out	*minimum*	A size less than the *small_must_have_separator* size, and greater than or equal to zero, which is the minimum pixel width for the tab.

virtual wxRect wxRibbonArtProvider::GetBarToggleButtonArea (const wxRect & *rect*) [pure virtual]

Calculate the position and size of the ribbon's toggle button.

Parameters

rect	The ribbon bar rectangle from which calculate toggle button rectangle.

Since

2.9.5

virtual bool wxRibbonArtProvider::GetButtonBarButtonSize (wxDC & *dc*, wxWindow * *wnd*, wxRibbonButtonBarButtonKind *kind*, wxRibbonButtonBarButtonState *size*, const wxString & *label*, wxSize *bitmap_size_large*, wxSize *bitmap_size_small*, wxSize * *button_size*, wxRect * *normal_region*, wxRect * *dropdown_region*) [pure virtual]

Calculate the size of a button within a wxRibbonButtonBar.

Parameters

	dc	A device context to use when one is required for size calculations.
	wnd	The window onto which the button will eventually be drawn (which is normally a wxRibbonButtonBar, though this is not guaranteed).
	kind	The kind of button.
	size	The size-class to calculate the size for. Buttons on a button bar can have three distinct sizes: wxRIBBON_BUTTONBAR_BUTTON_SMALL, wxRIBBON_BUTTONBAR_BUTTON_MEDIUM, and wxRIBBON_BUTTONBAR_BUTTON_LARGE. If the requested size-class is not applicable, then false should be returned.
	label	The label of the button.
	bitmap_size_large	The size of all "large" bitmaps on the button bar.
	bitmap_size_small	The size of all "small" bitmaps on the button bar.
out	*button_size*	The size, in pixels, of the button.
out	*normal_region*	The region of the button which constitutes the normal button.
out	*dropdown_region*	The region of the button which constitutes the dropdown button.

Returns

true if a size exists for the button, false otherwise.

wxColour wxRibbonArtProvider::GetColor (int *id*) const

See also

wxRibbonArtProvider::GetColour()

virtual wxColour wxRibbonArtProvider::GetColour (int *id*) const [pure virtual]

Get the value of a certain colour setting.

id can be one of the colour values of wxRibbonArtSetting.

virtual void wxRibbonArtProvider::GetColourScheme (wxColour * *primary*, wxColour * *secondary*, wxColour * *tertiary*) const [pure virtual]

Get the current colour scheme.

Returns three colours such that if SetColourScheme() were called with them, the colour scheme would be restored to what it was when SetColourScheme() was last called. In practice, this usually means that the returned values are the three colours given in the last call to SetColourScheme(), however if SetColourScheme() performs an idempotent operation upon the colours it is given (like clamping a component of the colour), then the returned values may not be

the three colours given in the last call to SetColourScheme(). If SetColourScheme() has not been called, then the returned values should result in a colour scheme similar to, if not identical to, the default colours of the art provider. Note that if SetColour() is called, then GetColourScheme() does not try and return a colour scheme similar to colours being used - it's return values are dependent upon the last values given to SetColourScheme(), as described above.

Parameters

out	*primary*	Pointer to a location to store the primary colour, or NULL.
out	*secondary*	Pointer to a location to store the secondary colour, or NULL.
out	*tertiary*	Pointer to a location to store the tertiary colour, or NULL.

virtual long wxRibbonArtProvider::GetFlags ( ) const `[pure virtual]`

Get the previously set style flags.

virtual wxFont wxRibbonArtProvider::GetFont (int *id*) const `[pure virtual]`

Get the value of a certain font setting.

id can be one of the font values of wxRibbonArtSetting.

virtual wxSize wxRibbonArtProvider::GetGalleryClientSize (wxDC & *dc*, const wxRibbonGallery * *wnd*, wxSize *size*, wxPoint * *client_offset*, wxRect * *scroll_up_button*, wxRect * *scroll_down_button*, wxRect * *extension_button*) `[pure virtual]`

Calculate the client size of a wxRibbonGallery control for a given size.

This should act as the inverse to GetGallerySize(), and decrement the given size by enough to fir the gallery border, buttons, and other chrome.

Parameters

	dc	A device context to use if one is required for size calculations.
	wnd	The gallery in question.
	size	The overall size to calculate the client size for.
out	*client_offset*	The position within the given size at which the returned client size begins.
out	*scroll_up_button*	The rectangle within the given size which the scroll up button occupies.
out	*scroll_down_button*	The rectangle within the given size which the scroll down button occupies.
out	*extension_button*	The rectangle within the given size which the extension button occupies.

virtual wxSize wxRibbonArtProvider::GetGallerySize (wxDC & *dc*, const wxRibbonGallery * *wnd*, wxSize *client_size*) `[pure virtual]`

Calculate the size of a wxRibbonGallery control for a given client size.

This should increment the given size by enough to fit the gallery border, buttons, and any other chrome.

Parameters

dc	A device context to use if one is required for size calculations.
wnd	The gallery in question.
client_size	The client size.

See also

GetGalleryClientSize()

virtual int wxRibbonArtProvider::GetMetric (int *id*) const [pure virtual]

Get the value of a certain integer setting.

id can be one of the size values of wxRibbonArtSetting.

virtual wxSize wxRibbonArtProvider::GetMinimisedPanelMinimumSize (wxDC & *dc*, const wxRibbonPanel * *wnd*, wxSize * *desired_bitmap_size*, wxDirection * *expanded_panel_direction*) [pure virtual]

Calculate the size of a minimised ribbon panel.

Parameters

	dc	A device context to use when one is required for size calculations.
	wnd	The ribbon panel in question. Attributes like the panel label can be queried from this.
out	*desired_bitmap_size*	Optional parameter which is filled with the size of the bitmap suitable for a minimised ribbon panel.
out	*expanded_panel_direction*	Optional parameter which is filled with the direction of the minimised panel (wxEAST or wxSOUTH depending on the style).

virtual wxRect wxRibbonArtProvider::GetPageBackgroundRedrawArea (wxDC & *dc*, const wxRibbonPage * *wnd*, wxSize *page_old_size*, wxSize *page_new_size*) [pure virtual]

Calculate the portion of a page background which needs to be redrawn when a page is resized.

To optimise the drawing of page backgrounds, as small an area as possible should be returned. Of course, if the way in which a background is drawn means that the entire background needs to be repainted on resize, then the entire new size should be returned.

Parameters

dc	A device context to use when one is required for size calculations.
wnd	The page which is being resized.
page_old_size	The size of the page prior to the resize (which has already been painted).
page_new_size	The size of the page after the resize.

virtual wxSize wxRibbonArtProvider::GetPanelClientSize (wxDC & *dc*, const wxRibbonPanel * *wnd*, wxSize *size*, wxPoint * *client_offset*) [pure virtual]

Calculate the client size of a panel for a given overall size.

This should act as the inverse to GetPanelSize(), and decrement the given size by enough to fit the panel label and other chrome.

Parameters

	dc	A device context to use if one is required for size calculations.
	wnd	The ribbon panel in question.
	size	The overall size to calculate client size for.
out	*client_offset*	The offset where the returned client size begins within the given *size* (may be NULL).

See also

GetPanelSize()

virtual wxRect wxRibbonArtProvider::GetPanelExtButtonArea (wxDC & *dc,* const wxRibbonPanel * *wnd,* wxRect *rect*)
`[pure virtual]`

Calculate the position and size of the panel extension button.

Parameters

dc	A device context to use if one is required for size calculations.
wnd	The ribbon panel in question.
rect	The panel rectangle from which calculate extension button rectangle.

Since

2.9.4

virtual wxSize wxRibbonArtProvider::GetPanelSize (wxDC & *dc*, const wxRibbonPanel * *wnd*, wxSize *client_size*, wxPoint * *client_offset*) [pure virtual]

Calculate the size of a panel for a given client size.

This should increment the given size by enough to fit the panel label and other chrome.

Parameters

	dc	A device context to use if one is required for size calculations.
	wnd	The ribbon panel in question.
	client_size	The client size.
out	*client_offset*	The offset where the client rectangle begins within the panel (may be NULL).

See also

GetPanelClientSize()

virtual wxRect wxRibbonArtProvider::GetRibbonHelpButtonArea (const wxRect & *rect*) [pure virtual]

Calculate the position and size of the ribbon's help button.

Parameters

rect	The ribbon bar rectangle from which calculate help button rectangle.

Since

2.9.5

virtual wxSize wxRibbonArtProvider::GetScrollButtonMinimumSize (wxDC & *dc*, wxWindow * *wnd*, long *style*) [pure virtual]

Calculate the minimum size (in pixels) of a scroll button.

Parameters

dc	A device context to use when one is required for size calculations.
wnd	The window onto which the scroll button will eventually be drawn.
style	A combination of flags from wxRibbonScrollButtonStyle, including a direction, and a for flag (state flags may be given too, but should be ignored, as a button should retain a constant size, regardless of its state).

virtual int wxRibbonArtProvider::GetTabCtrlHeight (wxDC & *dc*, wxWindow * *wnd*, const wxRibbonPageTabInfoArray & *pages*) [pure virtual]

Calculate the height (in pixels) of the tab region of a ribbon bar.

Note that as the tab region can contain scroll buttons, the height should be greater than or equal to the minimum height for a tab scroll button.

Parameters

dc	A device context to use when one is required for size calculations.
wnd	The window onto which the tabs will eventually be drawn.
pages	The tabs which will acquire the returned height.

virtual wxSize wxRibbonArtProvider::GetToolSize (wxDC & *dc*, wxWindow * *wnd*, wxSize *bitmap_size*, wxRibbonButtonKind *kind*, bool *is_first*, bool *is_last*, wxRect * *dropdown_region*) [pure virtual]

Calculate the size of a tool within a wxRibbonToolBar.

Parameters

	dc	A device context to use when one is required for size calculations.
	wnd	The window onto which the tool will eventually be drawn.
	bitmap_size	The size of the tool's foreground bitmap.
	kind	The kind of tool (normal, dropdown, or hybrid).
	is_first	true if the tool is the first within its group. false otherwise.
	is_last	true if the tool is the last within its group. false otherwise.
out	*dropdown_region*	For dropdown and hybrid tools, the region within the returned size which counts as the dropdown part.

void wxRibbonArtProvider::SetColor (int *id*, const wxColour & *color*)

See also

wxRibbonArtProvider::SetColour()

virtual void wxRibbonArtProvider::SetColour (int *id*, const wxColor & *colour*) [pure virtual]

Set the value of a certain colour setting to the value *colour*.

id can be one of the colour values of wxRibbonArtSetting, though not all colour settings will have an effect on every art provider.

See also

SetColourScheme()

virtual void wxRibbonArtProvider::SetColourScheme (const wxColour & *primary*, const wxColour & *secondary*, const wxColour & *tertiary*) [pure virtual]

Set all applicable colour settings from a few base colours.

Uses any or all of the three given colours to create a colour scheme, and then sets all colour settings which are relevant to the art provider using that scheme. Note that some art providers may not use the tertiary colour for anything, and some may not use the secondary colour either.

See also

SetColour()
GetColourScheme()

virtual void wxRibbonArtProvider::SetFlags (long *flags*) `[pure virtual]`

Set the style flags.

Normally called automatically by wxRibbonBar::SetArtProvider with the ribbon bar's style flags, so that the art provider has the same flags as the bar which it is serving.

virtual void wxRibbonArtProvider::SetFont (int *id*, const wxFont & *font*) `[pure virtual]`

Set the value of a certain font setting to the value *font*.

id can be one of the font values of wxRibbonArtSetting.

virtual void wxRibbonArtProvider::SetMetric (int *id*, int *new_val*) `[pure virtual]`

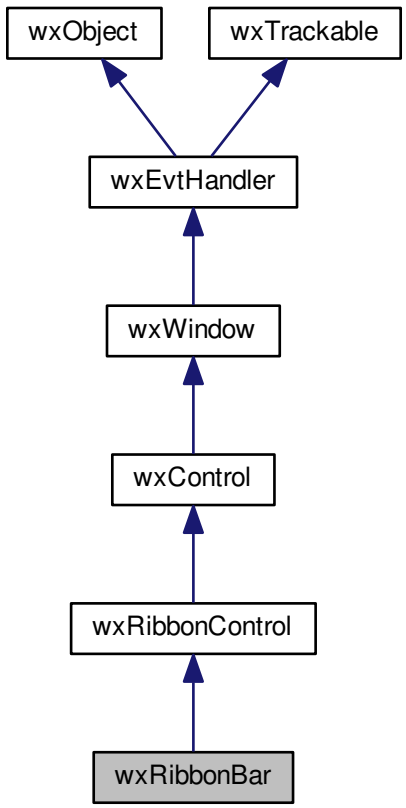
Set the value of a certain integer setting to the value *new_val*.

id can be one of the size values of wxRibbonArtSetting.

21.595 wxRibbonBar Class Reference

`#include <wx/ribbon/bar.h>`

Inheritance diagram for wxRibbonBar:



21.595.1 Detailed Description

Top-level control in a ribbon user interface.

Serves as a tabbed container for wxRibbonPage - a ribbon user interface typically has a ribbon bar, which contains one or more wxRibbonPages, which in turn each contain one or more wxRibbonPanels, which in turn contain controls.

While a wxRibbonBar has tabs similar to a wxNotebook, it does not follow the same API for adding pages. Containers like wxNotebook can contain any type of window as a page, hence the normal procedure is to create the sub-window and then call wxBookCtrlBase::AddPage(). As wxRibbonBar can only have wxRibbonPage as children (and a wxRibbonPage can only have a wxRibbonBar as parent), when a page is created, it is automatically added to the bar - there is no AddPage equivalent to call.

After all pages have been created, and all controls and panels placed on those pages, Realize() must be called.

See also

wxRibbonPage
wxRibbonPanel

Styles

This class supports the following styles:

- wxRIBBON_BAR_DEFAULT_STYLE: Defined as wxRIBBON_BAR_FLOW_HORIZONTAL | wxRIBBON_BAR_SHOW_PAGE_LABELS | wxRIBBON_BAR_SHOW_PANEL_EXT_BUTTONS | wxRIBBON_BAR_SHOW_TOGGLE_BUTTON | wxRIBBON_BAR_SHOW_HELP_BUTTON.
- wxRIBBON_BAR_FOLDBAR_STYLE: Defined as wxRIBBON_BAR_FLOW_VERTICAL | wxRIBBON_BAR_SHOW_PAGE_ICONS | wxRIBBON_BAR_SHOW_PANEL_EXT_BUTTONS | wxRIBBON_BAR_SHOW_PANEL_MINIMISE_BUTTONS
- wxRIBBON_BAR_SHOW_PAGE_LABELS: Causes labels to be shown on the tabs in the ribbon bar.
- wxRIBBON_BAR_SHOW_PAGE_ICONS: Causes icons to be shown on the tabs in the ribbon bar.
- wxRIBBON_BAR_FLOW_HORIZONTAL: Causes panels within pages to stack horizontally.
- wxRIBBON_BAR_FLOW_VERTICAL: Causes panels within pages to stack vertically.
- wxRIBBON_BAR_SHOW_PANEL_EXT_BUTTONS: Causes extension buttons to be shown on panels (where the panel has such a button).
- wxRIBBON_BAR_SHOW_PANEL_MINIMISE_BUTTONS: Causes minimise buttons to be shown on panels (where the panel has such a button).
- wxRIBBON_BAR_SHOW_TOGGLE_BUTTON: Causes a toggle button to appear on the ribbon bar at top-right corner. This style is new since wxWidgets 2.9.5.
- wxRIBBON_BAR_SHOW_HELP_BUTTON: Causes a help button to appear on the ribbon bar at the top-right corner. This style is new since wxWidgets 2.9.5.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxRibbonBarEvent& event)

Event macros for events emitted by this class:

- EVT_RIBBONBAR_PAGE_CHANGED(id, func): Triggered after the transition from one page being active to a different page being active.

- EVT_RIBBONBAR_PAGE_CHANGING(id, func): Triggered prior to the transition from one page being active to a different page being active, and can veto the change.
- EVT_RIBBONBAR_TAB_MIDDLE_DOWN(id, func): Triggered when the middle mouse button is pressed on a tab.
- EVT_RIBBONBAR_TAB_MIDDLE_UP(id, func): Triggered when the middle mouse button is released on a tab.
- EVT_RIBBONBAR_TAB_RIGHT_DOWN(id, func): Triggered when the right mouse button is pressed on a tab.
- EVT_RIBBONBAR_TAB_RIGHT_UP(id, func): Triggered when the right mouse button is released on a tab.
- EVT_RIBBONBAR_TAB_LEFT_DCLICK(id, func): Triggered when the left mouse button is double clicked on a tab.
- EVT_RIBBONBAR_TOGGLED(id, func): Triggered when the button triggering the ribbon bar is clicked. This event is new since wxWidgets 2.9.5.
- EVT_RIBBONBAR_HELP_CLICK(id, func): Triggered when the help button is clicked. This even is new since wxWidgets 2.9.5.

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonBar ()

  Default constructor.
- wxRibbonBar (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxRIBBON_BAR_DEFAULT_STYLE)

  Construct a ribbon bar with the given parameters.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxRIBBON_BAR_DEFAULT_STYLE)

  Create a ribbon bar in two-step ribbon bar construction.
- virtual ~wxRibbonBar ()

  Destructor.
- void SetTabCtrlMargins (int left, int right)

  Set the margin widths (in pixels) on the left and right sides of the tab bar region of the ribbon bar.
- void SetArtProvider (wxRibbonArtProvider *art)

  Set the art provider to be used be the ribbon bar.
- bool SetActivePage (size_t page)

  Set the active page by index, without triggering any events.
- bool SetActivePage (wxRibbonPage *page)

  Set the active page, without triggering any events.
- int GetActivePage () const

  Get the index of the active page.
- wxRibbonPage * GetPage (int n)

  Get a page by index.
- size_t GetPageCount () const

  Get the number of pages in this bar.
- bool DismissExpandedPanel ()

Dismiss the expanded panel of the currently active page.

- int GetPageNumber (wxRibbonPage *page) const

  Returns the number for a given ribbon bar page.
- void DeletePage (size_t n)

  Delete a single page from this ribbon bar.
- void ClearPages ()

  Delete all pages from the ribbon bar.
- bool IsPageShown (size_t page) const

  Indicates whether the tab for the given page is shown to the user or not.
- void ShowPage (size_t page, bool show_tab=true)

  Show or hide the tab for a given page.
- void HidePage (size_t page)

  Hides the tab for a given page.
- bool IsPageHighlighted (size_t page) const

  Indicates whether a tab is currently highlighted.
- void AddPageHighlight (size_t page, bool highlight=true)

  Highlight the specified tab.
- void RemovePageHighlight (size_t page)

  Changes a tab to not be highlighted.
- void ShowPanels (wxRibbonDisplayMode mode)

  Shows or hide the panel area of the ribbon bar according to the given display mode.
- void ShowPanels (bool show=true)

  Shows or hides the panel area of the ribbon bar.
- void HidePanels ()

  Hides the panel area of the ribbon bar.
- bool ArePanelsShown () const

  Indicates whether the panel area of the ribbon bar is shown.
- wxRibbonDisplayMode GetDisplayMode () const

  Returns the current display mode of the panel area.
- virtual bool Realize ()

  Perform initial layout and size calculations of the bar and its children.

Additional Inherited Members

21.595.2 Constructor & Destructor Documentation

wxRibbonBar::wxRibbonBar ( )

Default constructor.

With this constructor, Create() should be called in order to create the ribbon bar.

wxRibbonBar::wxRibbonBar (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `wxRIBBON_BAR_DEFAULT_STYLE`)

Construct a ribbon bar with the given parameters.

virtual wxRibbonBar::~wxRibbonBar ( ) `[virtual]`

Destructor.

21.595.3 Member Function Documentation

void wxRibbonBar::AddPageHighlight (size_t *page*, bool *highlight* = `true`)

Highlight the specified tab.

Highlighted tabs have a colour between that of the active tab and a tab over which the mouse is hovering. This can be used to make a tab (usually temporarily) more noticeable to the user.

Since

2.9.5

bool wxRibbonBar::ArePanelsShown ( ) const

Indicates whether the panel area of the ribbon bar is shown.

See also

ShowPanels()

Since

2.9.2

void wxRibbonBar::ClearPages ( )

Delete all pages from the ribbon bar.

Since

2.9.4

bool wxRibbonBar::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `wxRIBBON_BAR_DEFAULT_STYLE`)

Create a ribbon bar in two-step ribbon bar construction.

Should only be called when the default constructor is used, and arguments have the same meaning as in the full constructor.

void wxRibbonBar::DeletePage (size_t *n*)

Delete a single page from this ribbon bar.

The user must call wxRibbonBar::Realize() after one (or more) calls to this function.

Since

2.9.4

bool wxRibbonBar::DismissExpandedPanel ( )

Dismiss the expanded panel of the currently active page.

Calls and returns the value from wxRibbonPage::DismissExpandedPanel() for the currently active page, or false if there is no active page.

int wxRibbonBar::GetActivePage ( ) const

Get the index of the active page.

In the rare case of no page being active, -1 is returned.

wxRibbonDisplayMode wxRibbonBar::GetDisplayMode ( ) const

Returns the current display mode of the panel area.

See also

ShowPanels()

Since

3.1.0

wxRibbonPage* wxRibbonBar::GetPage (int *n*)

Get a page by index.

NULL will be returned if the given index is out of range.

size_t wxRibbonBar::GetPageCount ( ) const

Get the number of pages in this bar.

Since

2.9.4

int wxRibbonBar::GetPageNumber (wxRibbonPage * *page*) const

Returns the number for a given ribbon bar page.

The number can be used in other ribbon bar calls.

Since

2.9.5

void wxRibbonBar::HidePage (size_t *page*)

Hides the tab for a given page.

Equivalent to `ShowPage(page, false)`.

Since

2.9.5

void wxRibbonBar::HidePanels ( )

Hides the panel area of the ribbon bar.

This method behaves like ShowPanels() with false argument.

Since

2.9.2

bool wxRibbonBar::IsPageHighlighted (size_t *page*) const

Indicates whether a tab is currently highlighted.

See also

AddPageHighlight()

Since

2.9.5

bool wxRibbonBar::IsPageShown (size_t *page*) const

Indicates whether the tab for the given page is shown to the user or not.

By default all page tabs are shown.

Since

2.9.5

virtual bool wxRibbonBar::Realize ( ) [virtual]

Perform initial layout and size calculations of the bar and its children.

This must be called after all of the bar's children have been created (and their children created, etc.) - if it is not, then windows may not be laid out or sized correctly.

Also calls wxRibbonPage::Realize() on each child page.

Reimplemented from wxRibbonControl.

void wxRibbonBar::RemovePageHighlight (size_t *page*)

Changes a tab to not be highlighted.

See also

AddPageHighlight()

Since

2.9.5

bool wxRibbonBar::SetActivePage (size_t *page*)

Set the active page by index, without triggering any events.

Parameters

page	The zero-based index of the page to activate.

Returns

true if the specified page is now active, false if it could not be activated (for example because the page index is invalid).

bool wxRibbonBar::SetActivePage (wxRibbonPage * *page*)

Set the active page, without triggering any events.

Parameters

page	The page to activate.

Returns

true if the specified page is now active, false if it could not be activated (for example because the given page is not a child of the ribbon bar).

void wxRibbonBar::SetArtProvider (wxRibbonArtProvider * *art*) [virtual]

Set the art provider to be used be the ribbon bar.

Also sets the art provider on all current wxRibbonPage children, and any wxRibbonPage children added in the future.

Note that unlike most other ribbon controls, the ribbon bar creates a default art provider when initialised, so an explicit call to SetArtProvider() is not required if the default art provider is sufficient. Also, unlike other ribbon controls, the ribbon bar takes ownership of the given pointer, and will delete it when the art provider is changed or the bar is destroyed. If this behaviour is not desired, then clone the art provider before setting it.

Reimplemented from wxRibbonControl.

void wxRibbonBar::SetTabCtrlMargins (int *left*, int *right*)

Set the margin widths (in pixels) on the left and right sides of the tab bar region of the ribbon bar.

These margins will be painted with the tab background, but tabs and scroll buttons will never be painted in the margins.

The left margin could be used for rendering something equivalent to the "Office Button", though this is not currently implemented. The right margin could be used for rendering a help button, and/or MDI buttons, but again, this is not currently implemented.

void wxRibbonBar::ShowPage (size_t *page*, bool *show_tab* = true)

Show or hide the tab for a given page.

After showing or hiding a tab, you need to call wxRibbonBar::Realize(). If you hide the tab for the currently active page (GetActivePage) then you should call SetActivePage to activate a different page.

Since

2.9.5

void wxRibbonBar::ShowPanels (wxRibbonDisplayMode *mode*)

Shows or hide the panel area of the ribbon bar according to the given display mode.

Since

3.1.0

void wxRibbonBar::ShowPanels (bool *show* = `true`)

Shows or hides the panel area of the ribbon bar.

If the panel area is hidden, then only the tab of the ribbon bar will be shown. This is useful for giving the user more screen space to work with when he/she doesn't need to see the ribbon's options.

If the panel is currently shown, this method pins it, use the other overload of this method to specify the exact panel display mode to avoid it.

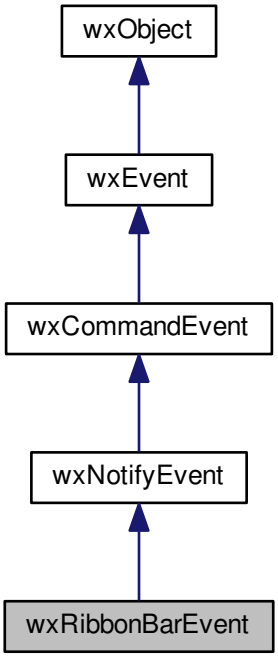
Since

2.9.2

21.596 wxRibbonBarEvent Class Reference

`#include <wx/ribbon/bar.h>`

Inheritance diagram for wxRibbonBarEvent:



21.596.1 Detailed Description

Event used to indicate various actions relating to a wxRibbonBar.

See wxRibbonBar for available event types.

Library: wxRibbon

Category: Events, Ribbon User Interface

See also

wxRibbonBar

Public Member Functions

- wxRibbonBarEvent (wxEventType command_type=wxEVT_NULL, int win_id=0, wxRibbonPage *page=NULL)

  Constructor.
- wxRibbonPage * GetPage ()

  Returns the page being changed to, or being clicked on.
- void SetPage (wxRibbonPage *page)

  Sets the page relating to this event.

Additional Inherited Members

21.596.2 Constructor & Destructor Documentation

wxRibbonBarEvent::wxRibbonBarEvent (wxEventType *command_type* = wxEVT_NULL, int *win_id* = 0, wxRibbonPage * *page* = NULL)

Constructor.

21.596.3 Member Function Documentation

wxRibbonPage* wxRibbonBarEvent::GetPage ( )

Returns the page being changed to, or being clicked on.

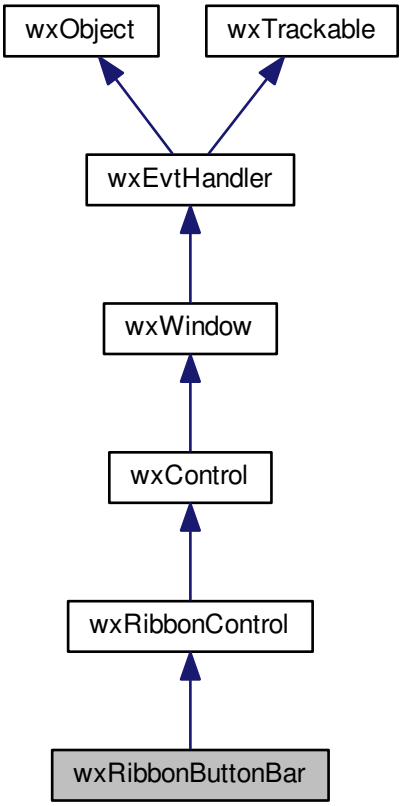
void wxRibbonBarEvent::SetPage (wxRibbonPage * *page*)

Sets the page relating to this event.

21.597 wxRibbonButtonBar Class Reference

```
#include <wx/ribbon/buttonbar.h>
```

Inheritance diagram for wxRibbonButtonBar:



21.597.1 Detailed Description

A ribbon button bar is similar to a traditional toolbar.

It contains one or more buttons (button bar buttons, not wxButtons), each of which has a label and an icon. It differs from a wxRibbonToolBar in several ways:

- Individual buttons can grow and contract.
- Buttons have labels as well as bitmaps.
- Bitmaps are typically larger (at least 32x32 pixels) on a button bar compared to a tool bar (which typically has 16x15).
- There is no grouping of buttons on a button bar
- A button bar typically has a border around each individual button, whereas a tool bar typically has a border around each group of buttons.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxRibbonButtonBarEvent& event)

Event macros for events emitted by this class:

- EVT_RIBBONBUTTONBAR_CLICKED(id, func): Triggered when the normal (non-dropdown) region of a button on the button bar is clicked.
- EVT_RIBBONBUTTONBAR_DROPDOWN_CLICKED(id, func): Triggered when the dropdown region of a button on the button bar is clicked. wxRibbonButtonBarEvent::PopupMenu() should be called by the event handler if it wants to display a popup menu (which is what most dropdown buttons should be doing).

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonButtonBar ()

  Default constructor.
- wxRibbonButtonBar (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Construct a ribbon button bar with the given parameters.
- virtual ~wxRibbonButtonBar ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Create a button bar in two-step button bar construction.
- virtual wxRibbonButtonBarButtonBase * AddButton (int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL)

  Add a button to the button bar (simple version).
- virtual wxRibbonButtonBarButtonBase * AddDropdownButton (int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Add a dropdown button to the button bar (simple version).
- virtual wxRibbonButtonBarButtonBase * AddHybridButton (int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Add a hybrid button to the button bar (simple version).
- virtual wxRibbonButtonBarButtonBase * AddToggleButton (int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Add a toggle button to the button bar (simple version).
- virtual wxRibbonButtonBarButtonBase * AddButton (int button_id, const wxString &label, const wxBitmap &bitmap, const wxBitmap &bitmap_small=wxNullBitmap, const wxBitmap &bitmap_disabled=wxNullBitmap, const wxBitmap &bitmap_small_disabled=wxNullBitmap, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL, const wxString &help_string=wxEmptyString)

  Add a button to the button bar.
- virtual wxRibbonButtonBarButtonBase * InsertButton (size_t pos, int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL)

  Inserts a button to the button bar (simple version) at the given position.

- virtual wxRibbonButtonBarButtonBase * InsertDropdownButton (size_t pos, int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Inserts a dropdown button to the button bar (simple version) at the given position.
- virtual wxRibbonButtonBarButtonBase * InsertHybridButton (size_t pos, int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Inserts a hybrid button to the button bar (simple version) at the given position.
- virtual wxRibbonButtonBarButtonBase * InsertToggleButton (size_t pos, int button_id, const wxString &label, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Inserts a toggle button to the button bar (simple version) at the given position.
- virtual wxRibbonButtonBarButtonBase * InsertButton (size_t pos, int button_id, const wxString &label, const wxBitmap &bitmap, const wxBitmap &bitmap_small=wxNullBitmap, const wxBitmap &bitmap_disabled=wxNullBitmap, const wxBitmap &bitmap_small_disabled=wxNullBitmap, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL, const wxString &help_string=wxEmptyString)

  Insert a button to the button bar at the given position.
- virtual size_t GetButtonCount () const

  Returns the number of buttons in this button bar.
- void SetItemClientObject (wxRibbonButtonBarButtonBase *item, wxClientData *data)

  Set the client object associated with a button.
- wxClientData * GetItemClientObject (const wxRibbonButtonBarButtonBase *item) const

  Get the client object associated with a button.
- void SetItemClientData (wxRibbonButtonBarButtonBase *item, void *data)

  Set the client data associated with a button.
- void * GetItemClientData (const wxRibbonButtonBarButtonBase *item) const

  Get the client data associated with a button.
- virtual wxRibbonButtonBarButtonBase * GetItem (size_t n) const

  Returns the N-th button of the bar.
- virtual wxRibbonButtonBarButtonBase * GetItemById (int id) const

  Returns the first button having a given id or NULL if none matches.
- virtual int GetItemId (wxRibbonButtonBarButtonBase *) const

  Returns the id of a button.
- virtual bool Realize ()

  Calculate button layouts and positions.
- virtual void ClearButtons ()

  Delete all buttons from the button bar.
- virtual bool DeleteButton (int button_id)

  Delete a single button from the button bar.
- virtual void EnableButton (int button_id, bool enable=true)

  Enable or disable a single button on the bar.
- virtual void ToggleButton (int button_id, bool checked)

  Set a toggle button to the checked or unchecked state.
- virtual wxRibbonButtonBarButtonBase * GetActiveItem () const

  Returns the active item of the button bar or NULL if there is none.
- virtual wxRibbonButtonBarButtonBase * GetHoveredItem () const

  Returns the hovered item of the button bar or NULL if there is none.

- void SetShowToolTipsForDisabled (bool show)

  Indicates whether tooltips are shown for disabled buttons.
- bool GetShowToolTipsForDisabled () const

  Sets whether tooltips should be shown for disabled buttons or not.

Additional Inherited Members

21.597.2 Constructor & Destructor Documentation

wxRibbonButtonBar::wxRibbonButtonBar ( )

Default constructor.

With this constructor, Create() should be called in order to create the button bar.

wxRibbonButtonBar::wxRibbonButtonBar (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Construct a ribbon button bar with the given parameters.

Parameters

parent	Parent window for the button bar (typically a wxRibbonPanel).
id	An identifier for the button bar. `wxID_ANY` is taken to mean a default.
pos	Initial position of the button bar.
size	Initial size of the button bar.
style	Button bar style, currently unused.

virtual wxRibbonButtonBar::~wxRibbonButtonBar ( ) `[virtual]`

Destructor.

21.597.3 Member Function Documentation

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::AddButton (int *button_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxString & *help_string*, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL) `[virtual]`

Add a button to the button bar (simple version).

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::AddButton (int *button_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxBitmap & *bitmap_small* = wxNullBitmap, const wxBitmap & *bitmap_disabled* = wxNullBitmap, const wxBitmap & *bitmap_small_disabled* = wxNullBitmap, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL, const wxString & *help_string* = wxEmptyString) `[virtual]`

Add a button to the button bar.

Parameters

button_id	ID of the new button (used for event callbacks).

label	Label of the new button.
bitmap	Large bitmap of the new button. Must be the same size as all other large bitmaps used on the button bar.
bitmap_small	Small bitmap of the new button. If left as null, then a small bitmap will be automatically generated. Must be the same size as all other small bitmaps used on the button bar.
bitmap_disabled	Large bitmap of the new button when it is disabled. If left as null, then a bitmap will be automatically generated from *bitmap*.
bitmap_small_disabled	Small bitmap of the new button when it is disabled. If left as null, then a bitmap will be automatically generated from *bitmap_small*.
kind	The kind of button to add.
help_string	The UI help string to associate with the new button.

Returns

An opaque pointer which can be used only with other button bar methods.

See also

AddDropdownButton()
AddHybridButton()
AddToggleButton()

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::AddDropdownButton (int *button_id,* const wxString & *label,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Add a dropdown button to the button bar (simple version).

See also

AddButton()

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::AddHybridButton (int *button_id,* const wxString & *label,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Add a hybrid button to the button bar (simple version).

See also

AddButton()

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::AddToggleButton (int *button_id,* const wxString & *label,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Add a toggle button to the button bar (simple version).

See also

AddButton()

virtual void wxRibbonButtonBar::ClearButtons ( ) [virtual]

Delete all buttons from the button bar.

See also

DeleteButton()

bool wxRibbonButtonBar::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Create a button bar in two-step button bar construction.

Should only be called when the default constructor is used, and arguments have the same meaning as in the full constructor.

virtual bool wxRibbonButtonBar::DeleteButton (int *button_id*) [virtual]

Delete a single button from the button bar.

The corresponding button is deleted by this function, so any pointers to it previously obtained by GetItem() or GetItemById() become invalid.

See also

ClearButtons()

virtual void wxRibbonButtonBar::EnableButton (int *button_id*, bool *enable* = true) [virtual]

Enable or disable a single button on the bar.

Parameters

button_id	ID of the button to enable or disable.
enable	true to enable the button, false to disable it.

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::GetActiveItem ( ) const [virtual]

Returns the active item of the button bar or NULL if there is none.

The active button is the one being clicked.

Since

2.9.5

virtual size_t wxRibbonButtonBar::GetButtonCount ( ) const [virtual]

Returns the number of buttons in this button bar.

Since

2.9.4

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::GetHoveredItem ( ) const [virtual]

Returns the hovered item of the button bar or NULL if there is none.

The hovered button is the one the mouse is over.

Since

2.9.5

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::GetItem (size_t *n*) const [virtual]

Returns the N-th button of the bar.

See also

GetButtonCount()

Since

2.9.5

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::GetItemById (int *id*) const [virtual]

Returns the first button having a given id or NULL if none matches.

Since

2.9.5

void* wxRibbonButtonBar::GetItemClientData (const wxRibbonButtonBarButtonBase * *item*) const

Get the client data associated with a button.

Since

2.9.5

wxClientData* wxRibbonButtonBar::GetItemClientObject (const wxRibbonButtonBarButtonBase * *item*) const

Get the client object associated with a button.

Since

2.9.5

virtual int wxRibbonButtonBar::GetItemId (wxRibbonButtonBarButtonBase *) const [virtual]

Returns the id of a button.

Since

2.9.5

bool wxRibbonButtonBar::GetShowToolTipsForDisabled ( ) const

Sets whether tooltips should be shown for disabled buttons or not.

You may wish to show it to explain why a button is disabled or what it normally does when enabled.

Since

2.9.5

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::InsertButton (size_t *pos*, int *button_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxString & *help_string*, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL) [virtual]

Inserts a button to the button bar (simple version) at the given position.

See also

AddButton()

Since

2.9.4

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::InsertButton (size_t *pos*, int *button_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxBitmap & *bitmap_small* = wxNullBitmap, const wxBitmap & *bitmap_disabled* = wxNullBitmap, const wxBitmap & *bitmap_small_disabled* = wxNullBitmap, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL, const wxString & *help_string* = wxEmptyString) [virtual]

Insert a button to the button bar at the given position.

Parameters

pos	Position of the new button in the button bar.
button_id	ID of the new button (used for event callbacks).
label	Label of the new button.
bitmap	Large bitmap of the new button. Must be the same size as all other large bitmaps used on the button bar.
bitmap_small	Small bitmap of the new button. If left as null, then a small bitmap will be automatically generated. Must be the same size as all other small bitmaps used on the button bar.
bitmap_disabled	Large bitmap of the new button when it is disabled. If left as null, then a bitmap will be automatically generated from *bitmap*.
bitmap_small_disabled	Small bitmap of the new button when it is disabled. If left as null, then a bitmap will be automatically generated from *bitmap_small*.
kind	The kind of button to add.
help_string	The UI help string to associate with the new button.

Returns

An opaque pointer which can be used only with other button bar methods.

See also

InsertDropdownButton()
InsertHybridButton()
InsertToggleButton()
AddButton()

Since

2.9.4

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::InsertDropdownButton (size_t *pos*, int *button_id*, const wxString & *label*, const wxBitmap & *bitmap*, const wxString & *help_string* = wxEmptyString) [virtual]

Inserts a dropdown button to the button bar (simple version) at the given position.

See also

InsertButton()
AddDropdownButton()
AddButton()

Since

2.9.4

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::InsertHybridButton (size_t *pos,* int *button_id,* const wxString & *label,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Inserts a hybrid button to the button bar (simple version) at the given position.

See also

InsertButton()
AddHybridButton()
AddButton()

Since

2.9.4

virtual wxRibbonButtonBarButtonBase* wxRibbonButtonBar::InsertToggleButton (size_t *pos,* int *button_id,* const wxString & *label,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Inserts a toggle button to the button bar (simple version) at the given position.

See also

InsertButton()
AddToggleButton()
AddButton()

Since

2.9.4

virtual bool wxRibbonButtonBar::Realize ( ) [virtual]

Calculate button layouts and positions.

Must be called after buttons are added to the button bar, as otherwise the newly added buttons will not be displayed. In normal situations, it will be called automatically when wxRibbonBar::Realize() is called.

Reimplemented from wxRibbonControl.

void wxRibbonButtonBar::SetItemClientData (wxRibbonButtonBarButtonBase * *item,* void * *data*)

Set the client data associated with a button.

Please, note that you cannot use both client object and client data.

Since

2.9.5

void wxRibbonButtonBar::SetItemClientObject (wxRibbonButtonBarButtonBase * *item,* wxClientData * *data*)

Set the client object associated with a button.

The button bar owns the given object and takes care of its deletion. Please, note that you cannot use both client object and client data.

Since

2.9.5

void wxRibbonButtonBar::SetShowToolTipsForDisabled (bool *show*)

Indicates whether tooltips are shown for disabled buttons.

By default they are not shown.

Since

2.9.5

virtual void wxRibbonButtonBar::ToggleButton (int *button_id,* bool *checked*) `[virtual]`

Set a toggle button to the checked or unchecked state.

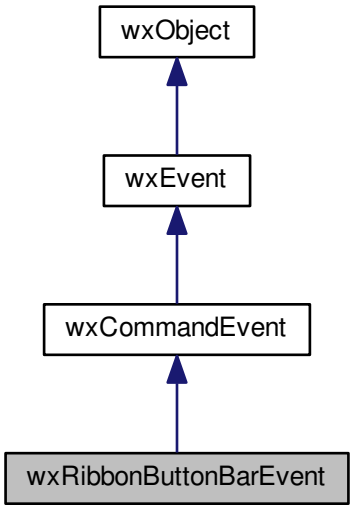
Parameters

button_id	ID of the toggle button to manipulate.
checked	true to set the button to the toggled/pressed/checked state, false to set it to the untoggled/unpressed/unchecked state.

21.598 wxRibbonButtonBarEvent Class Reference

`#include <wx/ribbon/buttonbar.h>`

Inheritance diagram for wxRibbonButtonBarEvent:



21.598.1 Detailed Description

Event used to indicate various actions relating to a button on a wxRibbonButtonBar.

For toggle buttons, IsChecked() can be used to test the state of the button.

See wxRibbonButtonBar for available event types.

Library: wxRibbon

Category: Events, Ribbon User Interface

See also

wxRibbonBar

Public Member Functions

- wxRibbonButtonBarEvent (wxEventType command_type=wxEVT_NULL, int win_id=0, wxRibbonButtonBar *bar=NULL, wxRibbonButtonBarButtonBase *button=NULL)

  Constructor.
- wxRibbonButtonBar * GetBar ()

  Returns the bar which contains the button which the event relates to.
- void SetBar (wxRibbonButtonBar *bar)

  Sets the button bar relating to this event.
- wxRibbonButtonBarButtonBase * GetButton ()

  Returns the button which the event relates to.
- void SetButton (wxRibbonButtonBarButtonBase *bar)

Sets the button relating to this event.

- bool PopupMenu (wxMenu *menu)

  Display a popup menu as a result of this (dropdown clicked) event.

Additional Inherited Members

21.598.2 Constructor & Destructor Documentation

wxRibbonButtonBarEvent::wxRibbonButtonBarEvent (wxEventType *command_type* = wxEVT_NULL, int *win_id* = 0, wxRibbonButtonBar * *bar* = NULL, wxRibbonButtonBarButtonBase * *button* = NULL)

Constructor.

21.598.3 Member Function Documentation

wxRibbonButtonBar* wxRibbonButtonBarEvent::GetBar ( )

Returns the bar which contains the button which the event relates to.

wxRibbonButtonBarButtonBase* wxRibbonButtonBarEvent::GetButton ( )

Returns the button which the event relates to.

Since

2.9.5

bool wxRibbonButtonBarEvent::PopupMenu (wxMenu * *menu*)

Display a popup menu as a result of this (dropdown clicked) event.

void wxRibbonButtonBarEvent::SetBar (wxRibbonButtonBar * *bar*)

Sets the button bar relating to this event.

void wxRibbonButtonBarEvent::SetButton (wxRibbonButtonBarButtonBase * *bar*)

Sets the button relating to this event.

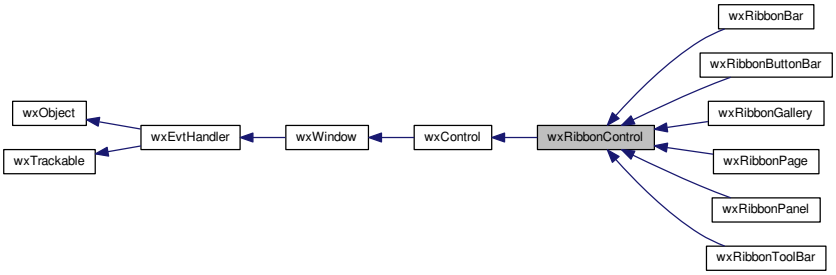
Since

2.9.5

21.599 wxRibbonControl Class Reference

`#include <wx/ribbon/control.h>`

Inheritance diagram for wxRibbonControl:



21.599.1 Detailed Description

wxRibbonControl serves as a base class for all controls which share the ribbon characteristics of having a ribbon art provider, and (optionally) non-continuous resizing.

Despite what the name may imply, it is not the top-level control for creating a ribbon interface - that is wxRibbonBar.

Ribbon controls often have a region which is "transparent", and shows the contents of the ribbon page or panel behind it. If implementing a new ribbon control, then it may be useful to realise that this effect is done by the art provider when painting the background of the control, and hence in the paint handler for the new control, you should call a draw background method on the art provider (wxRibbonArtProvider::DrawButtonBarBackground() and wxRibbonArtProvider::DrawToolBarBackground() typically just redraw what is behind the rectangle being painted) if you want transparent regions.

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonControl ()

  Constructor.
- wxRibbonControl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxControlNameStr)

  Constructor.
- virtual void SetArtProvider (wxRibbonArtProvider *art)

  Set the art provider to be used.
- wxRibbonArtProvider * GetArtProvider () const

  Get the art provider to be used.
- virtual bool IsSizingContinuous () const
- wxSize GetNextSmallerSize (wxOrientation direction) const

  If sizing is not continuous, then return a suitable size for the control which is smaller than the current size.
- wxSize GetNextSmallerSize (wxOrientation direction, wxSize relative_to) const

  If sizing is not continuous, then return a suitable size for the control which is smaller than the given size.
- wxSize GetNextLargerSize (wxOrientation direction) const

  If sizing is not continuous, then return a suitable size for the control which is larger than the current size.

- wxSize GetNextLargerSize (wxOrientation direction, wxSize relative_to) const

  If sizing is not continuous, then return a suitable size for the control which is larger than the given size.
- virtual bool Realize ()

  Perform initial size and layout calculations after children have been added, and/or realize children.
- bool Realise ()

  Alias for Realize().
- virtual wxRibbonBar * GetAncestorRibbonBar () const

  Get the first ancestor which is a wxRibbonBar (or derived) or NULL if not having such parent.
- virtual wxSize GetBestSizeForParentSize (const wxSize &parentSize) const

  Finds the best width and height given the parent's width and height.

Protected Member Functions

- virtual wxSize DoGetNextSmallerSize (wxOrientation direction, wxSize relative_to) const

  Implementation of GetNextSmallerSize().
- virtual wxSize DoGetNextLargerSize (wxOrientation direction, wxSize relative_to) const

  Implementation of GetNextLargerSize().

Additional Inherited Members

21.599.2 Constructor & Destructor Documentation

wxRibbonControl::wxRibbonControl ( )

Constructor.

wxRibbonControl::wxRibbonControl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxControlNameStr`)

Constructor.

If *parent* is a wxRibbonControl with a non-NULL art provider, then the art provider of new control is set to that of *parent*.

21.599.3 Member Function Documentation

virtual wxSize wxRibbonControl::DoGetNextLargerSize (wxOrientation *direction*, wxSize *relative_to*) const `[protected], [virtual]`

Implementation of GetNextLargerSize().

Controls which have non-continuous sizing must override this virtual function rather than GetNextLargerSize().

virtual wxSize wxRibbonControl::DoGetNextSmallerSize (wxOrientation *direction*, wxSize *relative_to*) const `[protected], [virtual]`

Implementation of GetNextSmallerSize().

Controls which have non-continuous sizing must override this virtual function rather than GetNextSmallerSize().

virtual wxRibbonBar* wxRibbonControl::GetAncestorRibbonBar ( ) const `[virtual]`

Get the first ancestor which is a wxRibbonBar (or derived) or NULL if not having such parent.

Since

2.9.4

wxRibbonArtProvider* wxRibbonControl::GetArtProvider ( ) const

Get the art provider to be used.

Note that until an art provider has been set in some way, this function may return NULL.

virtual wxSize wxRibbonControl::GetBestSizeForParentSize (const wxSize & *parentSize*) const `[virtual]`

Finds the best width and height given the parent's width and height.

Used to implement the wxRIBBON_PANEL_FLEXIBLE panel style.

wxSize wxRibbonControl::GetNextLargerSize (wxOrientation *direction*) const

If sizing is not continuous, then return a suitable size for the control which is larger than the current size.

Parameters

direction	The direction(s) in which the size should increase.

Returns

The current size if there is no larger size, otherwise a suitable size which is larger in the given direction(s), and the same as the current size in the other direction (if any).

See also

IsSizingContinuous()

wxSize wxRibbonControl::GetNextLargerSize (wxOrientation *direction*, wxSize *relative_to*) const

If sizing is not continuous, then return a suitable size for the control which is larger than the given size.

Parameters

direction	The direction(s) in which the size should increase.
relative_to	The size for which a larger size should be found.

Returns

relative_to if there is no larger size, otherwise a suitable size which is larger in the given direction(s), and the same as *relative_to* in the other direction (if any).

See also

IsSizingContinuous()
DoGetNextLargerSize()

wxSize wxRibbonControl::GetNextSmallerSize (wxOrientation *direction*) const

If sizing is not continuous, then return a suitable size for the control which is smaller than the current size.

Parameters

direction	The direction(s) in which the size should reduce.

Returns

The current size if there is no smaller size, otherwise a suitable size which is smaller in the given direction(s), and the same as the current size in the other direction (if any).

See also

IsSizingContinuous()

wxSize wxRibbonControl::GetNextSmallerSize (wxOrientation *direction,* wxSize *relative_to*) const

If sizing is not continuous, then return a suitable size for the control which is smaller than the given size.

Parameters

direction	The direction(s) in which the size should reduce.
relative_to	The size for which a smaller size should be found.

Returns

relative_to if there is no smaller size, otherwise a suitable size which is smaller in the given direction(s), and the same as *relative_to* in the other direction (if any).

See also

IsSizingContinuous()
DoGetNextSmallerSize()

virtual bool wxRibbonControl::IsSizingContinuous ( ) const [virtual]

Returns

true if this window can take any size (greater than its minimum size), false if it can only take certain sizes.

See also

GetNextSmallerSize()
GetNextLargerSize()

bool wxRibbonControl::Realise ( )

Alias for Realize().

virtual bool wxRibbonControl::Realize ( ) [virtual]

Perform initial size and layout calculations after children have been added, and/or realize children.

Reimplemented in wxRibbonButtonBar, wxRibbonBar, wxRibbonToolBar, wxRibbonPanel, and wxRibbonPage.

virtual void wxRibbonControl::SetArtProvider (**wxRibbonArtProvider** * *art*) [virtual]

Set the art provider to be used.

In many cases, setting the art provider will also set the art provider on all child windows which extend wxRibbonControl.

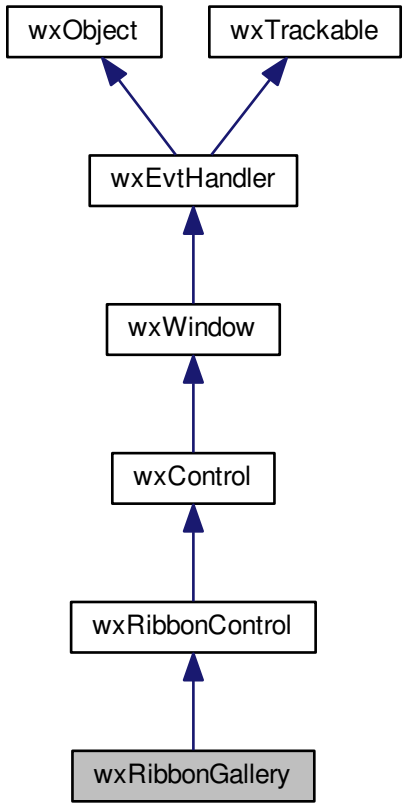
In most cases, controls will not take ownership of the given pointer, with the notable exception being wxRibbonBar::SetArtProvider().

Reimplemented in wxRibbonPanel, wxRibbonBar, and wxRibbonPage.

21.600 wxRibbonGallery Class Reference

`#include <wx/ribbon/gallery.h>`

Inheritance diagram for wxRibbonGallery:



21.600.1 Detailed Description

A ribbon gallery is like a wxListBox, but for bitmaps rather than strings.

It displays a collection of bitmaps arranged in a grid and allows the user to choose one. As there are typically more bitmaps in a gallery than can be displayed in the space used for a ribbon, a gallery always has scroll buttons to allow

the user to navigate through the entire gallery. It also has an "extension" button, the behaviour of which is outside the scope of the gallery control itself, though it typically displays some kind of dialog related to the gallery.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxRibbonGalleryEvent& event)

Event macros for events emitted by this class:

- EVT_RIBBONGALLERY_SELECTED(id, func): Triggered when the user selects an item from the gallery. Note that the ID is that of the gallery, not of the item.
- EVT_RIBBONGALLERY_CLICKED(id, func): Similar to EVT_RIBBONGALLERY_SELECTED but triggered every time a gallery item is clicked, even if it is already selected. Note that the ID of the event is that of the gallery, not of the item, just as above. This event is available since wxWidgets 2.9.2.
- EVT_RIBBONGALLERY_HOVER_CHANGED(id, func): Triggered when the item being hovered over by the user changes. The item in the event will be the new item being hovered, or NULL if there is no longer an item being hovered. Note that the ID is that of the gallery, not of the item.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_BUTTON(id, func): Triggered when the "extension" button of the gallery is pressed.

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonGallery ()

  Default constructor.
- wxRibbonGallery (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Construct a ribbon gallery with the given parameters.
- virtual ~wxRibbonGallery ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Create a gallery in two-step gallery construction.
- void Clear ()

  Remove all items from the gallery.
- bool IsEmpty () const

  Query if the gallery has no items in it.
- unsigned int GetCount () const

  Get the number of items in the gallery.
- wxRibbonGalleryItem * GetItem (unsigned int n)

Get an item by index.

- wxRibbonGalleryItem * Append (const wxBitmap &bitmap, int id)

  Add an item to the gallery (with no client data).
- wxRibbonGalleryItem * Append (const wxBitmap &bitmap, int id, void *clientData)

  Add an item to the gallery (with simple client data).
- wxRibbonGalleryItem * Append (const wxBitmap &bitmap, int id, wxClientData *clientData)

  Add an item to the gallery (with complex client data)
- void SetItemClientObject (wxRibbonGalleryItem *item, wxClientData *data)

  Set the client object associated with a gallery item.
- wxClientData * GetItemClientObject (const wxRibbonGalleryItem *item) const

  Get the client object associated with a gallery item.
- void SetItemClientData (wxRibbonGalleryItem *item, void *data)

  Set the client data associated with a gallery item.
- void * GetItemClientData (const wxRibbonGalleryItem *item) const

  Get the client data associated with a gallery item.
- void SetSelection (wxRibbonGalleryItem *item)

  Set the selection to the given item, or removes the selection if item == NULL.
- wxRibbonGalleryItem * GetSelection () const

  Get the currently selected item, or NULL if there is none.
- wxRibbonGalleryItem * GetHoveredItem () const

  Get the currently hovered item, or NULL if there is none.
- wxRibbonGalleryItem * GetActiveItem () const

  Get the currently active item, or NULL if there is none.
- wxRibbonGalleryButtonState GetUpButtonState () const

  Get the state of the scroll up button.
- wxRibbonGalleryButtonState GetDownButtonState () const

  Get the state of the scroll down button.
- wxRibbonGalleryButtonState GetExtensionButtonState () const

  Get the state of the "extension" button.
- bool IsHovered () const

  Query is the mouse is currently hovered over the gallery.
- virtual bool ScrollLines (int lines)

  Scroll the gallery contents by some amount.
- bool ScrollPixels (int pixels)

  Scroll the gallery contents by some fine-grained amount.
- void EnsureVisible (const wxRibbonGalleryItem *item)

  Scroll the gallery to ensure that the given item is visible.

Additional Inherited Members

21.600.2 Constructor & Destructor Documentation

wxRibbonGallery::wxRibbonGallery ( )

Default constructor.

With this constructor, Create() should be called in order to create the gallery.

wxRibbonGallery::wxRibbonGallery (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Construct a ribbon gallery with the given parameters.

Parameters

parent	Parent window for the gallery (typically a wxRibbonPanel).
id	An identifier for the gallery. `wxID_ANY` is taken to mean a default.
pos	Initial position of the gallery.
size	Initial size of the gallery.
style	Gallery style, currently unused.

virtual wxRibbonGallery::~wxRibbonGallery ( ) [virtual]

Destructor.

21.600.3 Member Function Documentation

wxRibbonGalleryItem* wxRibbonGallery::Append (const wxBitmap & *bitmap*, int *id*)

Add an item to the gallery (with no client data).

Parameters

bitmap	The bitmap to display for the item. Note that all items must have equally sized bitmaps.
id	ID number to associate with the item. Not currently used for anything important.

wxRibbonGalleryItem* wxRibbonGallery::Append (const wxBitmap & *bitmap*, int *id*, void * *clientData*)

Add an item to the gallery (with simple client data).

Parameters

bitmap	The bitmap to display for the item. Note that all items must have equally sized bitmaps.
id	ID number to associate with the item. Not currently used for anything important.
clientData	An opaque pointer to associate with the item.

wxRibbonGalleryItem* wxRibbonGallery::Append (const wxBitmap & *bitmap*, int *id*, wxClientData * *clientData*)

Add an item to the gallery (with complex client data)

Parameters

bitmap	The bitmap to display for the item. Note that all items must have equally sized bitmaps.
id	ID number to associate with the item. Not currently used for anything important.
clientData	An object which contains data to associate with the item. The item takes ownership of this pointer, and will delete it when the item client data is changed to something else, or when the item is destroyed.

void wxRibbonGallery::Clear ( )

Remove all items from the gallery.

bool wxRibbonGallery::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Create a gallery in two-step gallery construction.

Should only be called when the default constructor is used, and arguments have the same meaning as in the full constructor.

void wxRibbonGallery::EnsureVisible (const wxRibbonGalleryItem * *item*)

Scroll the gallery to ensure that the given item is visible.

wxRibbonGalleryItem* wxRibbonGallery::GetActiveItem ( ) const

Get the currently active item, or NULL if there is none.

The active item is the item being pressed by the user, and will thus become the selected item if the user releases the mouse button.

unsigned int wxRibbonGallery::GetCount ( ) const

Get the number of items in the gallery.

wxRibbonGalleryButtonState wxRibbonGallery::GetDownButtonState ( ) const

Get the state of the scroll down button.

wxRibbonGalleryButtonState wxRibbonGallery::GetExtensionButtonState ( ) const

Get the state of the "extension" button.

wxRibbonGalleryItem* wxRibbonGallery::GetHoveredItem ( ) const

Get the currently hovered item, or NULL if there is none.

The hovered item is the item underneath the mouse cursor.

wxRibbonGalleryItem* wxRibbonGallery::GetItem (unsigned int *n*)

Get an item by index.

void* wxRibbonGallery::GetItemClientData (const wxRibbonGalleryItem * *item*) const

Get the client data associated with a gallery item.

wxClientData* wxRibbonGallery::GetItemClientObject (const wxRibbonGalleryItem * *item*) const

Get the client object associated with a gallery item.

wxRibbonGalleryItem* wxRibbonGallery::GetSelection ( ) const

Get the currently selected item, or NULL if there is none.

The selected item is set by SetSelection(), or by the user clicking on an item.

wxRibbonGalleryButtonState wxRibbonGallery::GetUpButtonState ( ) const

Get the state of the scroll up button.

bool wxRibbonGallery::IsEmpty ( ) const

Query if the gallery has no items in it.

bool wxRibbonGallery::IsHovered ( ) const

Query is the mouse is currently hovered over the gallery.

Returns

true if the cursor is within the bounds of the gallery (not just hovering over an item), false otherwise.

virtual bool wxRibbonGallery::ScrollLines (int *lines*) [virtual]

Scroll the gallery contents by some amount.

Parameters

lines	Positive values scroll toward the end of the gallery, while negative values scroll toward the start.

Returns

true if the gallery scrolled at least one pixel in the given direction, false if it did not scroll.

Reimplemented from wxWindow.

bool wxRibbonGallery::ScrollPixels (int *pixels*)

Scroll the gallery contents by some fine-grained amount.

Parameters

pixels	Positive values scroll toward the end of the gallery, while negative values scroll toward the start.

Returns

true if the gallery scrolled at least one pixel in the given direction, false if it did not scroll.

void wxRibbonGallery::SetItemClientData (wxRibbonGalleryItem * *item,* void * *data*)

Set the client data associated with a gallery item.

void wxRibbonGallery::SetItemClientObject (wxRibbonGalleryItem * *item,* wxClientData * *data*)

Set the client object associated with a gallery item.

void wxRibbonGallery::SetSelection (wxRibbonGalleryItem * *item*)

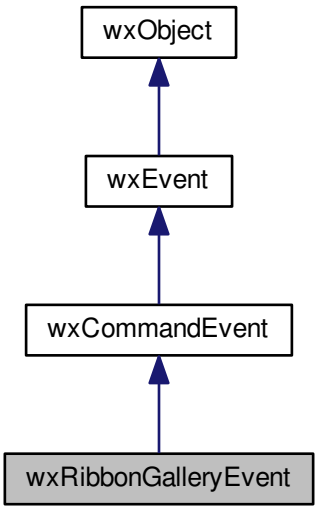
Set the selection to the given item, or removes the selection if *item* == NULL.

Note that this not cause any events to be emitted.

21.601 wxRibbonGalleryEvent Class Reference

`#include <wx/ribbon/gallery.h>`

Inheritance diagram for wxRibbonGalleryEvent:



21.601.1 Detailed Description

Library: wxRibbon

Category: Events, Ribbon User Interface

See also

wxRibbonBar

Public Member Functions

- wxRibbonGalleryEvent (wxEventType command_type=wxEVT_NULL, int win_id=0, wxRibbonGallery *gallery=NULL, wxRibbonGalleryItem *item=NULL)

  Constructor.
- wxRibbonGallery * GetGallery ()

  Returns the gallery which the event relates to.
- wxRibbonGalleryItem * GetGalleryItem ()

Returns the gallery item which the event relates to, or NULL if it does not relate to an item.

- void SetGallery (wxRibbonGallery *gallery)

  Sets the gallery relating to this event.

- void SetGalleryItem (wxRibbonGalleryItem *item)

  Sets the gallery item relating to this event.

Additional Inherited Members

21.601.2 Constructor & Destructor Documentation

wxRibbonGalleryEvent::wxRibbonGalleryEvent (wxEventType *command_type* = wxEVT_NULL, int *win_id* = 0, wxRibbonGallery * *gallery* = NULL, wxRibbonGalleryItem * *item* = NULL)

Constructor.

21.601.3 Member Function Documentation

wxRibbonGallery* wxRibbonGalleryEvent::GetGallery ( )

Returns the gallery which the event relates to.

wxRibbonGalleryItem* wxRibbonGalleryEvent::GetGalleryItem ( )

Returns the gallery item which the event relates to, or NULL if it does not relate to an item.

void wxRibbonGalleryEvent::SetGallery (wxRibbonGallery * *gallery*)

Sets the gallery relating to this event.

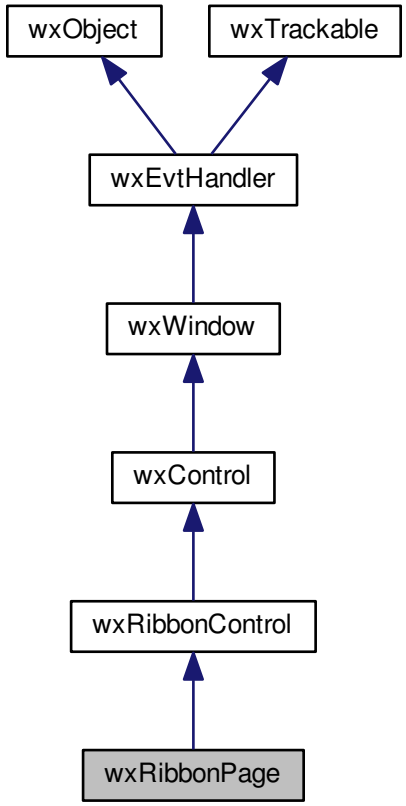
void wxRibbonGalleryEvent::SetGalleryItem (wxRibbonGalleryItem * *item*)

Sets the gallery item relating to this event.

21.602 wxRibbonPage Class Reference

```
#include <wx/ribbon/page.h>
```

Inheritance diagram for wxRibbonPage:



21.602.1 Detailed Description

Container for related ribbon panels, and a tab within a ribbon bar.

See also

wxRibbonBar
wxRibbonPanel

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonPage ()

  Default constructor.
- wxRibbonPage (wxRibbonBar *parent, wxWindowID id=wxID_ANY, const wxString &label=wxEmptyString, const wxBitmap &icon=wxNullBitmap, long style=0)

Constructs a ribbon page, which must be a child of a ribbon bar.

- virtual ~wxRibbonPage ()

  Destructor.

- bool Create (wxRibbonBar *parent, wxWindowID id=wxID_ANY, const wxString &label=wxEmptyString, const wxBitmap &icon=wxNullBitmap, long style=0)

  Create a ribbon page in two-step ribbon page construction.

- void SetArtProvider (wxRibbonArtProvider *art)

  Set the art provider to be used.

- wxBitmap & GetIcon ()

  Get the icon used for the page in the ribbon bar tab area (only displayed if the ribbon bar is actually showing icons).

- void SetSizeWithScrollButtonAdjustment (int x, int y, int width, int height)

  Set the size of the page and the external scroll buttons (if any).

- void AdjustRectToIncludeScrollButtons (wxRect *rect) const

  Expand a rectangle of the page to include external scroll buttons (if any).

- bool DismissExpandedPanel ()

  Dismiss the current externally expanded panel, if there is one.

- virtual bool Realize ()

  Perform a full re-layout of all panels on the page.

- virtual bool ScrollLines (int lines)

  Scroll the page by some amount up / down / left / right.

- bool ScrollPixels (int pixels)

  Scroll the page by a set number of pixels up / down / left / right.

- bool ScrollSections (int sections)

  Scroll the page by an entire child section.

- wxOrientation GetMajorAxis () const

  Get the direction in which ribbon panels are stacked within the page.

Additional Inherited Members

21.602.2 Constructor & Destructor Documentation

wxRibbonPage::wxRibbonPage ( )

Default constructor.

With this constructor, Create() should be called in order to create the ribbon page.

wxRibbonPage::wxRibbonPage (wxRibbonBar * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *label* = wxEmptyString, const wxBitmap & *icon* = wxNullBitmap, long *style* = 0)

Constructs a ribbon page, which must be a child of a ribbon bar.

Parameters

parent	Pointer to a parent wxRibbonBar (unlike most controls, a wxRibbonPage can only have wxRibbonBar as a parent).
id	Window identifier.
label	Label to be used in the wxRibbonBar's tab list for this page (if the ribbon bar is set to display labels).

icon	Icon to be used in the wxRibbonBar's tab list for this page (if the ribbon bar is set to display icons).
style	Currently unused, should be zero.

virtual wxRibbonPage::~wxRibbonPage ( ) [virtual]

Destructor.

21.602.3 Member Function Documentation

void wxRibbonPage::AdjustRectToIncludeScrollButtons (wxRect * *rect*) const

Expand a rectangle of the page to include external scroll buttons (if any).

When no scroll buttons are shown, has no effect.

Parameters

in,out	*rect*	The rectangle to adjust. The width and height will not be reduced, and the x and y will not be increased.

bool wxRibbonPage::Create (wxRibbonBar * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *label* = wxEmptyString, const wxBitmap & *icon* = wxNullBitmap, long *style* = 0)

Create a ribbon page in two-step ribbon page construction.

Should only be called when the default constructor is used, and arguments have the same meaning as in the full constructor.

bool wxRibbonPage::DismissExpandedPanel ( )

Dismiss the current externally expanded panel, if there is one.

When a ribbon panel automatically minimises, it can be externally expanded into a floating window. When the user clicks a button in such a panel, the panel should generally re-minimise. Event handlers for buttons on ribbon panels should call this method to achieve this behaviour.

Returns

true if a panel was minimised, false otherwise.

wxBitmap& wxRibbonPage::GetIcon ( )

Get the icon used for the page in the ribbon bar tab area (only displayed if the ribbon bar is actually showing icons).

wxOrientation wxRibbonPage::GetMajorAxis ( ) const

Get the direction in which ribbon panels are stacked within the page.

This is controlled by the style of the containing wxRibbonBar, meaning that all pages within a bar will have the same major axis. As well as being the direction in which panels are stacked, it is also the axis in which scrolling will occur (when required).

Returns

wxHORIZONTAL or wxVERTICAL (never wxBOTH).

virtual bool wxRibbonPage::Realize ( ) [virtual]

Perform a full re-layout of all panels on the page.

Should be called after panels are added to the page, or the sizing behaviour of a panel on the page changes (i.e. due to children being added to it). Usually called automatically when wxRibbonBar::Realize() is called.

Will invoke wxRibbonPanel::Realize() for all child panels.

Reimplemented from wxRibbonControl.

virtual bool wxRibbonPage::ScrollLines (int *lines*) [virtual]

Scroll the page by some amount up / down / left / right.

When the page's children are too big to fit in the onscreen area given to the page, scroll buttons will appear, and the page can be programmatically scrolled. Positive values of *lines* will scroll right or down, while negative values will scroll up or left (depending on the direction in which panels are stacked). A line is equivalent to a constant number of pixels.

Returns

true if the page scrolled at least one pixel in the given direction, false if it did not scroll.

See also

GetMajorAxis()
ScrollPixels()
ScrollSections()

Reimplemented from wxWindow.

bool wxRibbonPage::ScrollPixels (int *pixels*)

Scroll the page by a set number of pixels up / down / left / right.

When the page's children are too big to fit in the onscreen area given to the page, scroll buttons will appear, and the page can be programmatically scrolled. Positive values of *lines* will scroll right or down, while negative values will scroll up or left (depending on the direction in which panels are stacked).

Returns

true if the page scrolled at least one pixel in the given direction, false if it did not scroll.

See also

GetMajorAxis()
ScrollLines()
ScrollSections()

bool wxRibbonPage::ScrollSections (int *sections*)

Scroll the page by an entire child section.

The *sections* parameter value should be 1 or -1. This will scroll enough to uncover a partially visible child section or totally uncover the next child section that may not be visible at all.

Returns

true if the page scrolled at least one pixel in the given direction, false if it did not scroll.

See also

ScrollPixels()
ScrollSections()

Since

2.9.5

void wxRibbonPage::SetArtProvider (wxRibbonArtProvider * *art*) [virtual]

Set the art provider to be used.

Normally called automatically by wxRibbonBar when the page is created, or the art provider changed on the bar.

The new art provider will be propagated to the children of the page.

Reimplemented from wxRibbonControl.

void wxRibbonPage::SetSizeWithScrollButtonAdjustment (int *x*, int *y*, int *width*, int *height*)

Set the size of the page and the external scroll buttons (if any).

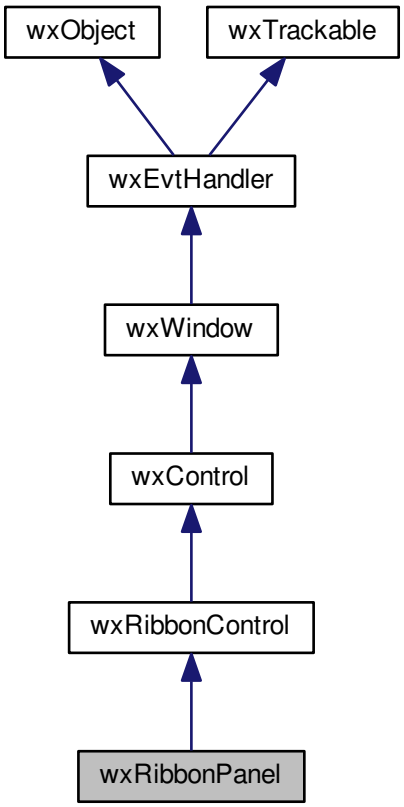
When a page is too small to display all of its children, scroll buttons will appear (and if the page is sized up enough, they will disappear again). Slightly counter-intuitively, these buttons are created as siblings of the page rather than children of the page (to achieve correct cropping and paint ordering of the children and the buttons). When there are no scroll buttons, this function behaves the same as SetSize(), however when there are scroll buttons, it positions them at the edges of the given area, and then calls SetSize() with the remaining area.

This is provided as a separate function to SetSize() rather than within the implementation of SetSize(), as interacting algorithms may not expect SetSize() to also set the size of siblings.

21.603 wxRibbonPanel Class Reference

`#include <wx/ribbon/panel.h>`

Inheritance diagram for wxRibbonPanel:



21.603.1 Detailed Description

Serves as a container for a group of (ribbon) controls.

A wxRibbonPage will typically have panels for children, with the controls for that page placed on the panels.

A panel adds a border and label to a group of controls, and can be minimised (either automatically to conserve space, or manually by the user).

Non ribbon controls can be placed on a panel using wxSizers to manage layout. Panel size is governed by the sizer's minimum calculated size and the parent wxRibbonPage's dimensions. For functional and aesthetic reasons it is recommended that ribbon and non ribbon controls are not mixed in one panel.

See also

wxRibbonPage

Styles

This class supports the following styles:

- wxRIBBON_PANEL_DEFAULT_STYLE: Defined as no other flags set.

- wxRIBBON_PANEL_NO_AUTO_MINIMISE: Prevents the panel from automatically minimising to conserve screen space.
- wxRIBBON_PANEL_EXT_BUTTON: Causes an extension button to be shown in the panel's chrome (if the bar in which it is contained has wxRIBBON_BAR_SHOW_PANEL_EXT_BUTTONS set). The behaviour of this button is application controlled, but typically will show an extended drop-down menu relating to the panel.
- wxRIBBON_PANEL_MINIMISE_BUTTON: Causes a (de)minimise button to be shown in the panel's chrome (if the bar in which it is contained has the wxRIBBON_BAR_SHOW_PANEL_MINIMISE_BUTTONS style set). This flag is typically combined with wxRIBBON_PANEL_NO_AUTO_MINIMISE to make a panel which the user always has manual control over when it minimises.
- wxRIBBON_PANEL_STRETCH: Stretches a single panel to fit the parent page.
- wxRIBBON_PANEL_FLEXIBLE: Allows the panel to size in both directions; currently only useful when a single wxRibbonToolBar is the child of the panel, particularly in vertical orientation where the number of rows is dependent on the amount of horizontal space available. Set the minimum and maximum toolbar rows to take full advantage of this wrapping behaviour.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxRibbonPanelEvent& event)

Event macros for events emitted by this class:

- EVT_RIBBONPANEL_EXTBUTTON_ACTIVATED(id, func): Triggered when the user activate the panel extension button.

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonPanel ()

  Default constructor.
- wxRibbonPanel (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &label=wxEmptyString, const wxBitmap &minimised_icon=wxNullBitmap, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxRIBBON_PANEL_DEFAULT_STYLE)

  Constructs a ribbon panel.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &label=wxEmptyString, const wxBitmap &icon=wxNullBitmap, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxRIBBON_PANEL_DEFAULT_STYLE)

  Create a ribbon panel in two-step ribbon panel construction.
- virtual ~wxRibbonPanel ()

  Destructor.
- wxBitmap & GetMinimisedIcon ()

  Get the bitmap to be used in place of the panel children when it is minimised.
- const wxBitmap & GetMinimisedIcon () const
- virtual bool HasExtButton () const

  Test if the panel has an extension button.
- bool IsMinimised () const

  Query if the panel is currently minimised.

- bool IsMinimised (wxSize at_size) const

  Query if the panel would be minimised at a given size.
- bool IsHovered () const

  Query is the mouse is currently hovered over the panel.
- bool IsExtButtonHovered () const

  Query if the mouse is currently hovered over the extension button.
- bool CanAutoMinimise () const

  Query if the panel can automatically minimise itself at small sizes.
- bool ShowExpanded ()

  Show the panel externally expanded.
- bool HideExpanded ()

  Hide the panel's external expansion.
- void SetArtProvider (wxRibbonArtProvider *art)

  Set the art provider to be used.
- bool Realize ()

  Realize all children of the panel.
- wxRibbonPanel * GetExpandedDummy ()

  Get the dummy panel of an expanded panel.
- wxRibbonPanel * GetExpandedPanel ()

  Get the expanded panel of a dummy panel.

Additional Inherited Members

21.603.2 Constructor & Destructor Documentation

wxRibbonPanel::wxRibbonPanel ( )

Default constructor.

With this constructor, Create() should be called in order to create the ribbon panel.

wxRibbonPanel::wxRibbonPanel (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *label* = wxEmptyString, const wxBitmap & *minimised_icon* = wxNullBitmap, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `wxRIBBON_PANEL_DEFAULT_STYLE`)

Constructs a ribbon panel.

Parameters

parent	Pointer to a parent window, which is typically a wxRibbonPage, though it can be any window.
id	Window identifier.
label	Label to be used in the wxRibbonPanel's chrome.
minimised_icon	Icon to be used in place of the panel's children when the panel is minimised.
pos	The initial position of the panel. Not relevant when the parent is a ribbon page, as the position and size of the panel will be dictated by the page.
size	The initial size of the panel. Not relevant when the parent is a ribbon page, as the position and size of the panel will be dictated by the page.
style	Style flags for the panel.

virtual wxRibbonPanel::~wxRibbonPanel ( ) `[virtual]`

Destructor.

21.603.3 Member Function Documentation

bool wxRibbonPanel::CanAutoMinimise ( ) const

Query if the panel can automatically minimise itself at small sizes.

bool wxRibbonPanel::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *label* = wxEmptyString, const wxBitmap & *icon* = wxNullBitmap, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `wxRIBBON_PANEL_DEFAULT_STYLE`)

Create a ribbon panel in two-step ribbon panel construction.

Should only be called when the default constructor is used, and arguments have the same meaning as in the full constructor.

wxRibbonPanel* wxRibbonPanel::GetExpandedDummy ( )

Get the dummy panel of an expanded panel.

Note that this should be called on an expanded panel to get the dummy associated with it - it will return NULL when called on the dummy itself.

See also

ShowExpanded()
GetExpandedPanel()

wxRibbonPanel* wxRibbonPanel::GetExpandedPanel ( )

Get the expanded panel of a dummy panel.

Note that this should be called on a dummy panel to get the expanded panel associated with it - it will return NULL when called on the expanded panel itself.

See also

ShowExpanded()
GetExpandedDummy()

wxBitmap& wxRibbonPanel::GetMinimisedIcon ( )

Get the bitmap to be used in place of the panel children when it is minimised.

const wxBitmap& wxRibbonPanel::GetMinimisedIcon ( ) const

virtual bool wxRibbonPanel::HasExtButton ( ) const `[virtual]`

Test if the panel has an extension button.

Such button is shown in the top right corner of the panel if `wxRIBBON_PANEL_EXT_BUTTON` style is used for it.

Since

2.9.4

Returns

true if the panel and its wxRibbonBar allow it in their styles.

bool wxRibbonPanel::HideExpanded ( )

Hide the panel's external expansion.

Returns

true if the panel was un-expanded, false if it was not (normally due to it not being expanded in the first place).

See also

HideExpanded()
GetExpandedPanel()

bool wxRibbonPanel::IsExtButtonHovered ( ) const

Query if the mouse is currently hovered over the extension button.

Extension button is only shown for panels with `wxRIBBON_PANEL_EXT_BUTTON` style.

Since

2.9.4

bool wxRibbonPanel::IsHovered ( ) const

Query is the mouse is currently hovered over the panel.

Returns

true if the cursor is within the bounds of the panel (i.e. hovered over the panel or one of its children), false otherwise.

bool wxRibbonPanel::IsMinimised ( ) const

Query if the panel is currently minimised.

bool wxRibbonPanel::IsMinimised (wxSize *at_size*) const

Query if the panel would be minimised at a given size.

bool wxRibbonPanel::Realize ( ) [virtual]

Realize all children of the panel.

Reimplemented from wxRibbonControl.

void wxRibbonPanel::SetArtProvider (wxRibbonArtProvider * *art*) [virtual]

Set the art provider to be used.

Normally called automatically by wxRibbonPage when the panel is created, or the art provider changed on the page.

The new art provider will be propagated to the children of the panel.

Reimplemented from wxRibbonControl.

bool wxRibbonPanel::ShowExpanded ( )

Show the panel externally expanded.

When a panel is minimised, it can be shown full-size in a pop-out window, which is referred to as being (externally) expanded. Note that when a panel is expanded, there exist two panels - the original panel (which is referred to as the dummy panel) and the expanded panel. The original is termed a dummy as it sits in the ribbon bar doing nothing, while the expanded panel holds the panel children.

Returns

true if the panel was expanded, false if it was not (possibly due to it not being minimised, or already being expanded).

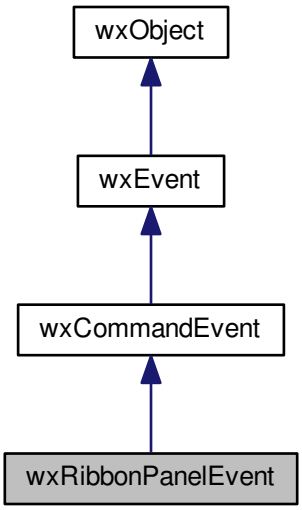
See also

HideExpanded()
GetExpandedPanel()

21.604 wxRibbonPanelEvent Class Reference

`#include <wx/ribbon/panel.h>`

Inheritance diagram for wxRibbonPanelEvent:



21.604.1 Detailed Description

Event used to indicate various actions relating to a wxRibbonPanel.

See wxRibbonPanel for available event types.

Since

2.9.4

Library: wxRibbon

Category: Events, Ribbon User Interface

See also

wxRibbonPanel

Public Member Functions

- wxRibbonPanelEvent (wxEventType command_type=wxEVT_NULL, int win_id=0, wxRibbonPanel *panel=NULL) wxRibbonPanel *GetPanel()

  Constructor.

- void SetPanel (wxRibbonPanel *page)

  Sets the page relating to this event.

Additional Inherited Members

21.604.2 Constructor & Destructor Documentation

wxRibbonPanelEvent::wxRibbonPanelEvent (wxEventType *command_type* = wxEVT_NULL, int *win_id* = 0, wxRibbonPanel * *panel* = NULL)

Constructor.

Returns the panel relating to this event.

21.604.3 Member Function Documentation

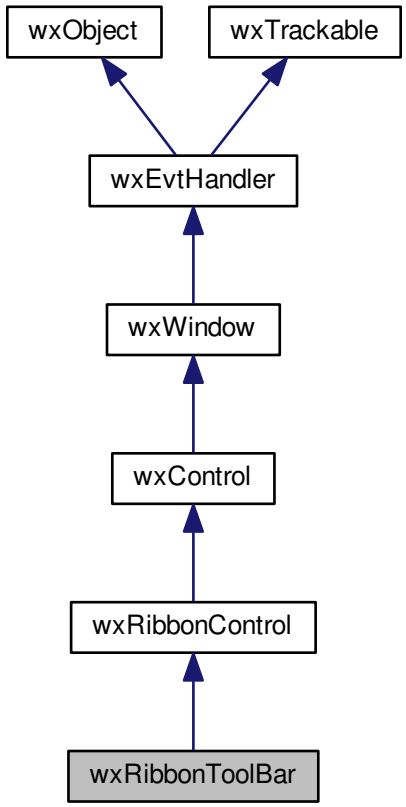
void wxRibbonPanelEvent::SetPanel (wxRibbonPanel * *page*)

Sets the page relating to this event.

21.605 wxRibbonToolBar Class Reference

```
#include <wx/ribbon/toolbar.h>
```

Inheritance diagram for wxRibbonToolBar:



21.605.1 Detailed Description

A ribbon tool bar is similar to a traditional toolbar which has no labels.

It contains one or more tool groups, each of which contains one or more tools. Each tool is represented by a (generally small, i.e. 16x15) bitmap.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxRibbonToolBarEvent& event)

Event macros for events emitted by this class:

- EVT_RIBBONTOOLBAR_CLICKED(id, func): Triggered when the normal (non-dropdown) region of a tool on the tool bar is clicked.
- EVT_RIBBONTOOLBAR_DROPDOWN_CLICKED(id, func): Triggered when the dropdown region of a tool on the tool bar is clicked. wxRibbonToolBarEvent::PopupMenu() should be called by the event handler if it wants to display a popup menu (which is what most dropdown tools should be doing).

Library: wxRibbon

Category: Ribbon User Interface

Public Member Functions

- wxRibbonToolBar ()

  Default constructor.
- wxRibbonToolBar (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Construct a ribbon tool bar with the given parameters.
- virtual ~wxRibbonToolBar ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Create a tool bar in two-step tool bar construction.
- virtual wxRibbonToolBarToolBase * AddTool (int tool_id, const wxBitmap &bitmap, const wxString &help_string, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL)

  Add a tool to the tool bar (simple version).
- virtual wxRibbonToolBarToolBase * AddDropdownTool (int tool_id, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Add a dropdown tool to the tool bar (simple version).
- virtual wxRibbonToolBarToolBase * AddHybridTool (int tool_id, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Add a hybrid tool to the tool bar (simple version).
- virtual wxRibbonToolBarToolBase * AddToggleTool (int tool_id, const wxBitmap &bitmap, const wxString &help_string)

  Add a toggle tool to the tool bar (simple version).
- virtual wxRibbonToolBarToolBase * AddTool (int tool_id, const wxBitmap &bitmap, const wxBitmap &bitmap_disabled=wxNullBitmap, const wxString &help_string=wxEmptyString, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL, wxObject *client_data=NULL)

  Add a tool to the tool bar.
- virtual wxRibbonToolBarToolBase * AddSeparator ()

  Add a separator to the tool bar.
- virtual wxRibbonToolBarToolBase * InsertTool (size_t pos, int tool_id, const wxBitmap &bitmap, const wxString &help_string, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL)

  Insert a tool to the tool bar (simple version) as the specified position.
- virtual wxRibbonToolBarToolBase * InsertDropdownTool (size_t pos, int tool_id, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Insert a dropdown tool to the tool bar (simple version) as the specified position.
- virtual wxRibbonToolBarToolBase * InsertHybridTool (size_t pos, int tool_id, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Insert a hybrid tool to the tool bar (simple version) as the specified position.
- virtual wxRibbonToolBarToolBase * InsertToggleTool (size_t pos, int tool_id, const wxBitmap &bitmap, const wxString &help_string=wxEmptyString)

  Insert a toggle tool to the tool bar (simple version) as the specified position.
- virtual wxRibbonToolBarToolBase * InsertTool (size_t pos, int tool_id, const wxBitmap &bitmap, const wxBitmap &bitmap_disabled=wxNullBitmap, const wxString &help_string=wxEmptyString, wxRibbonButtonKind kind=wxRIBBON_BUTTON_NORMAL, wxObject *client_data=NULL)

  Insert a tool to the tool bar at the specified position.
- virtual wxRibbonToolBarToolBase * InsertSeparator (size_t pos)

Insert a separator to the tool bar at the specified position.

- virtual void ClearTools ()

  Deletes all the tools in the toolbar.
- virtual bool DeleteTool (int tool_id)

  Removes the specified tool from the toolbar and deletes it.
- virtual bool DeleteToolByPos (size_t pos)

  This function behaves like DeleteTool() but it deletes the tool at the specified position and not the one with the given id.
- virtual wxRibbonToolBarToolBase * FindById (int tool_id) const

  Returns a pointer to the tool opaque structure by id or NULL if no corresponding tool is found.
- wxRibbonToolBarToolBase * GetToolByPos (size_t pos) const virtual size_t GetToolCount() const

  Return the opaque pointer corresponding to the given tool.
- virtual int GetToolId (const wxRibbonToolBarToolBase *tool) const

  Return the id assciated to the tool opaque structure.
- virtual wxObject * GetToolClientData (int tool_id) const

  Get any client data associated with the tool.
- virtual bool GetToolEnabled (int tool_id) const

  Called to determine whether a tool is enabled (responds to user input).
- virtual wxString GetToolHelpString (int tool_id) const

  Returns the help string for the given tool.
- virtual wxRibbonButtonKind GetToolKind (int tool_id) const

  Return the kind of the given tool.
- virtual int GetToolPos (int tool_id) const

  Returns the tool position in the toolbar, or `wxNOT_FOUND` if the tool is not found.
- virtual bool GetToolState (int tool_id) const

  Gets the on/off state of a toggle tool.
- virtual bool Realize ()

  Calculate tool layouts and positions.
- virtual void SetRows (int nMin, int nMax=-1)

  Set the number of rows to distribute tool groups over.
- virtual void SetToolClientData (int tool_id, wxObject *clientData)

  Sets the client data associated with the tool.
- virtual void SetToolDisabledBitmap (int tool_id, const wxBitmap &bitmap)

  Sets the bitmap to be used by the tool with the given ID when the tool is in a disabled state.
- virtual void SetToolHelpString (int tool_id, const wxString &helpString)

  Sets the help string shown in tooltip for the given tool.
- virtual void SetToolNormalBitmap (int tool_id, const wxBitmap &bitmap)

  Sets the bitmap to be used by the tool with the given ID.
- virtual void EnableTool (int tool_id, bool enable=true)

  Enable or disable a single tool on the bar.
- virtual void ToggleTool (int tool_id, bool checked)

  Set a toggle tool to the checked or unchecked state.

Additional Inherited Members

21.605.2 Constructor & Destructor Documentation

wxRibbonToolBar::wxRibbonToolBar ( )

Default constructor.

With this constructor, Create() should be called in order to create the tool bar.

wxRibbonToolBar::wxRibbonToolBar (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Construct a ribbon tool bar with the given parameters.

Parameters

parent	Parent window for the tool bar (typically a wxRibbonPanel).
id	An identifier for the toolbar. `wxID_ANY` is taken to mean a default.
pos	Initial position of the tool bar.
size	Initial size of the tool bar.
style	Tool bar style, currently unused.

virtual wxRibbonToolBar::~wxRibbonToolBar ( ) [virtual]

Destructor.

21.605.3 Member Function Documentation

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::AddDropdownTool (int *tool_id,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Add a dropdown tool to the tool bar (simple version).

See also

AddTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::AddHybridTool (int *tool_id,* const wxBitmap & *bitmap,* const wxString & *help_string* = wxEmptyString) [virtual]

Add a hybrid tool to the tool bar (simple version).

See also

AddTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::AddSeparator ( ) [virtual]

Add a separator to the tool bar.

Separators are used to separate tools into groups. As such, a separator is not explicitly drawn, but is visually seen as the gap between tool groups.

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::AddToggleTool (int *tool_id,* const wxBitmap & *bitmap,* const wxString & *help_string*) [virtual]

Add a toggle tool to the tool bar (simple version).

Since

2.9.4

See also

AddTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::AddTool (int *tool_id*, const wxBitmap & *bitmap*, const wxString & *help_string*, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL) [virtual]

Add a tool to the tool bar (simple version).

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::AddTool (int *tool_id*, const wxBitmap & *bitmap*, const wxBitmap & *bitmap_disabled* = wxNullBitmap, const wxString & *help_string* = wxEmptyString, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL, wxObject * *client_data* = NULL) [virtual]

Add a tool to the tool bar.

Parameters

tool_id	ID of the new tool (used for event callbacks).
bitmap	Bitmap to use as the foreground for the new tool. Does not have to be the same size as other tool bitmaps, but should be similar as otherwise it will look visually odd.
bitmap_disabled	Bitmap to use when the tool is disabled. If left as wxNullBitmap, then a bitmap will be automatically generated from *bitmap*.
help_string	The UI help string to associate with the new tool.
kind	The kind of tool to add.
client_data	Client data to associate with the new tool.

Returns

An opaque pointer which can be used only with other tool bar methods.

See also

AddDropdownTool(), AddHybridTool(), AddSeparator(), InsertTool()

virtual void wxRibbonToolBar::ClearTools ( ) [virtual]

Deletes all the tools in the toolbar.

Since

2.9.4

bool wxRibbonToolBar::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Create a tool bar in two-step tool bar construction.

Should only be called when the default constructor is used, and arguments have the same meaning as in the full constructor.

virtual bool wxRibbonToolBar::DeleteTool (int *tool_id*) [virtual]

Removes the specified tool from the toolbar and deletes it.

Parameters

tool_id	ID of the tool to delete.

Returns

true if the tool was deleted, false otherwise.

Since

2.9.4

See also

DeleteToolByPos()

virtual bool wxRibbonToolBar::DeleteToolByPos (size_t *pos*) [virtual]

This function behaves like DeleteTool() but it deletes the tool at the specified position and not the one with the given id.

Useful to delete separators.

Since

2.9.4

virtual void wxRibbonToolBar::EnableTool (int *tool_id*, bool *enable* = true) [virtual]

Enable or disable a single tool on the bar.

Parameters

tool_id	ID of the tool to enable or disable.
enable	true to enable the tool, false to disable it.

Since

2.9.4

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::FindById (int *tool_id*) const [virtual]

Returns a pointer to the tool opaque structure by *id* or NULL if no corresponding tool is found.

Since

2.9.4

wxRibbonToolBarToolBase* wxRibbonToolBar::GetToolByPos (size_t *pos*) const

Return the opaque pointer corresponding to the given tool.

Returns

an opaque pointer, NULL if is a separator or not found.

Since

2.9.4 Returns the number of tools in the toolbar.
2.9.4

virtual wxObject* wxRibbonToolBar::GetToolClientData (int *tool_id*) const `[virtual]`

Get any client data associated with the tool.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().

Returns

Client data, or NULL if there is none.

Since

2.9.4

virtual bool wxRibbonToolBar::GetToolEnabled (int *tool_id*) const `[virtual]`

Called to determine whether a tool is enabled (responds to user input).

Parameters

tool_id	ID of the tool in question, as passed to AddTool().

Returns

true if the tool is enabled, false otherwise.

Since

2.9.4

See also

EnableTool()

virtual wxString wxRibbonToolBar::GetToolHelpString (int *tool_id*) const `[virtual]`

Returns the help string for the given tool.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().

Since

2.9.4

virtual int wxRibbonToolBar::GetToolId (const wxRibbonToolBarToolBase * *tool*) const `[virtual]`

Return the id assciated to the tool opaque structure.

The structure pointer must not be NULL.

Since

2.9.4

virtual wxRibbonButtonKind wxRibbonToolBar::GetToolKind (int *tool_id*) const `[virtual]`

Return the kind of the given tool.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().

Since

2.9.4

virtual int wxRibbonToolBar::GetToolPos (int *tool_id*) const `[virtual]`

Returns the tool position in the toolbar, or `wxNOT_FOUND` if the tool is not found.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().

Since

2.9.4

virtual bool wxRibbonToolBar::GetToolState (int *tool_id*) const `[virtual]`

Gets the on/off state of a toggle tool.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().

Returns

true if the tool is toggled on, false otherwise.

See also

ToggleTool()

Since

2.9.4

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::InsertDropdownTool (size_t *pos*, int *tool_id*, const wxBitmap & *bitmap*, const wxString & *help_string* = wxEmptyString) `[virtual]`

Insert a dropdown tool to the tool bar (simple version) as the specified position.

Since

2.9.4

See also

AddDropdownTool(), InsertTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::InsertHybridTool (size_t *pos*, int *tool_id*, const wxBitmap & *bitmap*, const wxString & *help_string* = wxEmptyString) [virtual]

Insert a hybrid tool to the tool bar (simple version) as the specified position.

Since

2.9.4

See also

AddHybridTool(), InsertTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::InsertSeparator (size_t *pos*) [virtual]

Insert a separator to the tool bar at the specified position.

Since

2.9.4

See also

AddSeparator(), InsertTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::InsertToggleTool (size_t *pos*, int *tool_id*, const wxBitmap & *bitmap*, const wxString & *help_string* = wxEmptyString) [virtual]

Insert a toggle tool to the tool bar (simple version) as the specified position.

Since

2.9.4

See also

AddToggleTool(), InsertTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::InsertTool (size_t *pos*, int *tool_id*, const wxBitmap & *bitmap*, const wxString & *help_string*, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL) [virtual]

Insert a tool to the tool bar (simple version) as the specified position.

Since

2.9.4

See also

InsertTool()

virtual wxRibbonToolBarToolBase* wxRibbonToolBar::InsertTool (size_t *pos*, int *tool_id*, const wxBitmap & *bitmap*, const wxBitmap & *bitmap_disabled* = wxNullBitmap, const wxString & *help_string* = wxEmptyString, wxRibbonButtonKind *kind* = wxRIBBON_BUTTON_NORMAL, wxObject * *client_data* = NULL) [virtual]

Insert a tool to the tool bar at the specified position.

Parameters

pos	Position of the new tool (number of tools and separators from the beginning of the toolbar).
tool_id	ID of the new tool (used for event callbacks).
bitmap	Bitmap to use as the foreground for the new tool. Does not have to be the same size as other tool bitmaps, but should be similar as otherwise it will look visually odd.
bitmap_disabled	Bitmap to use when the tool is disabled. If left as wxNullBitmap, then a bitmap will be automatically generated from *bitmap*.
help_string	The UI help string to associate with the new tool.
kind	The kind of tool to add.
client_data	Client data to associate with the new tool.

Returns

An opaque pointer which can be used only with other tool bar methods.

Since

2.9.4

See also

InsertDropdownTool(), InsertHybridTool(), InsertSeparator()

virtual bool wxRibbonToolBar::Realize ( ) [virtual]

Calculate tool layouts and positions.

Must be called after tools are added to the tool bar, as otherwise the newly added tools will not be displayed.

Reimplemented from wxRibbonControl.

virtual void wxRibbonToolBar::SetRows (int *nMin*, int *nMax* = -1) [virtual]

Set the number of rows to distribute tool groups over.

Tool groups can be distributed over a variable number of rows. The way in which groups are assigned to rows is not specified, and the order of groups may change, but they will be distributed in such a way as to minimise the overall size of the tool bar.

Parameters

nMin	The minimum number of rows to use.
nMax	The maximum number of rows to use (defaults to nMin).

virtual void wxRibbonToolBar::SetToolClientData (int *tool_id*, wxObject * *clientData*) [virtual]

Sets the client data associated with the tool.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().
clientData	The client data to use.

Since

2.9.4

virtual void wxRibbonToolBar::SetToolDisabledBitmap (int *tool_id,* const wxBitmap & *bitmap*) `[virtual]`

Sets the bitmap to be used by the tool with the given ID when the tool is in a disabled state.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().
bitmap	Bitmap to use for disabled tools.

Since

2.9.4

virtual void wxRibbonToolBar::SetToolHelpString (int *tool_id,* const wxString & *helpString*) [virtual]

Sets the help string shown in tooltip for the given tool.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().
helpString	A string for the help.

See also

GetToolHelpString()

Since

2.9.4

virtual void wxRibbonToolBar::SetToolNormalBitmap (int *tool_id,* const wxBitmap & *bitmap*) [virtual]

Sets the bitmap to be used by the tool with the given ID.

Parameters

tool_id	ID of the tool in question, as passed to AddTool().
bitmap	Bitmap to use for normals tools.

Since

2.9.4

virtual void wxRibbonToolBar::ToggleTool (int *tool_id,* bool *checked*) [virtual]

Set a toggle tool to the checked or unchecked state.

Parameters

tool_id	ID of the toggle tool to manipulate.
checked	true to set the tool to the toggled/pressed/checked state, false to set it to the untoggled/unpressed/unchecked state.

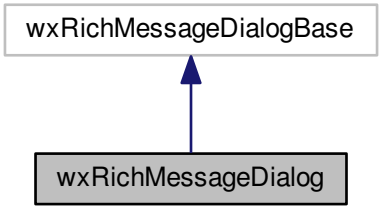
Since

2.9.4

21.606 wxRichMessageDialog Class Reference

`#include <wx/richmsgdlg.h>`

Inheritance diagram for wxRichMessageDialog:



21.606.1 Detailed Description

Extension of wxMessageDialog with additional functionality.

This class adds the possibility of using a checkbox (that is especially useful for implementing the "Don't ask me again" kind of dialogs) and an extra explanatory text which is initially collapsed and not shown to the user but can be expanded to show more information.

Notice that currently the native dialog is used only under MSW when using Vista or later Windows version. Elsewhere, or for older versions of Windows, a generic implementation which is less familiar to the users is used. Because of this it's recommended to use this class only if you do need its extra functionality and use wxMessageDialog which does have native implementation under all platforms otherwise. However if you do need to put e.g. a checkbox in a dialog, you should definitely consider using this class instead of using your own custom dialog because it will have much better appearance at least under recent Windows versions.

To use this class, you need to create the dialog object and call ShowCheckBox() and/or ShowDetailedText() to configure its contents. Other than that, it is used in exactly the same way as wxMessageDialog and supports all the styles supported by it. In particular, ShowModal() return value is the same as for wxMessageDialog. The only difference is that you need to use IsCheckBoxChecked() to examine the checkbox value if you had called ShowCheckBox().

Here is a simple example:

```
void MyFrame::ShowDialog()
{
    if ( ... shouldn't show this dialog again ... )
        return;

    wxRichMessageDialog dlg(this, "Welcome to my wonderful program!");
    dlg.ShowCheckBox("Don't show welcome dialog again");
    dlg.ShowModal(); // return value ignored as we have "Ok" only anyhow

    if ( dlg.IsCheckBoxChecked() )
        ... make sure we won't show it again the next time ...
}
```

Since

2.9.2

Library: wxCore

Category: Common Dialogs

See also

wxMessageDialog Overview

Public Member Functions

- wxRichMessageDialog (wxWindow *parent, const wxString &message, const wxString &caption=wxMessageBoxCaptionStr, long style=wxOK|wxCENTRE)

  Constructor specifying the rich message dialog properties.
- void ShowCheckBox (const wxString &checkBoxText, bool checked=false)

  Shows a checkbox with a given label or hides it.
- wxString GetCheckBoxText () const

  Retrieves the label for the checkbox.
- void ShowDetailedText (const wxString &detailedText)

  Shows or hides a detailed text and an expander that is used to show or hide the detailed text.
- wxString GetDetailedText () const

  Retrieves the detailed text.
- bool IsCheckBoxChecked () const

  Retrieves the state of the checkbox.
- virtual int ShowModal ()

  Shows the dialog, returning one of wxID_OK, wxID_CANCEL, wxID_YES, wxID_NO.

21.606.2 Constructor & Destructor Documentation

wxRichMessageDialog::wxRichMessageDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption* = wxMessageBoxCaptionStr, long *style* = wxOK|wxCENTRE)

Constructor specifying the rich message dialog properties.

Works just like the constructor for wxMessageDialog.

21.606.3 Member Function Documentation

wxString wxRichMessageDialog::GetCheckBoxText ( ) const

Retrieves the label for the checkbox.

Returns

The label for the checkbox, will be the empty string if no checkbox is used.

wxString wxRichMessageDialog::GetDetailedText ( ) const

Retrieves the detailed text.

Returns

The detailed text or empty if detailed text is not used.

bool wxRichMessageDialog::IsCheckBoxChecked ( ) const

Retrieves the state of the checkbox.

If this method is called before showing the dialog, the initial value of the checkbox, as set by ShowCheckBox() is used. If it is called after calling wxDialog::ShowModal(), the value set by the user is returned.

Returns

true if the checkbox is checked or false if not.

void wxRichMessageDialog::ShowCheckBox (const wxString & *checkBoxText,* bool *checked* = `false`)

Shows a checkbox with a given label or hides it.

Parameters

checkBoxText	If the parameter is non-empty a checkbox will be shown with that label, otherwise it will be hidden.
checked	The initial state of the checkbox.

void wxRichMessageDialog::ShowDetailedText (const wxString & *detailedText*)

Shows or hides a detailed text and an expander that is used to show or hide the detailed text.

Parameters

detailedText	The detailed text that can be expanded when the dialog is shown, if empty no detailed text will be used.

virtual int wxRichMessageDialog::ShowModal ( ) [virtual]

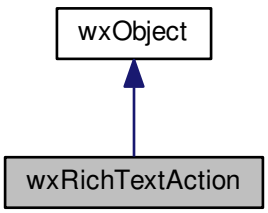
Shows the dialog, returning one of wxID_OK, wxID_CANCEL, wxID_YES, wxID_NO.

IsCheckBoxChecked() can be called afterwards to retrieve the value of the check box if one was used.

21.607 wxRichTextAction Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextAction:



21.607.1 Detailed Description

Implements a part of a command.

Library: wxRichText

Category: Rich Text

See also

wxRichTextCommand

Public Member Functions

- wxRichTextAction (wxRichTextCommand *cmd, const wxString &name, wxRichTextCommandId id, wxRichTextBuffer *buffer, wxRichTextParagraphLayoutBox *container, wxRichTextCtrl *ctrl, bool ignoreFirstTime=false)

  Constructor.
- virtual ~wxRichTextAction ()
- bool Do ()

  Performs the action.
- bool Undo ()

  Undoes the action.
- void UpdateAppearance (long caretPosition, bool sendUpdateEvent=false, const wxRect &oldFloatRect=wxRect(), wxArrayInt *optimizationLineCharPositions=NULL, wxArrayInt *optimizationLineYPositions=NULL, bool isDoCmd=true)

  Updates the control appearance, optimizing if possible given information from the call to Layout.
- void ApplyParagraphs (const wxRichTextParagraphLayoutBox &fragment)

  Replaces the buffer paragraphs with the given fragment.
- wxRichTextParagraphLayoutBox & GetNewParagraphs ()

  Returns the new fragments.
- wxRichTextParagraphLayoutBox & GetOldParagraphs ()

  Returns the old fragments.
- wxRichTextAttr & GetAttributes ()

  Returns the attributes, for single-object commands.
- wxRichTextObject * GetObject () const

  Returns the object to replace the one at the position defined by the container address and the action's range start position.
- void StoreObject (wxRichTextObject *obj)

  Stores the object to replace the one at the position defined by the container address without making an address for it.
- void SetObject (wxRichTextObject *obj)

  Sets the object to replace the one at the position defined by the container address and the action's range start position.
- void MakeObject (wxRichTextObject *obj)

  Makes an address from the given object.
- void SetOldAndNewObjects (wxRichTextObject *oldObj, wxRichTextObject *newObj)

  Sets the existing and new objects, for use with wxRICHTEXT_CHANGE_OBJECT.
- void CalculateRefreshOptimizations (wxArrayInt &optimizationLineCharPositions, wxArrayInt &optimizationLineYPositions, wxRect &oldFloatRect)

  Calculate arrays for refresh optimization.
- void SetPosition (long pos)

  Sets the position used for e.g.

- long GetPosition () const

  Returns the position used for e.g.
- void SetRange (const wxRichTextRange &range)

  Sets the range for e.g.
- const wxRichTextRange & GetRange () const

  Returns the range for e.g.
- wxRichTextObjectAddress & GetContainerAddress ()

  Returns the address (nested position) of the container within the buffer being manipulated.
- const wxRichTextObjectAddress & GetContainerAddress () const

  Returns the address (nested position) of the container within the buffer being manipulated.
- void SetContainerAddress (const wxRichTextObjectAddress &address)

  Sets the address (nested position) of the container within the buffer being manipulated.
- void SetContainerAddress (wxRichTextParagraphLayoutBox *container, wxRichTextObject *obj)

  Sets the address (nested position) of the container within the buffer being manipulated.
- wxRichTextParagraphLayoutBox * GetContainer () const

  Returns the container that this action refers to, using the container address and top-level buffer.
- const wxString & GetName () const

  Returns the action name.
- void SetIgnoreFirstTime (bool b)

  Instructs the first Do() command should be skipped as it's already been applied.
- bool GetIgnoreFirstTime () const

  Returns true if the first Do() command should be skipped as it's already been applied.

Protected Attributes

- wxString m_name
- wxRichTextBuffer * m_buffer
- wxRichTextObjectAddress m_containerAddress
- wxRichTextCtrl * m_ctrl
- wxRichTextParagraphLayoutBox m_newParagraphs
- wxRichTextParagraphLayoutBox m_oldParagraphs
- wxRichTextObject * m_object
- wxRichTextAttr m_attributes
- wxRichTextObjectAddress m_objectAddress
- wxRichTextRange m_range
- long m_position
- bool m_ignoreThis
- wxRichTextCommandId m_cmdId

Additional Inherited Members

21.607.2 Constructor & Destructor Documentation

wxRichTextAction::wxRichTextAction (wxRichTextCommand * *cmd*, const wxString & *name*, wxRichTextCommandId *id*, wxRichTextBuffer * *buffer*, wxRichTextParagraphLayoutBox * *container*, wxRichTextCtrl * *ctrl*, bool *ignoreFirstTime* = `false`)

Constructor.

buffer is the top-level buffer, while *container* is the object within which the action is taking place. In the simplest case, they are the same.

virtual wxRichTextAction::~wxRichTextAction ( ) [virtual]

21.607.3 Member Function Documentation

void wxRichTextAction::ApplyParagraphs (const wxRichTextParagraphLayoutBox & *fragment*)

Replaces the buffer paragraphs with the given fragment.

void wxRichTextAction::CalculateRefreshOptimizations (wxArrayInt & *optimizationLineCharPositions,* wxArrayInt & *optimizationLineYPositions,* wxRect & *oldFloatRect*)

Calculate arrays for refresh optimization.

bool wxRichTextAction::Do ( )

Performs the action.

wxRichTextAttr& wxRichTextAction::GetAttributes ( ) [inline]

Returns the attributes, for single-object commands.

wxRichTextParagraphLayoutBox* wxRichTextAction::GetContainer ( ) const

Returns the container that this action refers to, using the container address and top-level buffer.

wxRichTextObjectAddress& wxRichTextAction::GetContainerAddress ( ) [inline]

Returns the address (nested position) of the container within the buffer being manipulated.

const wxRichTextObjectAddress& wxRichTextAction::GetContainerAddress ( ) const [inline]

Returns the address (nested position) of the container within the buffer being manipulated.

bool wxRichTextAction::GetIgnoreFirstTime ( ) const

Returns true if the first Do() command should be skipped as it's already been applied.

const wxString& wxRichTextAction::GetName ( ) const [inline]

Returns the action name.

wxRichTextParagraphLayoutBox& wxRichTextAction::GetNewParagraphs ( ) [inline]

Returns the new fragments.

wxRichTextObject* wxRichTextAction::GetObject ( ) const [inline]

Returns the object to replace the one at the position defined by the container address and the action's range start position.

wxRichTextParagraphLayoutBox& wxRichTextAction::GetOldParagraphs ( ) [inline]

Returns the old fragments.

long wxRichTextAction::GetPosition ( ) const [inline]

Returns the position used for e.g.

insertion.

const wxRichTextRange& wxRichTextAction::GetRange ( ) const [inline]

Returns the range for e.g.

deletion.

void wxRichTextAction::MakeObject (wxRichTextObject * *obj*) [inline]

Makes an address from the given object.

void wxRichTextAction::SetContainerAddress (const wxRichTextObjectAddress & *address*) [inline]

Sets the address (nested position) of the container within the buffer being manipulated.

void wxRichTextAction::SetContainerAddress (wxRichTextParagraphLayoutBox * *container,* wxRichTextObject * *obj*) [inline]

Sets the address (nested position) of the container within the buffer being manipulated.

void wxRichTextAction::SetIgnoreFirstTime (bool *b*)

Instructs the first Do() command should be skipped as it's already been applied.

void wxRichTextAction::SetObject (wxRichTextObject * *obj*) [inline]

Sets the object to replace the one at the position defined by the container address and the action's range start position.

void wxRichTextAction::SetOldAndNewObjects (wxRichTextObject * *oldObj,* wxRichTextObject * *newObj*)

Sets the existing and new objects, for use with wxRICHTEXT_CHANGE_OBJECT.

void wxRichTextAction::SetPosition (long *pos*) [inline]

Sets the position used for e.g.

insertion.

void wxRichTextAction::SetRange (const wxRichTextRange & *range*) [inline]

Sets the range for e.g.

deletion.

void wxRichTextAction::StoreObject (wxRichTextObject * *obj*) [inline]

Stores the object to replace the one at the position defined by the container address without making an address for it.

See also

SetObject(), MakeObject()).

bool wxRichTextAction::Undo ( )

Undoes the action.

void wxRichTextAction::UpdateAppearance (long *caretPosition*, bool *sendUpdateEvent* = false, const wxRect & *oldFloatRect* = wxRect(), wxArrayInt * *optimizationLineCharPositions* = NULL, wxArrayInt * *optimizationLineYPositions* = NULL, bool *isDoCmd* = true)

Updates the control appearance, optimizing if possible given information from the call to Layout.

21.607.4 Member Data Documentation

wxRichTextAttr wxRichTextAction::m_attributes [protected]

wxRichTextBuffer* wxRichTextAction::m_buffer [protected]

wxRichTextCommandId wxRichTextAction::m_cmdId [protected]

wxRichTextObjectAddress wxRichTextAction::m_containerAddress [protected]

wxRichTextCtrl* wxRichTextAction::m_ctrl [protected]

bool wxRichTextAction::m_ignoreThis [protected]

wxString wxRichTextAction::m_name [protected]

wxRichTextParagraphLayoutBox wxRichTextAction::m_newParagraphs [protected]

wxRichTextObject* wxRichTextAction::m_object [protected]

wxRichTextObjectAddress wxRichTextAction::m_objectAddress [protected]

wxRichTextParagraphLayoutBox wxRichTextAction::m_oldParagraphs [protected]

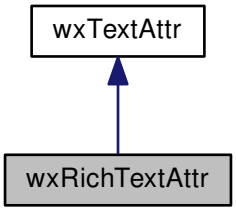
long wxRichTextAction::m_position [protected]

wxRichTextRange wxRichTextAction::m_range [protected]

21.608 wxRichTextAttr Class Reference

#include <wx/richtext/richtextbuffer.h>

Inheritance diagram for wxRichTextAttr:



21.608.1 Detailed Description

A class representing enhanced attributes for rich text objects.

This adds a wxTextBoxAttr member to the basic wxTextAttr class.

Library: wxRichText

Category: Rich Text

See also

wxTextAttr, wxTextBoxAttr, wxRichTextCtrl

Public Member Functions

- wxRichTextAttr (const wxTextAttr &attr)
  Constructor taking a wxTextAttr.
- wxRichTextAttr (const wxRichTextAttr &attr)
  Copy constructor.
- wxRichTextAttr ()
  Default constructor.
- void Copy (const wxRichTextAttr &attr)
  Copy function.
- void operator= (const wxRichTextAttr &attr)
  Assignment operator.
- void operator= (const wxTextAttr &attr)
  Assignment operator.
- bool operator== (const wxRichTextAttr &attr) const
  Equality test.
- bool EqPartial (const wxRichTextAttr &attr, bool weakTest=true) const
  Partial equality test.
- bool Apply (const wxRichTextAttr &style, const wxRichTextAttr *compareWith=NULL)
  Merges the given attributes.
- void CollectCommonAttributes (const wxRichTextAttr &attr, wxRichTextAttr &clashingAttr, wxRichTextAttr &absentAttr)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

- bool RemoveStyle (const wxRichTextAttr &attr)

  Removes the specified attributes from this object.
- wxTextBoxAttr & GetTextBoxAttr ()

  Returns the text box attributes.
- const wxTextBoxAttr & GetTextBoxAttr () const
- void SetTextBoxAttr (const wxTextBoxAttr &attr)

  Set the text box attributes.
- bool IsDefault () const

  Returns true if no attributes are set.

Public Attributes

- wxTextBoxAttr m_textBoxAttr

Additional Inherited Members

21.608.2 Constructor & Destructor Documentation

wxRichTextAttr::wxRichTextAttr (const wxTextAttr & *attr*) `[inline]`

Constructor taking a wxTextAttr.

wxRichTextAttr::wxRichTextAttr (const wxRichTextAttr & *attr*) `[inline]`

Copy constructor.

wxRichTextAttr::wxRichTextAttr ( ) `[inline]`

Default constructor.

21.608.3 Member Function Documentation

bool wxRichTextAttr::Apply (const wxRichTextAttr & *style,* const wxRichTextAttr * *compareWith* = `NULL`)

Merges the given attributes.

If *compareWith* is non-NULL, then it will be used to mask out those attributes that are the same in style and *compareWith*, for situations where we don't want to explicitly set inherited attributes.

void wxRichTextAttr::CollectCommonAttributes (const wxRichTextAttr & *attr,* wxRichTextAttr & *clashingAttr,* wxRichTextAttr & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

void wxRichTextAttr::Copy (const wxRichTextAttr & *attr*)

Copy function.

bool wxRichTextAttr::EqPartial (const wxRichTextAttr & *attr,* bool *weakTest* = `true`) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *attr* are present. If *weakTest* is false, the function will fail if an attribute is present in *attr* but not in this object.

wxTextBoxAttr& wxRichTextAttr::GetTextBoxAttr ( ) `[inline]`

Returns the text box attributes.

const wxTextBoxAttr& wxRichTextAttr::GetTextBoxAttr ( ) const `[inline]`

bool wxRichTextAttr::IsDefault ( ) const `[inline]`

Returns true if no attributes are set.

void wxRichTextAttr::operator= (const wxRichTextAttr & *attr*) `[inline]`

Assignment operator.

void wxRichTextAttr::operator= (const wxTextAttr & *attr*) `[inline]`

Assignment operator.

bool wxRichTextAttr::operator== (const wxRichTextAttr & *attr*) const

Equality test.

bool wxRichTextAttr::RemoveStyle (const wxRichTextAttr & *attr*)

Removes the specified attributes from this object.

void wxRichTextAttr::SetTextBoxAttr (const wxTextBoxAttr & *attr*) `[inline]`

Set the text box attributes.

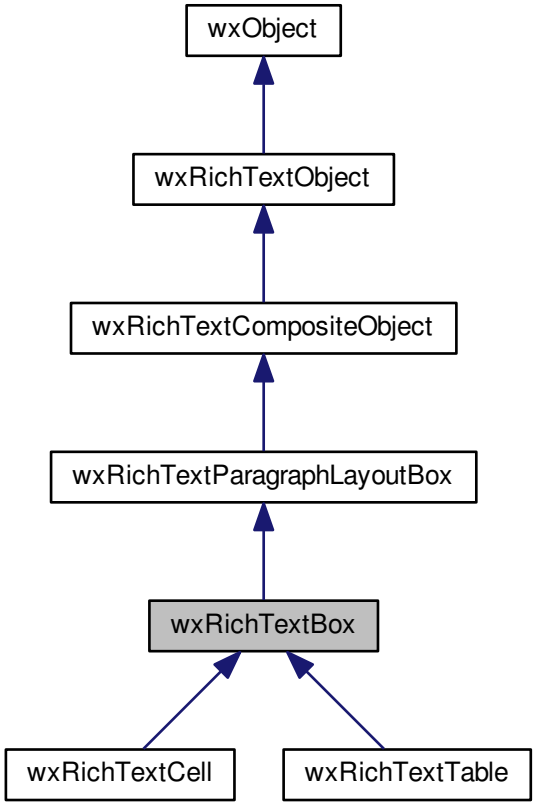
21.608.4 Member Data Documentation

wxTextBoxAttr wxRichTextAttr::m_textBoxAttr

21.609 wxRichTextBox Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextBox:



21.609.1 Detailed Description

This class implements a floating or inline text box, containing paragraphs.

Library: wxRichText

Category: Rich Text

See also

wxRichTextParagraphLayoutBox, wxRichTextObject, wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextBox (wxRichTextObject *parent=NULL)

  Default constructor; optionally pass the parent object.
- wxRichTextBox (const wxRichTextBox &obj)

  Copy constructor.

- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- virtual bool CanEditProperties () const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxWindow *parent, wxRichTextBuffer *buffer)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel () const

  Returns the label to be used for the properties context menu item.
- virtual wxRichTextObject * Clone () const

  Clones the object.
- void Copy (const wxRichTextBox &obj)

Additional Inherited Members

21.609.2 Constructor & Destructor Documentation

wxRichTextBox::wxRichTextBox (wxRichTextObject * *parent* = NULL)

Default constructor; optionally pass the parent object.

wxRichTextBox::wxRichTextBox (const wxRichTextBox & *obj*) [inline]

Copy constructor.

21.609.3 Member Function Documentation

virtual bool wxRichTextBox::CanEditProperties ( ) const [inline], [virtual]

Returns true if we can edit the object's properties via a GUI.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextCell.

virtual wxRichTextObject* wxRichTextBox::Clone ( ) const [inline], [virtual]

Clones the object.

Reimplemented from wxRichTextParagraphLayoutBox.

Reimplemented in wxRichTextTable, and wxRichTextCell.

void wxRichTextBox::Copy (const wxRichTextBox & *obj*)

virtual bool wxRichTextBox::Draw (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRichTextRange & *range*, const wxRichTextSelection & *selection*, const wxRect & *rect*, int *descent*, int *style*) [virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Reimplemented from wxRichTextParagraphLayoutBox.

Reimplemented in wxRichTextTable, and wxRichTextCell.

virtual bool wxRichTextBox::EditProperties (wxWindow * *parent,* wxRichTextBuffer * *buffer*) [virtual]

Edits the object's properties via a GUI.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextCell.

virtual wxString wxRichTextBox::GetPropertiesMenuLabel ( ) const [inline], [virtual]

Returns the label to be used for the properties context menu item.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextCell.

virtual wxString wxRichTextBox::GetXMLNodeName ( ) const [inline], [virtual]

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

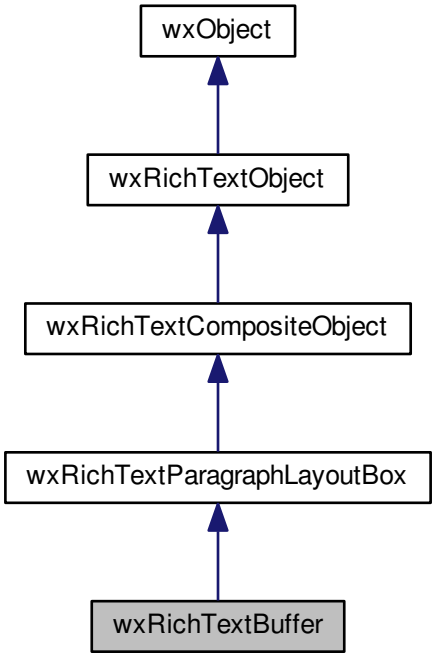
Reimplemented from wxRichTextParagraphLayoutBox.

Reimplemented in wxRichTextTable, and wxRichTextCell.

21.610 wxRichTextBuffer Class Reference

```
#include <wx/richtext/richtextbuffer.h>
```

Inheritance diagram for wxRichTextBuffer:



21.610.1 Detailed Description

This is a kind of paragraph layout box, used to represent the whole buffer.

Library: wxRichText

Category: Rich Text

See also

wxRichTextParagraphLayoutBox, wxRichTextCtrl

Public Member Functions

- wxRichTextBuffer ()

  Default constructor.
- wxRichTextBuffer (const wxRichTextBuffer &obj)

  Copy constructor.
- virtual ~wxRichTextBuffer ()
- wxCommandProcessor * GetCommandProcessor () const

  Returns the command processor.
- void SetStyleSheet (wxRichTextStyleSheet *styleSheet)

  Sets style sheet, if any.

- virtual wxRichTextStyleSheet * GetStyleSheet () const

  Returns the style sheet.
- bool SetStyleSheetAndNotify (wxRichTextStyleSheet *sheet)

  Sets the style sheet and sends a notification of the change.
- bool PushStyleSheet (wxRichTextStyleSheet *styleSheet)

  Pushes the style sheet to the top of the style sheet stack.
- wxRichTextStyleSheet * PopStyleSheet ()

  Pops the style sheet from the top of the style sheet stack.
- wxRichTextFontTable & GetFontTable ()

  Returns the table storing fonts, for quick access and font reuse.
- const wxRichTextFontTable & GetFontTable () const

  Returns the table storing fonts, for quick access and font reuse.
- void SetFontTable (const wxRichTextFontTable &table)

  Sets table storing fonts, for quick access and font reuse.
- void SetFontScale (double fontScale)

  Sets the scale factor for displaying fonts, for example for more comfortable editing.
- double GetFontScale () const

  Returns the scale factor for displaying fonts, for example for more comfortable editing.
- void SetDimensionScale (double dimScale)

  Sets the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.
- double GetDimensionScale () const

  Returns the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.
- void Init ()

  Initialisation.
- virtual void ResetAndClearCommands ()

  Clears the buffer, adds an empty paragraph, and clears the command processor.
- void SetHandlerFlags (int flags)

  Sets the handler flags, controlling loading and saving.
- int GetHandlerFlags () const

  Gets the handler flags, controlling loading and saving.
- virtual wxRichTextRange AddParagraph (const wxString &text, wxRichTextAttr *paraStyle=NULL)

  Convenience function to add a paragraph of text.
- virtual bool BeginBatchUndo (const wxString &cmdName)

  Begin collapsing undo/redo commands.
- virtual bool EndBatchUndo ()

  End collapsing undo/redo commands.
- virtual bool BatchingUndo () const

  Returns true if we are collapsing commands.
- virtual bool SubmitAction (wxRichTextAction *action)

  Submit the action immediately, or delay according to whether collapsing is on.
- virtual wxRichTextCommand * GetBatchedCommand () const

  Returns the collapsed command.
- virtual bool BeginSuppressUndo ()

  Begin suppressing undo/redo commands.
- virtual bool EndSuppressUndo ()

  End suppressing undo/redo commands.
- virtual bool SuppressingUndo () const

  Are we suppressing undo??
- virtual bool CopyToClipboard (const wxRichTextRange &range)

  Copy the range to the clipboard.
- virtual bool PasteFromClipboard (long position)

Paste the clipboard content to the buffer.

- virtual bool CanPasteFromClipboard () const

  Returns true if we can paste from the clipboard.

- virtual bool BeginStyle (const wxRichTextAttr &style)

  Begin using a style.

- virtual bool EndStyle ()

  End the style.

- virtual bool EndAllStyles ()

  End all styles.

- virtual void ClearStyleStack ()

  Clears the style stack.

- virtual size_t GetStyleStackSize () const

  Returns the size of the style stack, for example to check correct nesting.

- bool BeginBold ()

  Begins using bold.

- bool EndBold ()

  Ends using bold.

- bool BeginItalic ()

  Begins using italic.

- bool EndItalic ()

  Ends using italic.

- bool BeginUnderline ()

  Begins using underline.

- bool EndUnderline ()

  Ends using underline.

- bool BeginFontSize (int pointSize)

  Begins using point size.

- bool EndFontSize ()

  Ends using point size.

- bool BeginFont (const wxFont &font)

  Begins using this font.

- bool EndFont ()

  Ends using a font.

- bool BeginTextColour (const wxColour &colour)

  Begins using this colour.

- bool EndTextColour ()

  Ends using a colour.

- bool BeginAlignment (wxTextAttrAlignment alignment)

  Begins using alignment.

- bool EndAlignment ()

  Ends alignment.

- bool BeginLeftIndent (int leftIndent, int leftSubIndent=0)

  Begins using leftIndent for the left indent, and optionally leftSubIndent for the sub-indent.

- bool EndLeftIndent ()

  Ends left indent.

- bool BeginRightIndent (int rightIndent)

  Begins a right indent, specified in tenths of a millimetre.

- bool EndRightIndent ()

  Ends right indent.

- bool BeginParagraphSpacing (int before, int after)

  Begins paragraph spacing; pass the before-paragraph and after-paragraph spacing in tenths of a millimetre.

- bool EndParagraphSpacing ()

  Ends paragraph spacing.
- bool BeginLineSpacing (int lineSpacing)

  Begins line spacing using the specified value.
- bool EndLineSpacing ()

  Ends line spacing.
- bool BeginNumberedBullet (int bulletNumber, int leftIndent, int leftSubIndent, int bulletStyle=wxTEXT_ATTR_BULLET_STYLE_ARABIC|wxTEXT_ATTR_BULLET_STYLE_PERIOD)

  Begins numbered bullet.
- bool EndNumberedBullet ()

  Ends numbered bullet.
- bool BeginSymbolBullet (const wxString &symbol, int leftIndent, int leftSubIndent, int bulletStyle=wxTEXT_ATTR_BULLET_STYLE_SYMBOL)

  Begins applying a symbol bullet, using a character from the current font.
- bool EndSymbolBullet ()

  Ends symbol bullet.
- bool BeginStandardBullet (const wxString &bulletName, int leftIndent, int leftSubIndent, int bulletStyle=wxTEXT_ATTR_BULLET_STYLE_STANDARD)

  Begins applying a standard bullet, using one of the standard bullet names (currently `standard/circle` *or* `standard/square`*.*
- bool EndStandardBullet ()

  Ends standard bullet.
- bool BeginCharacterStyle (const wxString &characterStyle)

  Begins named character style.
- bool EndCharacterStyle ()

  Ends named character style.
- bool BeginParagraphStyle (const wxString &paragraphStyle)

  Begins named paragraph style.
- bool EndParagraphStyle ()

  Ends named character style.
- bool BeginListStyle (const wxString &listStyle, int level=1, int number=1)

  Begins named list style.
- bool EndListStyle ()

  Ends named character style.
- bool BeginURL (const wxString &url, const wxString &characterStyle=wxEmptyString)

  Begins applying wxTEXT_ATTR_URL to the content.
- bool EndURL ()

  Ends URL.
- bool AddEventHandler (wxEvtHandler *handler)

  Adds an event handler.
- bool RemoveEventHandler (wxEvtHandler *handler, bool deleteHandler=false)

  Removes an event handler from the buffer's list of handlers, deleting the object if deleteHandler is true.
- void ClearEventHandlers ()

  Clear event handlers.
- bool SendEvent (wxEvent &event, bool sendToAll=true)

  Send event to event handlers.
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

  Hit-testing: returns a flag indicating hit test details, plus information about position.
- void Copy (const wxRichTextBuffer &obj)

  Copies the buffer.

- void operator= (const wxRichTextBuffer &obj)

  Assignment operator.
- virtual wxRichTextObject * Clone () const

  Clones the buffer.
- bool InsertParagraphsWithUndo (long pos, const wxRichTextParagraphLayoutBox &paragraphs, wxRich↩ TextCtrl *ctrl, int flags=0)

  Submits a command to insert paragraphs.
- bool InsertTextWithUndo (long pos, const wxString &text, wxRichTextCtrl *ctrl, int flags=0)

  Submits a command to insert the given text.
- bool InsertNewlineWithUndo (long pos, wxRichTextCtrl *ctrl, int flags=0)

  Submits a command to insert a newline.
- bool InsertImageWithUndo (long pos, const wxRichTextImageBlock &imageBlock, wxRichTextCtrl *ctrl, int flags=0, const wxRichTextAttr &textAttr=wxRichTextAttr())

  Submits a command to insert the given image.
- wxRichTextObject * InsertObjectWithUndo (long pos, wxRichTextObject *object, wxRichTextCtrl *ctrl, int flags)

  Submits a command to insert an object.
- bool DeleteRangeWithUndo (const wxRichTextRange &range, wxRichTextCtrl *ctrl)

  Submits a command to delete this range.
- void Modify (bool modify=true)

  Mark modified.
- bool IsModified () const

  Returns true if the buffer was modified.
- double GetScale () const

  Returns the scale factor for calculating dimensions.
- void SetScale (double scale)

  Sets the scale factor for calculating dimensions.

- virtual bool LoadFile (const wxString &filename, wxRichTextFileType type=wxRICHTEXT_TYPE_ANY)

  Loads content from a stream or file.
- virtual bool LoadFile (wxInputStream &stream, wxRichTextFileType type=wxRICHTEXT_TYPE_ANY)

  Loads content from a stream or file.

- virtual bool SaveFile (const wxString &filename, wxRichTextFileType type=wxRICHTEXT_TYPE_ANY)

  Saves content to a stream or file.
- virtual bool SaveFile (wxOutputStream &stream, wxRichTextFileType type=wxRICHTEXT_TYPE_ANY)

  Saves content to a stream or file.

- virtual void Dump ()

  Dumps contents of buffer for debugging purposes.
- virtual void Dump (wxTextOutputStream &stream)

  Dumps contents of buffer for debugging purposes.

Static Public Member Functions

- static wxList & GetHandlers ()

  Returns the file handlers.
- static void AddHandler (wxRichTextFileHandler *handler)

  Adds a file handler to the end.
- static void InsertHandler (wxRichTextFileHandler *handler)

  Inserts a file handler at the front.

- static bool RemoveHandler (const wxString &name)

  Removes a file handler.
- static wxRichTextFileHandler * FindHandler (const wxString &name)

  Finds a file handler by name.
- static wxRichTextFileHandler * FindHandler (const wxString &extension, wxRichTextFileType imageType)

  Finds a file handler by extension and type.
- static wxRichTextFileHandler * FindHandlerFilenameOrType (const wxString &filename, wxRichTextFileType imageType)

  Finds a handler by filename or, if supplied, type.
- static wxRichTextFileHandler * FindHandler (wxRichTextFileType imageType)

  Finds a handler by type.
- static wxString GetExtWildcard (bool combine=false, bool save=false, wxArrayInt *types=NULL)

  Gets a wildcard incorporating all visible handlers.
- static void CleanUpHandlers ()

  Clean up file handlers.
- static void InitStandardHandlers ()

  Initialise the standard file handlers.
- static wxList & GetDrawingHandlers ()

  Returns the drawing handlers.
- static void AddDrawingHandler (wxRichTextDrawingHandler *handler)

  Adds a drawing handler to the end.
- static void InsertDrawingHandler (wxRichTextDrawingHandler *handler)

  Inserts a drawing handler at the front.
- static bool RemoveDrawingHandler (const wxString &name)

  Removes a drawing handler.
- static wxRichTextDrawingHandler * FindDrawingHandler (const wxString &name)

  Finds a drawing handler by name.
- static void CleanUpDrawingHandlers ()

  Clean up drawing handlers.
- static wxRichTextFieldTypeHashMap & GetFieldTypes ()

  Returns the field types.
- static void AddFieldType (wxRichTextFieldType *fieldType)

  Adds a field type.
- static bool RemoveFieldType (const wxString &name)

  Removes a field type by name.
- static wxRichTextFieldType * FindFieldType (const wxString &name)

  Finds a field type by name.
- static void CleanUpFieldTypes ()

  Cleans up field types.
- static wxRichTextRenderer * GetRenderer ()

  Returns the renderer object.
- static void SetRenderer (wxRichTextRenderer *renderer)

  Sets renderer as the object to be used to render certain aspects of the content, such as bullets.
- static int GetBulletRightMargin ()

  Returns the minimum margin between bullet and paragraph in 10ths of a mm.
- static void SetBulletRightMargin (int margin)

  Sets the minimum margin between bullet and paragraph in 10ths of a mm.
- static float GetBulletProportion ()

  Returns the factor to multiply by character height to get a reasonable bullet size.
- static void SetBulletProportion (float prop)

  Sets the factor to multiply by character height to get a reasonable bullet size.

- static bool GetFloatingLayoutMode ()

  Returns the floating layout mode.
- static void SetFloatingLayoutMode (bool mode)

  Sets the floating layout mode.

Protected Attributes

- wxCommandProcessor * m_commandProcessor

  Command processor.
- wxRichTextFontTable m_fontTable

  Table storing fonts.
- bool m_modified

  Has been modified?
- int m_batchedCommandDepth

  Collapsed command stack.
- wxString m_batchedCommandsName

  Name for collapsed command.
- wxRichTextCommand * m_batchedCommand

  Current collapsed command accumulating actions.
- int m_suppressUndo

  Whether to suppress undo.
- wxRichTextStyleSheet * m_styleSheet

  Style sheet, if any.
- wxList m_eventHandlers

  List of event handlers that will be notified of events.
- wxList m_attributeStack

  Stack of attributes for convenience functions.
- int m_handlerFlags

  Flags to be passed to handlers.
- double m_scale

  Scaling factor in use: needed to calculate correct dimensions when printing.

Static Protected Attributes

- static wxList sm_handlers

  File handlers.
- static wxList sm_drawingHandlers

  Drawing handlers.
- static wxRichTextFieldTypeHashMap sm_fieldTypes

  Field types.
- static wxRichTextRenderer * sm_renderer

  Renderer.
- static int sm_bulletRightMargin

  Minimum margin between bullet and paragraph in 10ths of a mm.
- static float sm_bulletProportion

  Factor to multiply by character height to get a reasonable bullet size.

Additional Inherited Members

21.610.2 Constructor & Destructor Documentation

wxRichTextBuffer::wxRichTextBuffer ( ) [inline]

Default constructor.

wxRichTextBuffer::wxRichTextBuffer (const wxRichTextBuffer & *obj*) [inline]

Copy constructor.

virtual wxRichTextBuffer::~wxRichTextBuffer ( ) [virtual]

21.610.3 Member Function Documentation

static void wxRichTextBuffer::AddDrawingHandler (wxRichTextDrawingHandler * *handler*) [static]

Adds a drawing handler to the end.

bool wxRichTextBuffer::AddEventHandler (wxEvtHandler * *handler*)

Adds an event handler.

A buffer associated with a control has the control as the only event handler, but the application is free to add more if further notification is required. All handlers are notified of an event originating from the buffer, such as the replacement of a style sheet during loading.

The buffer never deletes any of the event handlers, unless RemoveEventHandler() is called with true as the second argument.

static void wxRichTextBuffer::AddFieldType (wxRichTextFieldType * *fieldType*) [static]

Adds a field type.

See also

RemoveFieldType(), FindFieldType(), wxRichTextField, wxRichTextFieldType, wxRichTextFieldTypeStandard

static void wxRichTextBuffer::AddHandler (wxRichTextFileHandler * *handler*) [static]

Adds a file handler to the end.

virtual wxRichTextRange wxRichTextBuffer::AddParagraph (const wxString & *text*, wxRichTextAttr * *paraStyle* = NULL) [inline], [virtual]

Convenience function to add a paragraph of text.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextBuffer::BatchingUndo ( ) const [inline], [virtual]

Returns true if we are collapsing commands.

bool wxRichTextBuffer::BeginAlignment (wxTextAttrAlignment *alignment*)

Begins using alignment.

virtual bool wxRichTextBuffer::BeginBatchUndo (const wxString & *cmdName*) [virtual]

Begin collapsing undo/redo commands.

Note that this may not work properly if combining commands that delete or insert content, changing ranges for subsequent actions.

cmdName should be the name of the combined command that will appear next to Undo and Redo in the edit menu.

bool wxRichTextBuffer::BeginBold ( )

Begins using bold.

bool wxRichTextBuffer::BeginCharacterStyle (const wxString & *characterStyle*)

Begins named character style.

bool wxRichTextBuffer::BeginFont (const wxFont & *font*)

Begins using this font.

bool wxRichTextBuffer::BeginFontSize (int *pointSize*)

Begins using point size.

bool wxRichTextBuffer::BeginItalic ( )

Begins using italic.

bool wxRichTextBuffer::BeginLeftIndent (int *leftIndent,* int *leftSubIndent* = 0)

Begins using *leftIndent* for the left indent, and optionally *leftSubIndent* for the sub-indent.

Both are expressed in tenths of a millimetre.

The sub-indent is an offset from the left of the paragraph, and is used for all but the first line in a paragraph. A positive value will cause the first line to appear to the left of the subsequent lines, and a negative value will cause the first line to be indented relative to the subsequent lines.

bool wxRichTextBuffer::BeginLineSpacing (int *lineSpacing*)

Begins line spacing using the specified value.

spacing is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing.

The wxTextAttrLineSpacing enumeration values are defined for convenience.

bool wxRichTextBuffer::BeginListStyle (const wxString & *listStyle,* int *level* = 1, int *number* = 1)

Begins named list style.

Optionally, you can also pass a level and a number.

bool wxRichTextBuffer::BeginNumberedBullet (int *bulletNumber,* int *leftIndent,* int *leftSubIndent,* int *bulletStyle* = wxTEXT_ATTR_BULLET_STYLE_ARABIC|wxTEXT_ATTR_BULLET_STYLE_PERIOD)

Begins numbered bullet.

This call will be needed for each item in the list, and the application should take care of incrementing the numbering.

bulletNumber is a number, usually starting with 1. *leftIndent* and *leftSubIndent* are values in tenths of a millimetre. *bulletStyle* is a bitlist of the following values:

wxRichTextBuffer uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at leftMargin + leftSubIndent. So the distance between the left edge of the bullet and the left of the actual paragraph is leftSubIndent.

bool wxRichTextBuffer::BeginParagraphSpacing (int *before,* int *after*)

Begins paragraph spacing; pass the before-paragraph and after-paragraph spacing in tenths of a millimetre.

bool wxRichTextBuffer::BeginParagraphStyle (const wxString & *paragraphStyle*)

Begins named paragraph style.

bool wxRichTextBuffer::BeginRightIndent (int *rightIndent*)

Begins a right indent, specified in tenths of a millimetre.

bool wxRichTextBuffer::BeginStandardBullet (const wxString & *bulletName,* int *leftIndent,* int *leftSubIndent,* int *bulletStyle* = wxTEXT_ATTR_BULLET_STYLE_STANDARD)

Begins applying a standard bullet, using one of the standard bullet names (currently `standard/circle` or `standard/square`.

See BeginNumberedBullet() for an explanation of how indentation is used to render the bulleted paragraph.

virtual bool wxRichTextBuffer::BeginStyle (const wxRichTextAttr & *style*) [virtual]

Begin using a style.

virtual bool wxRichTextBuffer::BeginSuppressUndo ( ) [virtual]

Begin suppressing undo/redo commands.

The way undo is suppressed may be implemented differently by each command. If not dealt with by a command implementation, then it will be implemented automatically by not storing the command in the undo history when the action is submitted to the command processor.

bool wxRichTextBuffer::BeginSymbolBullet (const wxString & *symbol,* int *leftIndent,* int *leftSubIndent,* int *bulletStyle* = wxTEXT_ATTR_BULLET_STYLE_SYMBOL)

Begins applying a symbol bullet, using a character from the current font.

See BeginNumberedBullet() for an explanation of how indentation is used to render the bulleted paragraph.

bool wxRichTextBuffer::BeginTextColour (const wxColour & *colour*)

Begins using this colour.

bool wxRichTextBuffer::BeginUnderline ( )

Begins using underline.

bool wxRichTextBuffer::BeginURL (const wxString & *url,* const wxString & *characterStyle* = wxEmptyString)

Begins applying wxTEXT_ATTR_URL to the content.

Pass a URL and optionally, a character style to apply, since it is common to mark a URL with a familiar style such as blue text with underlining.

virtual bool wxRichTextBuffer::CanPasteFromClipboard ( ) const `[virtual]`

Returns true if we can paste from the clipboard.

static void wxRichTextBuffer::CleanUpDrawingHandlers ( ) `[static]`

Clean up drawing handlers.

static void wxRichTextBuffer::CleanUpFieldTypes ( ) `[static]`

Cleans up field types.

static void wxRichTextBuffer::CleanUpHandlers ( ) `[static]`

Clean up file handlers.

void wxRichTextBuffer::ClearEventHandlers ( )

Clear event handlers.

virtual void wxRichTextBuffer::ClearStyleStack ( ) `[virtual]`

Clears the style stack.

virtual wxRichTextObject* wxRichTextBuffer::Clone ( ) const `[inline], [virtual]`

Clones the buffer.

Reimplemented from wxRichTextParagraphLayoutBox.

void wxRichTextBuffer::Copy (const wxRichTextBuffer & *obj*)

Copies the buffer.

virtual bool wxRichTextBuffer::CopyToClipboard (const wxRichTextRange & *range*) [virtual]

Copy the range to the clipboard.

bool wxRichTextBuffer::DeleteRangeWithUndo (const wxRichTextRange & *range,* wxRichTextCtrl * *ctrl*)

Submits a command to delete this range.

virtual void wxRichTextBuffer::Dump ( ) [virtual]

Dumps contents of buffer for debugging purposes.

virtual void wxRichTextBuffer::Dump (wxTextOutputStream & *stream*) [inline], [virtual]

Dumps contents of buffer for debugging purposes.

Reimplemented from wxRichTextCompositeObject.

bool wxRichTextBuffer::EndAlignment ( ) [inline]

Ends alignment.

virtual bool wxRichTextBuffer::EndAllStyles ( ) [virtual]

End all styles.

virtual bool wxRichTextBuffer::EndBatchUndo ( ) [virtual]

End collapsing undo/redo commands.

bool wxRichTextBuffer::EndBold ( ) [inline]

Ends using bold.

bool wxRichTextBuffer::EndCharacterStyle ( ) [inline]

Ends named character style.

bool wxRichTextBuffer::EndFont ( ) [inline]

Ends using a font.

bool wxRichTextBuffer::EndFontSize ( ) [inline]

Ends using point size.

bool wxRichTextBuffer::EndItalic ( ) [inline]

Ends using italic.

bool wxRichTextBuffer::EndLeftIndent ( ) [inline]

Ends left indent.

bool wxRichTextBuffer::EndLineSpacing ( ) [inline]

Ends line spacing.

bool wxRichTextBuffer::EndListStyle ( ) [inline]

Ends named character style.

bool wxRichTextBuffer::EndNumberedBullet ( ) [inline]

Ends numbered bullet.

bool wxRichTextBuffer::EndParagraphSpacing ( ) [inline]

Ends paragraph spacing.

bool wxRichTextBuffer::EndParagraphStyle ( ) [inline]

Ends named character style.

bool wxRichTextBuffer::EndRightIndent ( ) [inline]

Ends right indent.

bool wxRichTextBuffer::EndStandardBullet ( ) [inline]

Ends standard bullet.

virtual bool wxRichTextBuffer::EndStyle ( ) [virtual]

End the style.

virtual bool wxRichTextBuffer::EndSuppressUndo ( ) [virtual]

End suppressing undo/redo commands.

bool wxRichTextBuffer::EndSymbolBullet ( ) [inline]

Ends symbol bullet.

bool wxRichTextBuffer::EndTextColour ( ) [inline]

Ends using a colour.

bool wxRichTextBuffer::EndUnderline ( ) [inline]

Ends using underline.

bool wxRichTextBuffer::EndURL ( ) [inline]

Ends URL.

static wxRichTextDrawingHandler* wxRichTextBuffer::FindDrawingHandler (const wxString & *name*) [static]

Finds a drawing handler by name.

static wxRichTextFieldType* wxRichTextBuffer::FindFieldType (const wxString & *name*) [static]

Finds a field type by name.

See also

RemoveFieldType(), AddFieldType(), wxRichTextField, wxRichTextFieldType, wxRichTextFieldTypeStandard

static wxRichTextFileHandler* wxRichTextBuffer::FindHandler (const wxString & *name*) [static]

Finds a file handler by name.

static wxRichTextFileHandler* wxRichTextBuffer::FindHandler (const wxString & *extension,* wxRichTextFileType *imageType*) [static]

Finds a file handler by extension and type.

static wxRichTextFileHandler* wxRichTextBuffer::FindHandler (wxRichTextFileType *imageType*) [static]

Finds a handler by type.

static wxRichTextFileHandler* wxRichTextBuffer::FindHandlerFilenameOrType (const wxString & *filename,* wxRichTextFileType *imageType*) [static]

Finds a handler by filename or, if supplied, type.

virtual wxRichTextCommand* wxRichTextBuffer::GetBatchedCommand ( ) const [inline], [virtual]

Returns the collapsed command.

static float wxRichTextBuffer::GetBulletProportion ( ) [inline], [static]

Returns the factor to multiply by character height to get a reasonable bullet size.

static int wxRichTextBuffer::GetBulletRightMargin ( ) [inline], [static]

Returns the minimum margin between bullet and paragraph in 10ths of a mm.

wxCommandProcessor* wxRichTextBuffer::GetCommandProcessor ( ) const [inline]

Returns the command processor.

A text buffer always creates its own command processor when it is initialized.

double wxRichTextBuffer::GetDimensionScale ( ) const [inline]

Returns the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.

static wxList& wxRichTextBuffer::GetDrawingHandlers ( ) [inline], [static]

Returns the drawing handlers.

static wxString wxRichTextBuffer::GetExtWildcard (bool *combine* = false, bool *save* = false, wxArrayInt * *types* = NULL) [static]

Gets a wildcard incorporating all visible handlers.

If *types* is present, it will be filled with the file type corresponding to each filter. This can be used to determine the type to pass to LoadFile given a selected filter.

static wxRichTextFieldTypeHashMap& wxRichTextBuffer::GetFieldTypes ( ) [inline], [static]

Returns the field types.

static bool wxRichTextBuffer::GetFloatingLayoutMode ( ) [static]

Returns the floating layout mode.

The default is true, where objects are laid out according to their floating status.

double wxRichTextBuffer::GetFontScale ( ) const [inline]

Returns the scale factor for displaying fonts, for example for more comfortable editing.

wxRichTextFontTable& wxRichTextBuffer::GetFontTable ( ) [inline]

Returns the table storing fonts, for quick access and font reuse.

const wxRichTextFontTable& wxRichTextBuffer::GetFontTable ( ) const [inline]

Returns the table storing fonts, for quick access and font reuse.

int wxRichTextBuffer::GetHandlerFlags ( ) const [inline]

Gets the handler flags, controlling loading and saving.

static wxList& wxRichTextBuffer::GetHandlers ( ) [inline], [static]

Returns the file handlers.

static wxRichTextRenderer* wxRichTextBuffer::GetRenderer ( ) [inline], [static]

Returns the renderer object.

double wxRichTextBuffer::GetScale ( ) const [inline]

Returns the scale factor for calculating dimensions.

virtual wxRichTextStyleSheet* wxRichTextBuffer::GetStyleSheet ( ) const [inline], [virtual]

Returns the style sheet.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual size_t wxRichTextBuffer::GetStyleStackSize ( ) const [inline], [virtual]

Returns the size of the style stack, for example to check correct nesting.

virtual int wxRichTextBuffer::HitTest (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxPoint & *pt,* long & *textPosition,* wxRichTextObject ** *obj,* wxRichTextObject ** *contextObj,* int *flags* = 0) [virtual]

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented from wxRichTextParagraphLayoutBox.

void wxRichTextBuffer::Init ( )

Initialisation.

static void wxRichTextBuffer::InitStandardHandlers ( ) [static]

Initialise the standard file handlers.

Currently, only the plain text loading/saving handler is initialised by default.

static void wxRichTextBuffer::InsertDrawingHandler (wxRichTextDrawingHandler * *handler*) [static]

Inserts a drawing handler at the front.

static void wxRichTextBuffer::InsertHandler (wxRichTextFileHandler * *handler*) [static]

Inserts a file handler at the front.

bool wxRichTextBuffer::InsertImageWithUndo (long *pos,* const wxRichTextImageBlock & *imageBlock,* wxRichTextCtrl * *ctrl,* int *flags* = 0, const wxRichTextAttr & *textAttr* = wxRichTextAttr())

Submits a command to insert the given image.

bool wxRichTextBuffer::InsertNewlineWithUndo (long *pos,* wxRichTextCtrl * *ctrl,* int *flags* = 0)

Submits a command to insert a newline.

wxRichTextObject* wxRichTextBuffer::InsertObjectWithUndo (long *pos,* wxRichTextObject * *object,* wxRichTextCtrl * *ctrl,* int *flags*)

Submits a command to insert an object.

bool wxRichTextBuffer::InsertParagraphsWithUndo (long *pos,* const wxRichTextParagraphLayoutBox & *paragraphs,* wxRichTextCtrl * *ctrl,* int *flags* = 0)

Submits a command to insert paragraphs.

bool wxRichTextBuffer::InsertTextWithUndo (long *pos,* const wxString & *text,* wxRichTextCtrl * *ctrl,* int *flags* = 0)

Submits a command to insert the given text.

bool wxRichTextBuffer::IsModified ( ) const [inline]

Returns true if the buffer was modified.

virtual bool wxRichTextBuffer::LoadFile (const wxString & *filename,* wxRichTextFileType *type* = wxRICHTEXT_TYPE_ANY) [virtual]

Loads content from a stream or file.

Not all handlers will implement file loading.

virtual bool wxRichTextBuffer::LoadFile (wxInputStream & *stream,* wxRichTextFileType *type* = wxRICHTEXT_TYPE_ANY) [virtual]

Loads content from a stream or file.

Not all handlers will implement file loading.

void wxRichTextBuffer::Modify (bool *modify* = true) [inline]

Mark modified.

void wxRichTextBuffer::operator= (const wxRichTextBuffer & *obj*) [inline]

Assignment operator.

virtual bool wxRichTextBuffer::PasteFromClipboard (long *position*) `[virtual]`

Paste the clipboard content to the buffer.

wxRichTextStyleSheet* wxRichTextBuffer::PopStyleSheet ( )

Pops the style sheet from the top of the style sheet stack.

bool wxRichTextBuffer::PushStyleSheet (wxRichTextStyleSheet * *styleSheet*)

Pushes the style sheet to the top of the style sheet stack.

static bool wxRichTextBuffer::RemoveDrawingHandler (const wxString & *name*) `[static]`

Removes a drawing handler.

bool wxRichTextBuffer::RemoveEventHandler (wxEvtHandler * *handler,* bool *deleteHandler* = `false`)

Removes an event handler from the buffer's list of handlers, deleting the object if *deleteHandler* is true.

static bool wxRichTextBuffer::RemoveFieldType (const wxString & *name*) `[static]`

Removes a field type by name.

See also

AddFieldType(), FindFieldType(), wxRichTextField, wxRichTextFieldType, wxRichTextFieldTypeStandard

static bool wxRichTextBuffer::RemoveHandler (const wxString & *name*) `[static]`

Removes a file handler.

virtual void wxRichTextBuffer::ResetAndClearCommands ( ) `[virtual]`

Clears the buffer, adds an empty paragraph, and clears the command processor.

virtual bool wxRichTextBuffer::SaveFile (const wxString & *filename,* wxRichTextFileType *type* = wxRICHTEXT_TYPE_ANY) `[virtual]`

Saves content to a stream or file.

Not all handlers will implement file saving.

virtual bool wxRichTextBuffer::SaveFile (wxOutputStream & *stream,* wxRichTextFileType *type* = wxRICHTEXT_TYPE_ANY) `[virtual]`

Saves content to a stream or file.

Not all handlers will implement file saving.

bool wxRichTextBuffer::SendEvent (wxEvent & *event*, bool *sendToAll* = `true`)

Send event to event handlers.

If sendToAll is true, will send to all event handlers, otherwise will stop at the first successful one.

static void wxRichTextBuffer::SetBulletProportion (float *prop*) `[inline], [static]`

Sets the factor to multiply by character height to get a reasonable bullet size.

static void wxRichTextBuffer::SetBulletRightMargin (int *margin*) `[inline], [static]`

Sets the minimum margin between bullet and paragraph in 10ths of a mm.

void wxRichTextBuffer::SetDimensionScale (double *dimScale*)

Sets the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.

This can be useful when editing in a small control where you still want legible text, but a minimum of wasted white space.

static void wxRichTextBuffer::SetFloatingLayoutMode (bool *mode*) `[static]`

Sets the floating layout mode.

Pass false to speed up editing by not performing floating layout. This setting affects all buffers.

void wxRichTextBuffer::SetFontScale (double *fontScale*)

Sets the scale factor for displaying fonts, for example for more comfortable editing.

void wxRichTextBuffer::SetFontTable (const wxRichTextFontTable & *table*) `[inline]`

Sets table storing fonts, for quick access and font reuse.

void wxRichTextBuffer::SetHandlerFlags (int *flags*) `[inline]`

Sets the handler flags, controlling loading and saving.

static void wxRichTextBuffer::SetRenderer (wxRichTextRenderer * *renderer*) `[static]`

Sets *renderer* as the object to be used to render certain aspects of the content, such as bullets.

You can override default rendering by deriving a new class from wxRichTextRenderer or wxRichTextStdRenderer, overriding one or more virtual functions, and setting an instance of the class using this function.

void wxRichTextBuffer::SetScale (double *scale*) `[inline]`

Sets the scale factor for calculating dimensions.

void wxRichTextBuffer::SetStyleSheet (wxRichTextStyleSheet * *styleSheet*) [inline]

Sets style sheet, if any.

This will allow the application to use named character and paragraph styles found in the style sheet.

Neither the buffer nor the control owns the style sheet so must be deleted by the application.

bool wxRichTextBuffer::SetStyleSheetAndNotify (wxRichTextStyleSheet * *sheet*)

Sets the style sheet and sends a notification of the change.

virtual bool wxRichTextBuffer::SubmitAction (wxRichTextAction * *action*) [virtual]

Submit the action immediately, or delay according to whether collapsing is on.

virtual bool wxRichTextBuffer::SuppressingUndo ( ) const [inline], [virtual]

Are we suppressing undo??

21.610.4 Member Data Documentation

wxList wxRichTextBuffer::m_attributeStack [protected]

Stack of attributes for convenience functions.

wxRichTextCommand* wxRichTextBuffer::m_batchedCommand [protected]

Current collapsed command accumulating actions.

int wxRichTextBuffer::m_batchedCommandDepth [protected]

Collapsed command stack.

wxString wxRichTextBuffer::m_batchedCommandsName [protected]

Name for collapsed command.

wxCommandProcessor* wxRichTextBuffer::m_commandProcessor [protected]

Command processor.

wxList wxRichTextBuffer::m_eventHandlers [protected]

List of event handlers that will be notified of events.

wxRichTextFontTable wxRichTextBuffer::m_fontTable [protected]

Table storing fonts.

int wxRichTextBuffer::m_handlerFlags [protected]

Flags to be passed to handlers.

bool wxRichTextBuffer::m_modified [protected]

Has been modified?

double wxRichTextBuffer::m_scale [protected]

Scaling factor in use: needed to calculate correct dimensions when printing.

wxRichTextStyleSheet* **wxRichTextBuffer::m_styleSheet** [protected]

Style sheet, if any.

int wxRichTextBuffer::m_suppressUndo [protected]

Whether to suppress undo.

float wxRichTextBuffer::sm_bulletProportion [static], [protected]

Factor to multiply by character height to get a reasonable bullet size.

int wxRichTextBuffer::sm_bulletRightMargin [static], [protected]

Minimum margin between bullet and paragraph in 10ths of a mm.

wxList wxRichTextBuffer::sm_drawingHandlers [static], [protected]

Drawing handlers.

wxRichTextFieldTypeHashMap wxRichTextBuffer::sm_fieldTypes [static], [protected]

Field types.

wxList wxRichTextBuffer::sm_handlers [static], [protected]

File handlers.

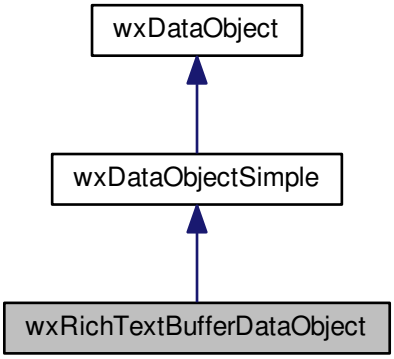
wxRichTextRenderer* **wxRichTextBuffer::sm_renderer** [static], [protected]

Renderer.

21.611 wxRichTextBufferDataObject Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextBufferDataObject:



21.611.1 Detailed Description

Implements a rich text data object for clipboard transfer.

Library: wxRichText

Category: Rich Text

See also

wxDataObjectSimple, wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextBufferDataObject (wxRichTextBuffer *richTextBuffer=NULL)

  The constructor doesn't copy the pointer, so it shouldn't go away while this object is alive.
- virtual ~wxRichTextBufferDataObject ()
- wxRichTextBuffer * GetRichTextBuffer ()

  After a call to this function, the buffer is owned by the caller and it is responsible for deleting it.
- virtual wxDataFormat GetPreferredFormat (Direction dir) const

  Returns the preferred format for either rendering the data (if dir is `Get`, its default value) or for setting it.
- virtual size_t GetDataSize () const

  Gets the size of our data.
- virtual bool GetDataHere (void *pBuf) const

  Copy the data to the buffer, return true on success.
- virtual bool SetData (size_t len, const void *buf)

  Copy the data from the buffer, return true on success.
- virtual size_t GetDataSize (const wxDataFormat &) const

  Returns the data size of the given format format.
- virtual bool GetDataHere (const wxDataFormat &, void *buf) const

  The method will write the data of the format format to the buffer buf.

- virtual bool SetData (const wxDataFormat &, size_t len, const void *buf)

  Set the data in the format format of the length len provided in the buffer buf.

Static Public Member Functions

- static const wxChar * GetRichTextBufferFormatId ()

  Returns the id for the new data format.

Private Attributes

- wxDataFormat m_formatRichTextBuffer
- wxRichTextBuffer * m_richTextBuffer

Static Private Attributes

- static const wxChar * ms_richTextBufferFormatId

Additional Inherited Members

21.611.2 Constructor & Destructor Documentation

wxRichTextBufferDataObject::wxRichTextBufferDataObject (wxRichTextBuffer * *richTextBuffer* = NULL)

The constructor doesn't copy the pointer, so it shouldn't go away while this object is alive.

virtual wxRichTextBufferDataObject::~wxRichTextBufferDataObject ( ) [virtual]

21.611.3 Member Function Documentation

virtual bool wxRichTextBufferDataObject::GetDataHere (void * *buf*) const [virtual]

Copy the data to the buffer, return true on success.

Must be implemented in the derived class if the object supports rendering its data.

Reimplemented from wxDataObjectSimple.

virtual bool wxRichTextBufferDataObject::GetDataHere (const wxDataFormat & *format*, void * *buf*) const [inline], [virtual]

The method will write the data of the format *format* to the buffer *buf*.

In other words, copy the data from this object in the given format to the supplied buffer. Returns true on success, false on failure.

Implements wxDataObject.

virtual size_t wxRichTextBufferDataObject::GetDataSize ( ) const [virtual]

Gets the size of our data.

Must be implemented in the derived class if the object supports rendering its data.

Reimplemented from wxDataObjectSimple.

virtual size_t wxRichTextBufferDataObject::GetDataSize (const wxDataFormat & *format*) const `[inline], [virtual]`

Returns the data size of the given format *format*.

Implements wxDataObject.

virtual wxDataFormat wxRichTextBufferDataObject::GetPreferredFormat (Direction *dir*) const `[virtual]`

Returns the preferred format for either rendering the data (if *dir* is `Get`, its default value) or for setting it.

Usually this will be the native format of the wxDataObject.

Implements wxDataObject.

wxRichTextBuffer* wxRichTextBufferDataObject::GetRichTextBuffer ( )

After a call to this function, the buffer is owned by the caller and it is responsible for deleting it.

static const wxChar* wxRichTextBufferDataObject::GetRichTextBufferFormatId ( ) `[inline], [static]`

Returns the id for the new data format.

virtual bool wxRichTextBufferDataObject::SetData (size_t *len*, const void * *buf*) `[virtual]`

Copy the data from the buffer, return true on success.

Must be implemented in the derived class if the object supports setting its data.

Reimplemented from wxDataObjectSimple.

virtual bool wxRichTextBufferDataObject::SetData (const wxDataFormat & *format*, size_t *len*, const void * *buf*) `[inline], [virtual]`

Set the data in the format *format* of the length *len* provided in the buffer *buf*.

In other words, copy length bytes of data from the buffer to this data object.

Parameters

format	The format for which to set the data.
len	The size of data in bytes.
buf	Non-NULL pointer to the data.

Returns

true on success, false on failure.

Reimplemented from wxDataObject.

21.611.4 Member Data Documentation

wxDataFormat wxRichTextBufferDataObject::m_formatRichTextBuffer `[private]`

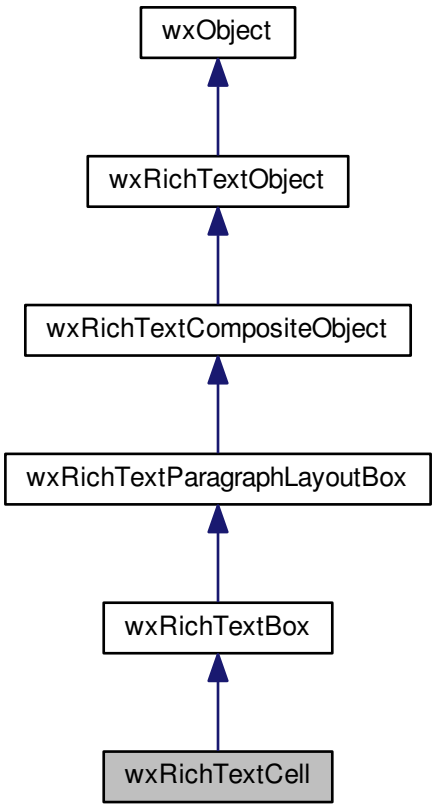
wxRichTextBuffer* wxRichTextBufferDataObject::m_richTextBuffer `[private]`

const wxChar* wxRichTextBufferDataObject::ms_richTextBufferFormatId [static],[private]

21.612 wxRichTextCell Class Reference

#include <wx/richtext/richtextbuffer.h>

Inheritance diagram for wxRichTextCell:



21.612.1 Detailed Description

wxRichTextCell is the cell in a table, in which the user can type.

As well as text, it can also contain objects e.g. an image, or even another wxRichTextTable.

A cell's appearance can be changed via its associated wxRichTextAttr; for example its size altered or its background colour set. It also has the properties of column- and row-span. By default these are 1, meaning that the cell only spans itself, but can be increased using the SetColSpan() and SetRowSpan() methods. Attempts to set too large a span are silently truncated to the table edge.

Public Member Functions

- wxRichTextCell (wxRichTextObject *parent=NULL)

  Default constructor; optionally pass the parent object.

- wxRichTextCell (const wxRichTextCell &obj)

  Copy constructor.
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

  Hit-testing: returns a flag indicating hit test details, plus information about position.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- virtual bool CanEditProperties () const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxWindow *parent, wxRichTextBuffer *buffer)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel () const

  Returns the label to be used for the properties context menu item.
- int GetColSpan () const

  Returns the number of columns spanned by the cell.
- void SetColSpan (long span)

  Set the number of columns spanned by the cell.
- int GetRowSpan () const

  Returns the number of rows spanned by the cell.
- void SetRowSpan (long span)

  Set the number of rows spanned by the cell.
- virtual wxRichTextObject * Clone () const

  Clones the object.
- void Copy (const wxRichTextCell &obj)

Additional Inherited Members

21.612.2 Constructor & Destructor Documentation

wxRichTextCell::wxRichTextCell (wxRichTextObject * *parent* = NULL)

Default constructor; optionally pass the parent object.

wxRichTextCell::wxRichTextCell (const wxRichTextCell & *obj*) `[inline]`

Copy constructor.

21.612.3 Member Function Documentation

virtual bool wxRichTextCell::CanEditProperties ( ) const `[inline], [virtual]`

Returns true if we can edit the object's properties via a GUI.

Reimplemented from wxRichTextBox.

virtual wxRichTextObject* wxRichTextCell::Clone ( ) const `[inline], [virtual]`

Clones the object.

Reimplemented from wxRichTextBox.

void wxRichTextCell::Copy (const wxRichTextCell & *obj*)

virtual bool wxRichTextCell::Draw (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) [virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Reimplemented from wxRichTextBox.

virtual bool wxRichTextCell::EditProperties (wxWindow * *parent,* wxRichTextBuffer * *buffer*) [virtual]

Edits the object's properties via a GUI.

Reimplemented from wxRichTextBox.

int wxRichTextCell::GetColSpan ( ) const

Returns the number of columns spanned by the cell.

By default a cell doesn't span extra columns, so this function returns 1.

Since

2.9.5

See also

SetColSpan(), GetRowSpan()

virtual wxString wxRichTextCell::GetPropertiesMenuLabel ( ) const [inline], [virtual]

Returns the label to be used for the properties context menu item.

Reimplemented from wxRichTextBox.

int wxRichTextCell::GetRowSpan ( ) const

Returns the number of rows spanned by the cell.

By default a cell doesn't span extra rows, so this function returns 1.

Since

2.9.5

See also

SetRowSpan(), GetColSpan()

virtual wxString wxRichTextCell::GetXMLNodeName ( ) const [inline], [virtual]

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextBox.

virtual int wxRichTextCell::HitTest (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxPoint & *pt,* long & *textPosition,* wxRichTextObject ** *obj,* wxRichTextObject ** *contextObj,* int *flags* = 0) [virtual]

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented from wxRichTextParagraphLayoutBox.

void wxRichTextCell::SetColSpan (long *span*)

Set the number of columns spanned by the cell.

By default colspan is 1 i.e. a cell doesn't span extra columns. Pass a value >1 to change this. Attempting to set a colspan <1 will assert and be ignored.

Since

2.9.5

See also

GetColSpan(), SetRowSpan()

void wxRichTextCell::SetRowSpan (long *span*)

Set the number of rows spanned by the cell.

By default colspan is 1 i.e. a cell doesn't span extra rows. Pass a value >1 to change this. Attempting to set a rowspan <1 will assert and be ignored.

Since

2.9.5

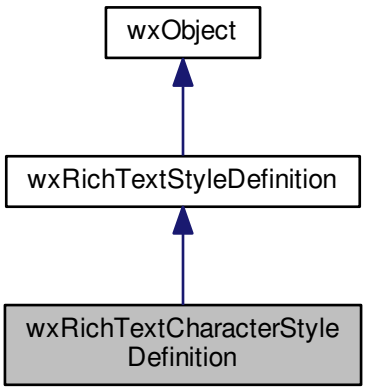
See also

GetRowSpan(), SetColSpan()

21.613 wxRichTextCharacterStyleDefinition Class Reference

```
#include <wx/richtext/richtextstyles.h>
```

Inheritance diagram for wxRichTextCharacterStyleDefinition:



21.613.1 Detailed Description

This class represents a character style definition, usually added to a wxRichTextStyleSheet.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextCharacterStyleDefinition (const wxString &name=wxEmptyString)

  Constructor.
- virtual ~wxRichTextCharacterStyleDefinition ()

  Destructor.

Additional Inherited Members

21.613.2 Constructor & Destructor Documentation

wxRichTextCharacterStyleDefinition::wxRichTextCharacterStyleDefinition (const wxString & *name* = wxEmptyString)

Constructor.

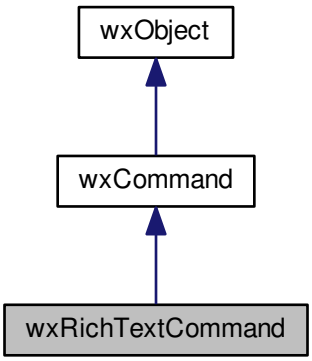
virtual wxRichTextCharacterStyleDefinition::~wxRichTextCharacterStyleDefinition ( ) [virtual]

Destructor.

21.614 wxRichTextCommand Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextCommand:



21.614.1 Detailed Description

Implements a command on the undo/redo stack.

A wxRichTextCommand object contains one or more wxRichTextAction objects, allowing aggregation of a number of operations into one command.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAction

Public Member Functions

- wxRichTextCommand (const wxString &name, wxRichTextCommandId id, wxRichTextBuffer *buffer, wxRichTextParagraphLayoutBox *container, wxRichTextCtrl *ctrl, bool ignoreFirstTime=false)

  Constructor for one action.
- wxRichTextCommand (const wxString &name)

  Constructor for multiple actions.
- virtual ~wxRichTextCommand ()
- bool Do ()

  Performs the command.
- bool Undo ()

  Undoes the command.
- void AddAction (wxRichTextAction *action)

Adds an action to the action list.

- void ClearActions ()

  Clears the action list.

- wxList & GetActions ()

  Returns the action list.

Protected Attributes

- wxList m_actions

Additional Inherited Members

21.614.2 Constructor & Destructor Documentation

wxRichTextCommand::wxRichTextCommand (const wxString & *name,* wxRichTextCommandId *id,* wxRichTextBuffer * *buffer,* wxRichTextParagraphLayoutBox * *container,* wxRichTextCtrl * *ctrl,* bool *ignoreFirstTime* = `false`)

Constructor for one action.

wxRichTextCommand::wxRichTextCommand (const wxString & *name*)

Constructor for multiple actions.

virtual wxRichTextCommand::~wxRichTextCommand ( ) `[virtual]`

21.614.3 Member Function Documentation

void wxRichTextCommand::AddAction (wxRichTextAction * *action*)

Adds an action to the action list.

void wxRichTextCommand::ClearActions ( )

Clears the action list.

bool wxRichTextCommand::Do ( ) `[virtual]`

Performs the command.

Implements wxCommand.

wxList& wxRichTextCommand::GetActions ( ) `[inline]`

Returns the action list.

bool wxRichTextCommand::Undo ( ) `[virtual]`

Undoes the command.

Implements wxCommand.

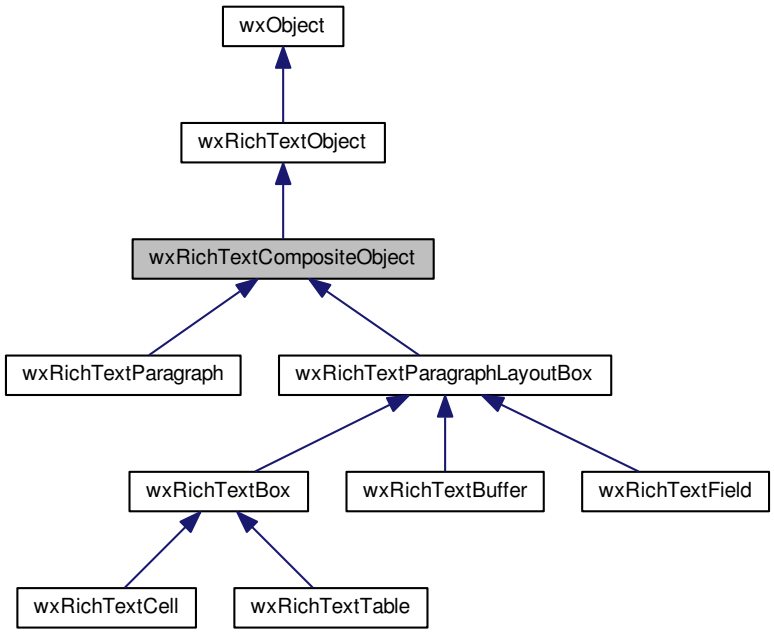
21.614.4 Member Data Documentation

wxList wxRichTextCommand::m_actions [protected]

21.615 wxRichTextCompositeObject Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextCompositeObject:



21.615.1 Detailed Description

Objects of this class can contain other objects.

Library: wxRichText

Category: Rich Text

See also

wxRichTextObject, wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextCompositeObject (wxRichTextObject *parent=NULL)
- virtual ~wxRichTextCompositeObject ()
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

Hit-testing: returns a flag indicating hit test details, plus information about position.

- virtual bool FindPosition (wxDC &dc, wxRichTextDrawingContext &context, long index, wxPoint &pt, int *height, bool forceLineStart)

  Finds the absolute position and row height for the given character position.

- virtual void CalculateRange (long start, long &end)

  Calculates the range of the object.

- virtual bool DeleteRange (const wxRichTextRange &range)

  Deletes the given range.

- virtual wxString GetTextForRange (const wxRichTextRange &range) const

  Returns any text in this object for the given range.

- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.

- virtual void Dump (wxTextOutputStream &stream)

  Dump object data to the given output stream for debugging.

- virtual void Invalidate (const wxRichTextRange &invalidRange=wxRICHTEXT_ALL)

  Invalidates the object at the given range.

- wxRichTextObjectList & GetChildren ()

  Returns the children.

- const wxRichTextObjectList & GetChildren () const

  Returns the children.

- size_t GetChildCount () const

  Returns the number of children.

- wxRichTextObject * GetChild (size_t n) const

  Returns the nth child.

- virtual bool IsComposite () const

  Returns true if this object is composite.

- virtual bool IsAtomic () const

  Returns true if no user editing can be done inside the object.

- virtual bool IsEmpty () const

  Returns true if the buffer is empty.

- virtual wxRichTextObject * GetChildAtPosition (long pos) const

  Returns the child object at the given character position.

- void Copy (const wxRichTextCompositeObject &obj)
- void operator= (const wxRichTextCompositeObject &obj)
- size_t AppendChild (wxRichTextObject *child)

  Appends a child, returning the position.

- bool InsertChild (wxRichTextObject *child, wxRichTextObject *inFrontOf)

  Inserts the child in front of the given object, or at the beginning.

- bool RemoveChild (wxRichTextObject *child, bool deleteChild=false)

  Removes and optionally deletes the specified child.

- bool DeleteChildren ()

  Deletes all the children.

- bool Defragment (wxRichTextDrawingContext &context, const wxRichTextRange &range=wxRICHTEXT_ALL)

  Recursively merges all pieces that can be merged.

- virtual void Move (const wxPoint &pt)

  Moves the object recursively, by adding the offset from old to new.

Protected Attributes

- wxRichTextObjectList m_children

Additional Inherited Members

21.615.2 Constructor & Destructor Documentation

wxRichTextCompositeObject::wxRichTextCompositeObject (wxRichTextObject * *parent* = NULL)

virtual wxRichTextCompositeObject::~wxRichTextCompositeObject ( ) [virtual]

21.615.3 Member Function Documentation

size_t wxRichTextCompositeObject::AppendChild (wxRichTextObject * *child*)

Appends a child, returning the position.

virtual void wxRichTextCompositeObject::CalculateRange (long *start*, long & *end*) [virtual]

Calculates the range of the object.

By default, guess that the object is 1 unit long.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, wxRichTextParagraph, and wxRichTextField.

void wxRichTextCompositeObject::Copy (const wxRichTextCompositeObject & *obj*)

bool wxRichTextCompositeObject::Defragment (wxRichTextDrawingContext & *context*, const wxRichTextRange & *range* = wxRICHTEXT_ALL)

Recursively merges all pieces that can be merged.

bool wxRichTextCompositeObject::DeleteChildren ( )

Deletes all the children.

virtual bool wxRichTextCompositeObject::DeleteRange (const wxRichTextRange & *range*) [virtual]

Deletes the given range.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextParagraphLayoutBox.

virtual void wxRichTextCompositeObject::Dump (wxTextOutputStream & *stream*) [virtual]

Dump object data to the given output stream for debugging.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextBuffer.

virtual bool wxRichTextCompositeObject::FindPosition (wxDC & *dc,* wxRichTextDrawingContext & *context,* long *index,* wxPoint & *pt,* int * *height,* bool *forceLineStart*) [virtual]

Finds the absolute position and row height for the given character position.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextParagraph.

wxRichTextObject* wxRichTextCompositeObject::GetChild (size_t *n*) const

Returns the nth child.

virtual wxRichTextObject* wxRichTextCompositeObject::GetChildAtPosition (long *pos*) const [virtual]

Returns the child object at the given character position.

size_t wxRichTextCompositeObject::GetChildCount ( ) const

Returns the number of children.

wxRichTextObjectList& wxRichTextCompositeObject::GetChildren ( )

Returns the children.

const wxRichTextObjectList& wxRichTextCompositeObject::GetChildren ( ) const

Returns the children.

virtual bool wxRichTextCompositeObject::GetRangeSize (const wxRichTextRange & *range,* wxSize & *size,* int & *descent,* wxDC & *dc,* wxRichTextDrawingContext & *context,* int *flags,* const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Implements wxRichTextObject.

Reimplemented in wxRichTextTable, wxRichTextParagraph, wxRichTextField, and wxRichTextParagraphLayoutBox.

virtual wxString wxRichTextCompositeObject::GetTextForRange (const wxRichTextRange & *range*) const [virtual]

Returns any text in this object for the given range.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextParagraphLayoutBox.

virtual int wxRichTextCompositeObject::HitTest (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxPoint & *pt,* long & *textPosition,* wxRichTextObject ** *obj,* wxRichTextObject ** *contextObj,* int *flags* = 0) [virtual]

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextBuffer, wxRichTextParagraph, and wxRichTextParagraphLayoutBox.

bool wxRichTextCompositeObject::InsertChild (wxRichTextObject * *child,* wxRichTextObject * *inFrontOf*)

Inserts the child in front of the given object, or at the beginning.

virtual void wxRichTextCompositeObject::Invalidate (const wxRichTextRange & *invalidRange* = wxRICHTEXT_ALL) [virtual]

Invalidates the object at the given range.

With no argument, invalidates the whole object.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextParagraphLayoutBox.

virtual bool wxRichTextCompositeObject::IsAtomic ( ) const [virtual]

Returns true if no user editing can be done inside the object.

This returns true for simple objects, false for most composite objects, but true for fields, which if composite, should not be user-edited.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextField.

virtual bool wxRichTextCompositeObject::IsComposite ( ) const [virtual]

Returns true if this object is composite.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextCompositeObject::IsEmpty ( ) const [virtual]

Returns true if the buffer is empty.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextField.

virtual void wxRichTextCompositeObject::Move (const wxPoint & *pt*) [virtual]

Moves the object recursively, by adding the offset from old to new.

Reimplemented from wxRichTextObject.

void wxRichTextCompositeObject::operator= (const wxRichTextCompositeObject & *obj*)

bool wxRichTextCompositeObject::RemoveChild (wxRichTextObject * *child*, bool *deleteChild* = `false`)

Removes and optionally deletes the specified child.

21.615.4 Member Data Documentation

wxRichTextObjectList wxRichTextCompositeObject::m_children `[protected]`

21.616 wxRichTextContextMenuPropertiesInfo Class Reference

`#include <wx/richtext/richtextctrl.h>`

21.616.1 Detailed Description

wxRichTextContextMenuPropertiesInfo keeps track of objects that appear in the context menu, whose properties are available to be edited.

Public Member Functions

- wxRichTextContextMenuPropertiesInfo ()

  Constructor.
- void Init ()

  Initialisation.
- bool AddItem (const wxString &label, wxRichTextObject *obj)

  Adds an item.
- int AddMenuItems (wxMenu *menu, int startCmd=wxID_RICHTEXT_PROPERTIES1) const

  Returns the number of menu items that were added.
- int AddItems (wxRichTextCtrl *ctrl, wxRichTextObject *container, wxRichTextObject *obj)

  Adds appropriate menu items for the current container and clicked on object (and container's parent, if appropriate).
- void Clear ()

  Clears the items.
- wxString GetLabel (int n) const

  Returns the nth label.
- wxRichTextObject * GetObject (int n) const

  Returns the nth object.
- wxRichTextObjectPtrArray & GetObjects ()

  Returns the array of objects.
- const wxRichTextObjectPtrArray & GetObjects () const

  Returns the array of objects.
- wxArrayString & GetLabels ()

  Returns the array of labels.
- const wxArrayString & GetLabels () const

  Returns the array of labels.
- int GetCount () const

  Returns the number of items.

Public Attributes

- wxRichTextObjectPtrArray m_objects
- wxArrayString m_labels

21.616.2 Constructor & Destructor Documentation

wxRichTextContextMenuPropertiesInfo::wxRichTextContextMenuPropertiesInfo ( )

Constructor.

21.616.3 Member Function Documentation

bool wxRichTextContextMenuPropertiesInfo::AddItem (const wxString & *label*, wxRichTextObject * *obj*)

Adds an item.

int wxRichTextContextMenuPropertiesInfo::AddItems (wxRichTextCtrl * *ctrl*, wxRichTextObject * *container*, wxRichTextObject * *obj*)

Adds appropriate menu items for the current container and clicked on object (and container's parent, if appropriate).

int wxRichTextContextMenuPropertiesInfo::AddMenuItems (wxMenu * *menu*, int *startCmd* = wxID_RICHTEXT_PROPERTIES1) const

Returns the number of menu items that were added.

void wxRichTextContextMenuPropertiesInfo::Clear ( )

Clears the items.

int wxRichTextContextMenuPropertiesInfo::GetCount ( ) const

Returns the number of items.

wxString wxRichTextContextMenuPropertiesInfo::GetLabel (int *n*) const

Returns the nth label.

wxArrayString& wxRichTextContextMenuPropertiesInfo::GetLabels ( )

Returns the array of labels.

const wxArrayString& wxRichTextContextMenuPropertiesInfo::GetLabels ( ) const

Returns the array of labels.

wxRichTextObject* wxRichTextContextMenuPropertiesInfo::GetObject (int *n*) const

Returns the nth object.

wxRichTextObjectPtrArray& wxRichTextContextMenuPropertiesInfo::GetObjects ( )

Returns the array of objects.

const wxRichTextObjectPtrArray& wxRichTextContextMenuPropertiesInfo::GetObjects ( ) const

Returns the array of objects.

void wxRichTextContextMenuPropertiesInfo::Init ( )

Initialisation.

21.616.4 Member Data Documentation

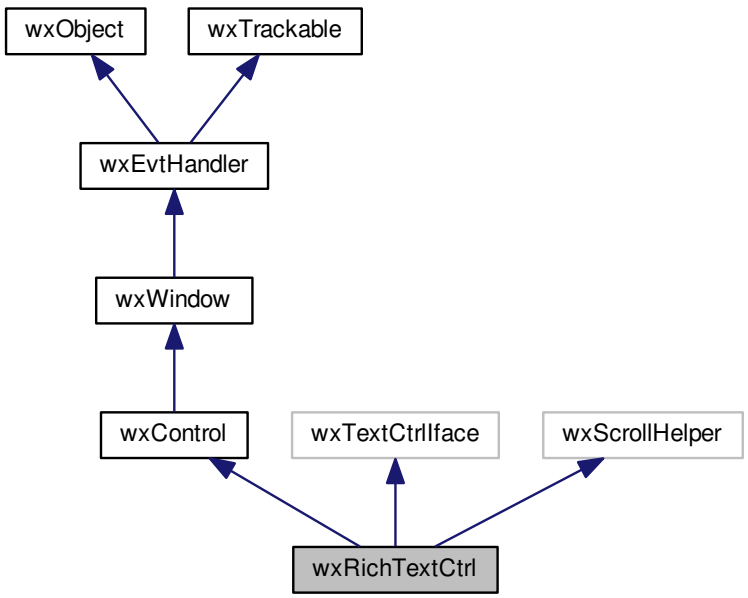
wxArrayString wxRichTextContextMenuPropertiesInfo::m_labels

wxRichTextObjectPtrArray wxRichTextContextMenuPropertiesInfo::m_objects

21.617 wxRichTextCtrl Class Reference

`#include <wx/richtext/richtextctrl.h>`

Inheritance diagram for wxRichTextCtrl:



21.617.1 Detailed Description

wxRichTextCtrl provides a generic, ground-up implementation of a text control capable of showing multiple styles and images.

wxRichTextCtrl sends notification events: see wxRichTextEvent.

It also sends the standard wxTextCtrl events `wxEVT_TEXT_ENTER` and `wxEVT_TEXT`, and wxTextUrlEvent when URL content is clicked.

For more information, see the wxRichTextCtrl Overview.

Styles

This class supports the following styles:

- wxRE_CENTRE_CARET: The control will try to keep the caret line centred vertically while editing. wxRE_↩ CENTER_CARET is a synonym for this style.
- wxRE_MULTILINE: The control will be multiline (mandatory).
- wxRE_READONLY: The control will not be editable.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextCtrl ()

  Default constructor.
- wxRichTextCtrl (wxWindow *parent, wxWindowID id=-1, const wxString &value=wxEmptyString, const wx↩ Point &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxRE_MULTILINE, const wx↩ Validator &validator=wxDefaultValidator, const wxString &name=wxTextCtrlNameStr)

  Constructor, creating and showing a rich text control.
- virtual ~wxRichTextCtrl ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id=-1, const wxString &value=wxEmptyString, const wx↩ Point &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxRE_MULTILINE, const wx↩ Validator &validator=wxDefaultValidator, const wxString &name=wxTextCtrlNameStr)

  Creates the underlying window.
- void Init ()

  Initialises the members of the control.
- virtual wxString GetRange (long from, long to) const

  Gets the text for the given range.
- virtual int GetLineLength (long lineNo) const

  Returns the length of the specified line in characters.
- virtual wxString GetLineText (long lineNo) const

  Returns the text for the given line.
- virtual int GetNumberOfLines () const

  Returns the number of lines in the buffer.
- virtual bool IsModified () const

  Returns true if the buffer has been modified.
- virtual bool IsEditable () const

  Returns true if the control is editable.
- bool IsSingleLine () const

  Returns true if the control is single-line.

- bool IsMultiLine () const

  Returns true if the control is multiline.
- virtual wxString GetStringSelection () const

  Returns the text within the current selection range, if any.
- wxString GetFilename () const

  Gets the current filename associated with the control.
- void SetFilename (const wxString &filename)

  Sets the current filename.
- void SetDelayedLayoutThreshold (long threshold)

  Sets the size of the buffer beyond which layout is delayed during resizing.
- long GetDelayedLayoutThreshold () const

  Gets the size of the buffer beyond which layout is delayed during resizing.
- bool GetFullLayoutRequired () const
- void SetFullLayoutRequired (bool b)
- wxLongLong GetFullLayoutTime () const
- void SetFullLayoutTime (wxLongLong t)
- long GetFullLayoutSavedPosition () const
- void SetFullLayoutSavedPosition (long p)
- void ForceDelayedLayout ()
- void SetTextCursor (const wxCursor &cursor)

  Sets the text (normal) cursor.
- wxCursor GetTextCursor () const

  Returns the text (normal) cursor.
- void SetURLCursor (const wxCursor &cursor)

  Sets the cursor to be used over URLs.
- wxCursor GetURLCursor () const

  Returns the cursor to be used over URLs.
- bool GetCaretAtLineStart () const

  Returns true if we are showing the caret position at the start of a line instead of at the end of the previous one.
- void SetCaretAtLineStart (bool atStart)

  Sets a flag to remember that we are showing the caret position at the start of a line instead of at the end of the previous one.
- bool GetDragging () const

  Returns true if we are extending a selection.
- void SetDragging (bool dragging)

  Sets a flag to remember if we are extending a selection.
- bool GetPreDrag () const

  Are we trying to start Drag'n'Drop?
- void SetPreDrag (bool pd)

  Set if we're trying to start Drag'n'Drop.
- const wxPoint GetDragStartPoint () const

  Get the possible Drag'n'Drop start point.
- void SetDragStartPoint (wxPoint sp)

  Set the possible Drag'n'Drop start point.
- const wxDateTime GetDragStartTime () const

  Get the possible Drag'n'Drop start time.
- void SetDragStartTime (wxDateTime st)

  Set the possible Drag'n'Drop start time.
- wxMenu * GetContextMenu () const

  Returns the current context menu.
- void SetContextMenu (wxMenu *menu)

Sets the current context menu.

- long GetSelectionAnchor () const

  Returns an anchor so we know how to extend the selection.
- void SetSelectionAnchor (long anchor)

  Sets an anchor so we know how to extend the selection.
- wxRichTextObject * GetSelectionAnchorObject () const

  Returns the anchor object if selecting multiple containers.
- void SetSelectionAnchorObject (wxRichTextObject *anchor)

  Sets the anchor object if selecting multiple containers.
- wxRichTextParagraphLayoutBox * GetFocusObject () const

  Returns the wxRichTextObject object that currently has the editing focus.
- void StoreFocusObject (wxRichTextParagraphLayoutBox *obj)

  Setter for m_focusObject.
- bool SetFocusObject (wxRichTextParagraphLayoutBox *obj, bool setCaretPosition=true)

  Sets the wxRichTextObject object that currently has the editing focus.
- void Invalidate ()

  Invalidates the whole buffer to trigger painting later.
- virtual void Clear ()

  Clears the buffer content, leaving a single empty paragraph.
- virtual void Replace (long from, long to, const wxString &value)

  Replaces the content in the specified range with the string specified by value.
- virtual void Remove (long from, long to)

  Removes the content in the specified range.
- bool LoadFile (const wxString &file, int type=wxRICHTEXT_TYPE_ANY)

  Loads content into the control's buffer using the given type.
- virtual bool DoLoadFile (const wxString &file, int fileType)

  Helper function for LoadFile().
- bool SaveFile (const wxString &file=wxEmptyString, int type=wxRICHTEXT_TYPE_ANY)

  Saves the buffer content using the given type.
- virtual bool DoSaveFile (const wxString &file=wxEmptyString, int fileType=wxRICHTEXT_TYPE_ANY)

  Helper function for SaveFile().
- void SetHandlerFlags (int flags)

  Sets flags that change the behaviour of loading or saving.
- int GetHandlerFlags () const

  Returns flags that change the behaviour of loading or saving.
- virtual void MarkDirty ()

  Marks the buffer as modified.
- virtual void DiscardEdits ()

  Sets the buffer's modified status to false, and clears the buffer's command history.
- void SetModified (bool modified)
- virtual void SetMaxLength (unsigned long len)

  Sets the maximum number of characters that may be entered in a single line text control.
- virtual void WriteText (const wxString &text)

  Writes text at the current position.
- virtual void AppendText (const wxString &text)

  Sets the insertion point to the end of the buffer and writes the text.
- virtual void SetStyle (wxRichTextObject *obj, const wxRichTextAttr &textAttr, int flags=wxRICHTEXT_SET↩ STYLE_WITH_UNDO)

  Sets the attributes for a single object.
- virtual bool SetStyleEx (const wxRichTextRange &range, const wxRichTextAttr &style, int flags=wxRICHTE↩ XT_SETSTYLE_WITH_UNDO)

Sets the attributes for the given range, passing flags to determine how the attributes are set.

- virtual const wxRichTextAttr & GetDefaultStyleEx () const

  Returns the current default style, which can be used to change how subsequently inserted text is displayed.

- virtual bool ClearListStyle (const wxRichTextRange &range, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO)

  Clears the list style from the given range, clearing list-related attributes and applying any named paragraph style associated with each paragraph.

- virtual bool SetProperties (const wxRichTextRange &range, const wxRichTextProperties &properties, int flags=wxRICHTEXT_SETPROPERTIES_WITH_UNDO)

  Sets the properties for the given range, passing flags to determine how the attributes are set.

- virtual bool Delete (const wxRichTextRange &range)

  Deletes the content within the given range.

- virtual long XYToPosition (long x, long y) const

  Translates from column and line number to position.

- virtual bool PositionToXY (long pos, long *x, long *y) const

  Converts a text position to zero-based column and line numbers.

- virtual void ShowPosition (long pos)

  Scrolls the buffer so that the given position is in view.

- virtual void Copy ()

  Copies the selected content (if any) to the clipboard.

- virtual void Cut ()

  Copies the selected content (if any) to the clipboard and deletes the selection.

- virtual void Paste ()

  Pastes content from the clipboard to the buffer.

- virtual void DeleteSelection ()

  Deletes the content in the selection, if any.

- virtual bool CanCopy () const

  Returns true if selected content can be copied to the clipboard.

- virtual bool CanCut () const

  Returns true if selected content can be copied to the clipboard and deleted.

- virtual bool CanPaste () const

  Returns true if the clipboard content can be pasted to the buffer.

- virtual bool CanDeleteSelection () const

  Returns true if selected content can be deleted.

- virtual void Undo ()

  Undoes the command at the top of the command history, if there is one.

- virtual void Redo ()

  Redoes the current command.

- virtual bool CanUndo () const

  Returns true if there is a command in the command history that can be undone.

- virtual bool CanRedo () const

  Returns true if there is a command in the command history that can be redone.

- virtual void SetInsertionPoint (long pos)

  Sets the insertion point and causes the current editing style to be taken from the new position (unlike wxRichTextCtrl::SetCaretPosition).

- virtual void SetInsertionPointEnd ()

  Sets the insertion point to the end of the text control.

- virtual long GetInsertionPoint () const

  Returns the current insertion point.

- virtual wxTextPos GetLastPosition () const

  Returns the last position in the buffer.

- virtual void SelectAll ()

  Selects all the text in the buffer.
- virtual void SetEditable (bool editable)

  Makes the control editable, or not.
- virtual bool HasSelection () const

  Returns true if there is a selection and the object containing the selection was the same as the current focus object.
- virtual bool HasUnfocusedSelection () const

  Returns true if there was a selection, whether or not the current focus object is the same as the selection's container object.
- virtual bool WriteImage (const wxString &filename, wxBitmapType bitmapType, const wxRichTextAttr &textAttr=wxRichTextAttr())

  Loads an image from a file and writes it at the current insertion point.
- virtual bool WriteImage (const wxRichTextImageBlock &imageBlock, const wxRichTextAttr &textAttr=wxRichTextAttr())

  Writes an image block at the current insertion point.
- virtual wxRichTextBox * WriteTextBox (const wxRichTextAttr &textAttr=wxRichTextAttr())

  Write a text box at the current insertion point, returning the text box.
- virtual wxRichTextField * WriteField (const wxString &fieldType, const wxRichTextProperties &properties, const wxRichTextAttr &textAttr=wxRichTextAttr())

  Writes a field at the current insertion point.
- virtual wxRichTextTable * WriteTable (int rows, int cols, const wxRichTextAttr &tableAttr=wxRichTextAttr(), const wxRichTextAttr &cellAttr=wxRichTextAttr())

  Write a table at the current insertion point, returning the table.
- virtual bool Newline ()

  Inserts a new paragraph at the current insertion point.
- virtual bool LineBreak ()

  Inserts a line break at the current insertion point.
- virtual void SetBasicStyle (const wxRichTextAttr &style)

  Sets the basic (overall) style.
- virtual const wxRichTextAttr & GetBasicStyle () const

  Gets the basic (overall) style.
- virtual bool BeginStyle (const wxRichTextAttr &style)

  Begins applying a style.
- virtual bool EndStyle ()

  Ends the current style.
- virtual bool EndAllStyles ()

  Ends application of all styles in the current style stack.
- bool BeginBold ()

  Begins using bold.
- bool EndBold ()

  Ends using bold.
- bool BeginItalic ()

  Begins using italic.
- bool EndItalic ()

  Ends using italic.
- bool BeginUnderline ()

  Begins using underlining.
- bool EndUnderline ()

  End applying underlining.
- bool BeginFontSize (int pointSize)

  Begins using the given point size.

- bool EndFontSize ()

  Ends using a point size.
- bool BeginFont (const wxFont &font)

  Begins using this font.
- bool EndFont ()

  Ends using a font.
- bool BeginTextColour (const wxColour &colour)

  Begins using this colour.
- bool EndTextColour ()

  Ends applying a text colour.
- bool BeginAlignment (wxTextAttrAlignment alignment)

  Begins using alignment.
- bool EndAlignment ()

  Ends alignment.
- bool BeginLeftIndent (int leftIndent, int leftSubIndent=0)

  Begins applying a left indent and subindent in tenths of a millimetre.
- bool EndLeftIndent ()

  Ends left indent.
- bool BeginRightIndent (int rightIndent)

  Begins a right indent, specified in tenths of a millimetre.
- bool EndRightIndent ()

  Ends right indent.
- bool BeginParagraphSpacing (int before, int after)

  Begins paragraph spacing; pass the before-paragraph and after-paragraph spacing in tenths of a millimetre.
- bool EndParagraphSpacing ()

  Ends paragraph spacing.
- bool BeginLineSpacing (int lineSpacing)

  Begins appling line spacing.
- bool EndLineSpacing ()

  Ends line spacing.
- bool BeginNumberedBullet (int bulletNumber, int leftIndent, int leftSubIndent, int bulletStyle=wxTEXT_ATTR_BULLET_STYLE_ARABIC|wxTEXT_ATTR_BULLET_STYLE_PERIOD)

  Begins a numbered bullet.
- bool EndNumberedBullet ()

  Ends application of a numbered bullet.
- bool BeginSymbolBullet (const wxString &symbol, int leftIndent, int leftSubIndent, int bulletStyle=wxTEXT_ATTR_BULLET_STYLE_SYMBOL)

  Begins applying a symbol bullet, using a character from the current font.
- bool EndSymbolBullet ()

  Ends applying a symbol bullet.
- bool BeginStandardBullet (const wxString &bulletName, int leftIndent, int leftSubIndent, int bulletStyle=wxTEXT_ATTR_BULLET_STYLE_STANDARD)

  Begins applying a symbol bullet.
- bool EndStandardBullet ()

  Begins applying a standard bullet.
- bool BeginCharacterStyle (const wxString &characterStyle)

  Begins using the named character style.
- bool EndCharacterStyle ()

  Ends application of a named character style.
- bool BeginParagraphStyle (const wxString &paragraphStyle)

  Begins applying the named paragraph style.

- bool EndParagraphStyle ()

  Ends application of a named paragraph style.
- bool BeginListStyle (const wxString &listStyle, int level=1, int number=1)

  Begins using a specified list style.
- bool EndListStyle ()

  Ends using a specified list style.
- bool BeginURL (const wxString &url, const wxString &characterStyle=wxEmptyString)

  Begins applying wxTEXT_ATTR_URL to the content.
- bool EndURL ()

  Ends applying a URL.
- bool SetDefaultStyleToCursorStyle ()

  Sets the default style to the style under the cursor.
- virtual void SelectNone ()

  Cancels any selection.
- virtual bool SelectWord (long position)

  Selects the word at the given character position.
- wxRichTextRange GetSelectionRange () const

  Returns the selection range in character positions.
- void SetSelectionRange (const wxRichTextRange &range)

  Sets the selection to the given range.
- wxRichTextRange GetInternalSelectionRange () const

  Returns the selection range in character positions.
- void SetInternalSelectionRange (const wxRichTextRange &range)

  Sets the selection range in character positions.
- virtual wxRichTextRange AddParagraph (const wxString &text)

  Adds a new paragraph of text to the end of the buffer.
- virtual wxRichTextRange AddImage (const wxImage &image)

  Adds an image to the control's buffer.
- virtual bool LayoutContent (bool onlyVisibleRect=false)

  Lays out the buffer, which must be done before certain operations, such as setting the caret position.
- virtual void DoLayoutBuffer (wxRichTextBuffer &buffer, wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int flags)

  Implements layout.
- virtual bool MoveCaret (long pos, bool showAtLineStart=false, wxRichTextParagraphLayoutBox *container=NULL)

  Move the caret to the given character position.
- virtual bool MoveRight (int noPositions=1, int flags=0)

  Moves right.
- virtual bool MoveLeft (int noPositions=1, int flags=0)

  Moves left.
- virtual bool MoveUp (int noLines=1, int flags=0)

  Moves to the start of the paragraph.
- virtual bool MoveDown (int noLines=1, int flags=0)

  Moves the caret down.
- virtual bool MoveToLineEnd (int flags=0)

  Moves to the end of the line.
- virtual bool MoveToLineStart (int flags=0)

  Moves to the start of the line.
- virtual bool MoveToParagraphEnd (int flags=0)

  Moves to the end of the paragraph.
- virtual bool MoveToParagraphStart (int flags=0)

Moves to the start of the paragraph.

- virtual bool MoveHome (int flags=0)

  Moves to the start of the buffer.
- virtual bool MoveEnd (int flags=0)

  Moves to the end of the buffer.
- virtual bool PageUp (int noPages=1, int flags=0)

  Moves one or more pages up.
- virtual bool PageDown (int noPages=1, int flags=0)

  Moves one or more pages down.
- virtual bool WordLeft (int noPages=1, int flags=0)

  Moves a number of words to the left.
- virtual bool WordRight (int noPages=1, int flags=0)

  Move a nuber of words to the right.
- virtual bool BeginBatchUndo (const wxString &cmdName)

  Starts batching undo history for commands.
- virtual bool EndBatchUndo ()

  Ends batching undo command history.
- virtual bool BatchingUndo () const

  Returns true if undo commands are being batched.
- virtual bool BeginSuppressUndo ()

  Starts suppressing undo history for commands.
- virtual bool EndSuppressUndo ()

  Ends suppressing undo command history.
- virtual bool SuppressingUndo () const

  Returns true if undo history suppression is on.
- virtual bool HasCharacterAttributes (const wxRichTextRange &range, const wxRichTextAttr &style) const

  Test if this whole range has character attributes of the specified kind.
- virtual bool HasParagraphAttributes (const wxRichTextRange &range, const wxRichTextAttr &style) const

  Test if this whole range has paragraph attributes of the specified kind.
- virtual bool IsSelectionBold ()

  Returns true if all of the selection, or the content at the caret position, is bold.
- virtual bool IsSelectionItalics ()

  Returns true if all of the selection, or the content at the caret position, is italic.
- virtual bool IsSelectionUnderlined ()

  Returns true if all of the selection, or the content at the caret position, is underlined.
- virtual bool DoesSelectionHaveTextEffectFlag (int flag)

  Returns true if all of the selection, or the content at the current caret position, has the supplied wxTextAttrEffects flag(s).
- virtual bool IsSelectionAligned (wxTextAttrAlignment alignment)

  Returns true if all of the selection is aligned according to the specified flag.
- virtual bool ApplyBoldToSelection ()

  Apples bold to the selection or the default style (undoable).
- virtual bool ApplyItalicToSelection ()

  Applies italic to the selection or the default style (undoable).
- virtual bool ApplyUnderlineToSelection ()

  Applies underline to the selection or the default style (undoable).
- virtual bool ApplyTextEffectToSelection (int flags)

  Applies one or more wxTextAttrEffects flags to the selection (undoable).
- virtual bool ApplyAlignmentToSelection (wxTextAttrAlignment alignment)

  Applies the given alignment to the selection or the default style (undoable).
- virtual bool ApplyStyle (wxRichTextStyleDefinition *def)

Applies the style sheet to the buffer, matching paragraph styles in the sheet against named styles in the buffer.

- void SetStyleSheet (wxRichTextStyleSheet *styleSheet)

  Sets the style sheet associated with the control.
- wxRichTextStyleSheet * GetStyleSheet () const

  Returns the style sheet associated with the control, if any.
- bool PushStyleSheet (wxRichTextStyleSheet *styleSheet)

  Push the style sheet to top of stack.
- wxRichTextStyleSheet * PopStyleSheet ()

  Pops the style sheet from top of stack.
- bool ApplyStyleSheet (wxRichTextStyleSheet *styleSheet=NULL)

  Applies the style sheet to the buffer, for example if the styles have changed.
- virtual bool ShowContextMenu (wxMenu *menu, const wxPoint &pt, bool addPropertyCommands)

  Shows the given context menu, optionally adding appropriate property-editing commands for the current position in the object hierarchy.
- virtual int PrepareContextMenu (wxMenu *menu, const wxPoint &pt, bool addPropertyCommands)

  Prepares the context menu, optionally adding appropriate property-editing commands.
- virtual bool CanEditProperties (wxRichTextObject *obj) const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxRichTextObject *obj, wxWindow *parent)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel (wxRichTextObject *obj)

  Gets the object's properties menu label.
- virtual void PrepareContent (wxRichTextParagraphLayoutBox &container)

  Prepares the content just before insertion (or after buffer reset).
- virtual bool CanDeleteRange (wxRichTextParagraphLayoutBox &container, const wxRichTextRange &range) const

  Can we delete this range? Sends an event to the control.
- virtual bool CanInsertContent (wxRichTextParagraphLayoutBox &container, long pos) const

  Can we insert content at this position? Sends an event to the control.
- virtual void EnableVerticalScrollbar (bool enable)

  Enable or disable the vertical scrollbar.
- virtual bool GetVerticalScrollbarEnabled () const

  Returns true if the vertical scrollbar is enabled.
- void SetFontScale (double fontScale, bool refresh=false)

  Sets the scale factor for displaying fonts, for example for more comfortable editing.
- double GetFontScale () const

  Returns the scale factor for displaying fonts, for example for more comfortable editing.
- void SetDimensionScale (double dimScale, bool refresh=false)

  Sets the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.
- double GetDimensionScale () const

  Returns the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.
- void SetScale (double scale, bool refresh=false)

  Sets an overall scale factor for displaying and editing the content.
- double GetScale () const

  Returns an overall scale factor for displaying and editing the content.
- wxPoint GetUnscaledPoint (const wxPoint &pt) const

  Returns an unscaled point.
- wxPoint GetScaledPoint (const wxPoint &pt) const

  Returns a scaled point.
- wxSize GetUnscaledSize (const wxSize &sz) const

  Returns an unscaled size.

- wxSize GetScaledSize (const wxSize &sz) const

  Returns a scaled size.
- wxRect GetUnscaledRect (const wxRect &rect) const

  Returns an unscaled rectangle.
- wxRect GetScaledRect (const wxRect &rect) const

  Returns a scaled rectangle.
- bool GetVirtualAttributesEnabled () const

  Returns true if this control can use virtual attributes and virtual text.
- void EnableVirtualAttributes (bool b)

  Pass true to let the control use virtual attributes.
- void Command (wxCommandEvent &event)

  Sends the event to the control.
- void OnDropFiles (wxDropFilesEvent &event)

  Loads the first dropped file.
- void OnCaptureLost (wxMouseCaptureLostEvent &event)
- void OnSysColourChanged (wxSysColourChangedEvent &event)
- void OnCut (wxCommandEvent &event)

  Standard handler for the wxID_CUT command.
- void OnCopy (wxCommandEvent &event)

  Standard handler for the wxID_COPY command.
- void OnPaste (wxCommandEvent &event)

  Standard handler for the wxID_PASTE command.
- void OnUndo (wxCommandEvent &event)

  Standard handler for the wxID_UNDO command.
- void OnRedo (wxCommandEvent &event)

  Standard handler for the wxID_REDO command.
- void OnSelectAll (wxCommandEvent &event)

  Standard handler for the wxID_SELECTALL command.
- void OnProperties (wxCommandEvent &event)

  Standard handler for property commands.
- void OnClear (wxCommandEvent &event)

  Standard handler for the wxID_CLEAR command.
- void OnUpdateCut (wxUpdateUIEvent &event)

  Standard update handler for the wxID_CUT command.
- void OnUpdateCopy (wxUpdateUIEvent &event)

  Standard update handler for the wxID_COPY command.
- void OnUpdatePaste (wxUpdateUIEvent &event)

  Standard update handler for the wxID_PASTE command.
- void OnUpdateUndo (wxUpdateUIEvent &event)

  Standard update handler for the wxID_UNDO command.
- void OnUpdateRedo (wxUpdateUIEvent &event)

  Standard update handler for the wxID_REDO command.
- void OnUpdateSelectAll (wxUpdateUIEvent &event)

  Standard update handler for the wxID_SELECTALL command.
- void OnUpdateProperties (wxUpdateUIEvent &event)

  Standard update handler for property commands.
- void OnUpdateClear (wxUpdateUIEvent &event)

  Standard update handler for the wxID_CLEAR command.
- void OnContextMenu (wxContextMenuEvent &event)

  Shows a standard context menu with undo, redo, cut, copy, paste, clear, and select all commands.
- void OnPaint (wxPaintEvent &event)

- void OnEraseBackground (wxEraseEvent &event)
- void OnLeftClick (wxMouseEvent &event)
- void OnLeftUp (wxMouseEvent &event)
- void OnMoveMouse (wxMouseEvent &event)
- void OnLeftDClick (wxMouseEvent &event)
- void OnMiddleClick (wxMouseEvent &event)
- void OnRightClick (wxMouseEvent &event)
- void OnChar (wxKeyEvent &event)
- void OnSize (wxSizeEvent &event)
- void OnSetFocus (wxFocusEvent &event)
- void OnKillFocus (wxFocusEvent &event)
- void OnIdle (wxIdleEvent &event)
- void OnScroll (wxScrollWinEvent &event)
- virtual bool SetFont (const wxFont &font)

  Sets the font, and also the basic and default attributes (see wxRichTextCtrl::SetDefaultStyle).
- virtual void SetupScrollbars (bool atTop=false)

  A helper function setting up scrollbars, for example after a resize.
- virtual bool KeyboardNavigate (int keyCode, int flags)

  Helper function implementing keyboard navigation.
- virtual void PaintBackground (wxDC &dc)

  Paints the background.
- virtual void PaintAboveContent (wxDC &WXUNUSED(dc))

  Other user defined painting after everything else (i.e. all text) is painted.
- virtual void DoWriteText (const wxString &value, int flags=0)
- virtual bool ShouldInheritColours () const

  Return true from here to allow the colours of this window to be changed by InheritAttributes().
- virtual void PositionCaret (wxRichTextParagraphLayoutBox *container=NULL)

  Internal function to position the visible caret according to the current caret position.
- virtual bool ExtendSelection (long oldPosition, long newPosition, int flags)

  Helper function for extending the selection, returning true if the selection was changed.
- virtual bool ExtendCellSelection (wxRichTextTable *table, int noRowSteps, int noColSteps)

  Extends a table selection in the given direction.
- virtual bool StartCellSelection (wxRichTextTable *table, wxRichTextParagraphLayoutBox *newCell)

  Starts selecting table cells.
- virtual bool ScrollIntoView (long position, int keyCode)

  Scrolls position into view.
- bool RefreshForSelectionChange (const wxRichTextSelection &oldSelection, const wxRichTextSelection &newSelection)

  Refreshes the area affected by a selection change.
- void SetCaretPosition (long position, bool showAtLineStart=false)

  Sets the caret position.
- long GetCaretPosition () const

  Returns the current caret position.
- long GetAdjustedCaretPosition (long caretPos) const

  The adjusted caret position is the character position adjusted to take into account whether we're at the start of a paragraph, in which case style information should be taken from the next position, not current one.
- void MoveCaretForward (long oldPosition)

  Move the caret one visual step forward: this may mean setting a flag and keeping the same position if we're going from the end of one line to the start of the next, which may be the exact same caret position.
- void MoveCaretBack (long oldPosition)

  Move the caret one visual step forward: this may mean setting a flag and keeping the same position if we're going from the end of one line to the start of the next, which may be the exact same caret position.

- bool GetCaretPositionForIndex (long position, wxRect &rect, wxRichTextParagraphLayoutBox *container=NULL)

  Returns the caret height and position for the given character position.
- wxRichTextLine * GetVisibleLineForCaretPosition (long caretPosition) const

  Internal helper function returning the line for the visible caret position.
- wxCommandProcessor * GetCommandProcessor () const

  Gets the command processor associated with the control's buffer.
- bool DeleteSelectedContent (long *newPos=NULL)

  Deletes content if there is a selection, e.g.
- wxPoint GetPhysicalPoint (const wxPoint &ptLogical) const

  Transforms logical (unscrolled) position to physical window position.
- wxPoint GetLogicalPoint (const wxPoint &ptPhysical) const

  Transforms physical window position to logical (unscrolled) position.
- virtual long FindNextWordPosition (int direction=1) const

  Helper function for finding the caret position for the next word.
- bool IsPositionVisible (long pos) const

  Returns true if the given position is visible on the screen.
- long GetFirstVisiblePosition () const

  Returns the first visible position in the current view.
- void EnableImages (bool b)

  Enable or disable images.
- bool GetImagesEnabled () const

  Returns true if images are enabled.
- void EnableDelayedImageLoading (bool b)

  Enable or disable delayed image loading.
- bool GetDelayedImageLoading () const

  Returns true if delayed image loading is enabled.
- bool GetDelayedImageProcessingRequired () const

  Gets the flag indicating that delayed image processing is required.
- void SetDelayedImageProcessingRequired (bool b)

  Sets the flag indicating that delayed image processing is required.
- wxLongLong GetDelayedImageProcessingTime () const

  Returns the last time delayed image processing was performed.
- void SetDelayedImageProcessingTime (wxLongLong t)

  Sets the last time delayed image processing was performed.
- long GetCaretPositionForDefaultStyle () const

  Returns the caret position since the default formatting was changed.
- void SetCaretPositionForDefaultStyle (long pos)

  Set the caret position for the default style that the user is selecting.
- bool IsDefaultStyleShowing () const

  Returns true if the user has recently set the default style without moving the caret, and therefore the UI needs to reflect the default style and not the style at the caret.
- void SetAndShowDefaultStyle (const wxRichTextAttr &attr)

  Sets attr as the default style and tells the control that the UI should reflect this attribute until the user moves the caret.
- wxPoint GetFirstVisiblePoint () const

  Returns the first visible point in the window.
- virtual wxString GetValue () const

  Returns the content of the entire control as a string.
- virtual void SetValue (const wxString &value)

  Replaces existing content with the given text.
- virtual bool ProcessBackKey (wxKeyEvent &event, int flags)

Processes the back key.

- virtual wxRichTextRange FindRangeForList (long pos, bool &isNumberedList)

  Given a character position at which there is a list style, find the range encompassing the same list style by looking backwards and forwards.
- bool SetCaretPositionAfterClick (wxRichTextParagraphLayoutBox *container, long position, int hitTestFlags, bool extendSelection=false)

  Sets up the caret for the given position and container, after a mouse click.
- long FindCaretPositionForCharacterPosition (long position, int hitTestFlags, wxRichTextParagraphLayoutBox *container, bool &caretLineStart)

  Find the caret position for the combination of hit-test flags and character position.
- virtual bool ProcessMouseMovement (wxRichTextParagraphLayoutBox *container, wxRichTextObject *obj, long position, const wxPoint &pos)

  Processes mouse movement in order to change the cursor.
- virtual wxString DoGetValue () const
- bool ProcessDelayedImageLoading (bool refresh)

  Do delayed image loading and garbage-collect other images.
- bool ProcessDelayedImageLoading (const wxRect &screenRect, wxRichTextParagraphLayoutBox *box, int &loadCount)
- void RequestDelayedImageProcessing ()

  Request delayed image processing.
- void OnTimer (wxTimerEvent &event)

  Respond to timer events.

- virtual void GetSelection (long *from, long *to) const

  Returns the range of the current selection.
- const wxRichTextSelection & GetSelection () const

  Returns the range of the current selection.
- wxRichTextSelection & GetSelection ()

  Returns the range of the current selection.

- wxRichTextContextMenuPropertiesInfo & GetContextMenuPropertiesInfo ()

  Returns an object that stores information about context menu property item(s), in order to communicate between the context menu event handler and the code that responds to it.
- const wxRichTextContextMenuPropertiesInfo & GetContextMenuPropertiesInfo () const

  Returns an object that stores information about context menu property item(s), in order to communicate between the context menu event handler and the code that responds to it.

- virtual bool GetStyle (long position, wxTextAttr &style)

  Gets the attributes at the given position.
- virtual bool GetStyle (long position, wxRichTextAttr &style)

  Gets the attributes at the given position.
- virtual bool GetStyle (long position, wxRichTextAttr &style, wxRichTextParagraphLayoutBox *container)

  Gets the attributes at the given position.

- virtual bool SetStyle (long start, long end, const wxTextAttr &style)

  Sets the attributes for the given range.
- virtual bool SetStyle (long start, long end, const wxRichTextAttr &style)

  Sets the attributes for the given range.
- virtual bool SetStyle (const wxRichTextRange &range, const wxTextAttr &style)

  Sets the attributes for the given range.
- virtual bool SetStyle (const wxRichTextRange &range, const wxRichTextAttr &style)

Sets the attributes for the given range.

- virtual bool GetStyleForRange (const wxRichTextRange &range, wxTextAttr &style)

  Gets the attributes common to the specified range.
- virtual bool GetStyleForRange (const wxRichTextRange &range, wxRichTextAttr &style)

  Gets the attributes common to the specified range.
- virtual bool GetStyleForRange (const wxRichTextRange &range, wxRichTextAttr &style, wxRichTextParagraphLayoutBox *container)

  Gets the attributes common to the specified range.

- virtual bool GetUncombinedStyle (long position, wxRichTextAttr &style)

  Gets the attributes at the given position.
- virtual bool GetUncombinedStyle (long position, wxRichTextAttr &style, wxRichTextParagraphLayoutBox *container)

  Gets the attributes at the given position.

- virtual bool SetDefaultStyle (const wxTextAttr &style)

  Sets the current default style, which can be used to change how subsequently inserted text is displayed.
- virtual bool SetDefaultStyle (const wxRichTextAttr &style)

  Sets the current default style, which can be used to change how subsequently inserted text is displayed.

- virtual bool SetListStyle (const wxRichTextRange &range, wxRichTextListStyleDefinition *def, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Sets the list attributes for the given range, passing flags to determine how the attributes are set.
- virtual bool SetListStyle (const wxRichTextRange &range, const wxString &defName, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Sets the list attributes for the given range, passing flags to determine how the attributes are set.

- virtual bool NumberList (const wxRichTextRange &range, wxRichTextListStyleDefinition *def=NULL, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Numbers the paragraphs in the given range.
- virtual bool NumberList (const wxRichTextRange &range, const wxString &defName, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Numbers the paragraphs in the given range.

- virtual bool PromoteList (int promoteBy, const wxRichTextRange &range, wxRichTextListStyleDefinition *def=NULL, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int specifiedLevel=-1)

  Promotes or demotes the paragraphs in the given range.
- virtual bool PromoteList (int promoteBy, const wxRichTextRange &range, const wxString &defName, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int specifiedLevel=-1)

  Promotes or demotes the paragraphs in the given range.

- virtual wxTextCtrlHitTestResult HitTest (const wxPoint &pt, long *pos) const

  Finds the character at the given position in pixels.
- virtual wxTextCtrlHitTestResult HitTest (const wxPoint &pt, wxTextCoord *col, wxTextCoord *row) const

  Finds the character at the given position in pixels.
- wxRichTextParagraphLayoutBox * FindContainerAtPoint (const wxPoint pt, long &position, int &hit, wxRichTextObject *hitObj, int flags=0)

  Finds the container at the given point, which is assumed to be in client coordinates.

- virtual void SetSelection (long from, long to)

  Sets the selection to the given range.

- void SetSelection (const wxRichTextSelection &sel)

  Sets the selection to the given range.

- virtual bool WriteImage (const wxImage &image, wxBitmapType bitmapType=wxBITMAP_TYPE_PNG, const wxRichTextAttr &textAttr=wxRichTextAttr())

  Write a bitmap or image at the current insertion point.
- virtual bool WriteImage (const wxBitmap &bitmap, wxBitmapType bitmapType=wxBITMAP_TYPE_PNG, const wxRichTextAttr &textAttr=wxRichTextAttr())

  Write a bitmap or image at the current insertion point.

- wxRichTextBuffer & GetBuffer ()

  Returns the buffer associated with the control.
- const wxRichTextBuffer & GetBuffer () const

  Returns the buffer associated with the control.

Static Public Member Functions

- static const wxArrayString & GetAvailableFontNames ()

  Font names take a long time to retrieve, so cache them (on demand).
- static void ClearAvailableFontNames ()

  Clears the cache of available font names.

Protected Member Functions

- virtual wxWindow * GetEditableWindow ()
- virtual bool DoSetMargins (const wxPoint &pt)
- virtual wxPoint DoGetMargins () const
- virtual wxSize DoGetBestSize () const

  Currently this simply returns `wxSize(10, 10)`.
- virtual void DoSetValue (const wxString &value, int flags=0)
- virtual void DoThaw ()

Protected Attributes

- wxRichTextBuffer m_buffer

  Text buffer.
- wxMenu * m_contextMenu
- long m_caretPosition

  Caret position (1 less than the character position, so -1 is the first caret position).
- long m_caretPositionForDefaultStyle

  Caret position when the default formatting has been changed.
- wxRichTextSelection m_selection

  Selection range in character positions. -2, -2 means no selection.
- wxRichTextCtrlSelectionState m_selectionState
- long m_selectionAnchor

  Anchor so we know how to extend the selection It's a caret position since it's between two characters.
- wxRichTextObject * m_selectionAnchorObject

  Anchor object if selecting multiple container objects, such as grid cells.
- bool m_editable

  Are we editable?
- bool m_caretAtLineStart

Are we showing the caret position at the start of a line instead of at the end of the previous one?

- bool m_dragging

  Are we dragging a selection?
- bool m_fullLayoutRequired

  Do we need full layout in idle?
- wxLongLong m_fullLayoutTime
- long m_fullLayoutSavedPosition
- long m_delayedLayoutThreshold

  Threshold for doing delayed layout.
- wxCursor m_textCursor

  Cursors.
- wxCursor m_urlCursor
- wxRichTextContextMenuPropertiesInfo m_contextMenuPropertiesInfo
- wxRichTextParagraphLayoutBox * m_focusObject

  The object that currently has the editing focus.
- double m_scale

  An overall scale factor.
- wxSize m_lastWindowSize

  Variables for scrollbar hysteresis detection.
- int m_setupScrollbarsCount
- int m_setupScrollbarsCountInOnSize
- bool m_enableImages

  Whether images are enabled for this control.
- bool m_enableDelayedImageLoading

  Whether delayed image loading is enabled for this control.
- bool m_delayedImageProcessingRequired
- wxLongLong m_delayedImageProcessingTime
- wxTimer m_delayedImageProcessingTimer

Static Protected Attributes

- static wxArrayString sm_availableFontNames

21.617.2 Constructor & Destructor Documentation

wxRichTextCtrl::wxRichTextCtrl ( )

Default constructor.

wxRichTextCtrl::wxRichTextCtrl (wxWindow * *parent*, wxWindowID *id* = -1, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxRE_MULTILINE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxTextCtrlNameStr)

Constructor, creating and showing a rich text control.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.

value	Default string.
pos	Window position.
size	Window size.
style	Window style.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxRichTextCtrl::~wxRichTextCtrl ( ) [virtual]

Destructor.

21.617.3 Member Function Documentation

virtual wxRichTextRange wxRichTextCtrl::AddImage (const wxImage & *image*) [virtual]

Adds an image to the control's buffer.

virtual wxRichTextRange wxRichTextCtrl::AddParagraph (const wxString & *text*) [virtual]

Adds a new paragraph of text to the end of the buffer.

virtual void wxRichTextCtrl::AppendText (const wxString & *text*) [virtual]

Sets the insertion point to the end of the buffer and writes the text.

virtual bool wxRichTextCtrl::ApplyAlignmentToSelection (wxTextAttrAlignment *alignment*) [virtual]

Applies the given alignment to the selection or the default style (undoable).

For alignment values, see wxTextAttr.

virtual bool wxRichTextCtrl::ApplyBoldToSelection ( ) [virtual]

Apples bold to the selection or the default style (undoable).

virtual bool wxRichTextCtrl::ApplyItalicToSelection ( ) [virtual]

Applies italic to the selection or the default style (undoable).

virtual bool wxRichTextCtrl::ApplyStyle (wxRichTextStyleDefinition * *def*) [virtual]

Applies the style sheet to the buffer, matching paragraph styles in the sheet against named styles in the buffer.

This might be useful if the styles have changed. If *sheet* is NULL, the sheet set with SetStyleSheet() is used. Currently this applies paragraph styles only.

bool wxRichTextCtrl::ApplyStyleSheet (wxRichTextStyleSheet * *styleSheet* = NULL)

Applies the style sheet to the buffer, for example if the styles have changed.

virtual bool wxRichTextCtrl::ApplyTextEffectToSelection (int *flags*) [virtual]

Applies one or more wxTextAttrEffects flags to the selection (undoable).

If there is no selection, it is applied to the default style.

virtual bool wxRichTextCtrl::ApplyUnderlineToSelection ( ) [virtual]

Applies underline to the selection or the default style (undoable).

virtual bool wxRichTextCtrl::BatchingUndo ( ) const [virtual]

Returns true if undo commands are being batched.

bool wxRichTextCtrl::BeginAlignment (wxTextAttrAlignment *alignment*)

Begins using alignment.

For alignment values, see wxTextAttr.

virtual bool wxRichTextCtrl::BeginBatchUndo (const wxString & *cmdName*) [virtual]

Starts batching undo history for commands.

bool wxRichTextCtrl::BeginBold ( )

Begins using bold.

bool wxRichTextCtrl::BeginCharacterStyle (const wxString & *characterStyle*)

Begins using the named character style.

bool wxRichTextCtrl::BeginFont (const wxFont & *font*)

Begins using this font.

bool wxRichTextCtrl::BeginFontSize (int *pointSize*)

Begins using the given point size.

bool wxRichTextCtrl::BeginItalic ( )

Begins using italic.

bool wxRichTextCtrl::BeginLeftIndent (int *leftIndent,* int *leftSubIndent* = 0)

Begins applying a left indent and subindent in tenths of a millimetre.

The subindent is an offset from the left edge of the paragraph, and is used for all but the first line in a paragraph. A positive value will cause the first line to appear to the left of the subsequent lines, and a negative value will cause the first line to be indented to the right of the subsequent lines.

wxRichTextBuffer uses indentation to render a bulleted item. The content of the paragraph, including the first line, starts at the *leftIndent* plus the *leftSubIndent*.

Parameters

leftIndent	The distance between the margin and the bullet.
leftSubIndent	The distance between the left edge of the bullet and the left edge of the actual paragraph.

bool wxRichTextCtrl::BeginLineSpacing (int *lineSpacing*)

Begins appling line spacing.

spacing is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing.

The wxTextAttrLineSpacing constants are defined for convenience.

bool wxRichTextCtrl::BeginListStyle (const wxString & *listStyle,* int *level* = 1, int *number* = 1)

Begins using a specified list style.

Optionally, you can also pass a level and a number.

bool wxRichTextCtrl::BeginNumberedBullet (int *bulletNumber,* int *leftIndent,* int *leftSubIndent,* int *bulletStyle* = wxTEXT_ATTR_BULLET_STYLE_ARABIC|wxTEXT_ATTR_BULLET_STYLE_PERIOD)

Begins a numbered bullet.

This call will be needed for each item in the list, and the application should take care of incrementing the numbering.

bulletNumber is a number, usually starting with 1. *leftIndent* and *leftSubIndent* are values in tenths of a millimetre. *bulletStyle* is a bitlist of the wxTextAttrBulletStyle values.

wxRichTextBuffer uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at leftMargin + leftSubIndent. So the distance between the left edge of the bullet and the left of the actual paragraph is leftSubIndent.

bool wxRichTextCtrl::BeginParagraphSpacing (int *before,* int *after*)

Begins paragraph spacing; pass the before-paragraph and after-paragraph spacing in tenths of a millimetre.

bool wxRichTextCtrl::BeginParagraphStyle (const wxString & *paragraphStyle*)

Begins applying the named paragraph style.

bool wxRichTextCtrl::BeginRightIndent (int *rightIndent*)

Begins a right indent, specified in tenths of a millimetre.

bool wxRichTextCtrl::BeginStandardBullet (const wxString & *bulletName,* int *leftIndent,* int *leftSubIndent,* int *bulletStyle =* wxTEXT_ATTR_BULLET_STYLE_STANDARD)

Begins applying a symbol bullet.

virtual bool wxRichTextCtrl::BeginStyle (const wxRichTextAttr & *style*) `[virtual]`

Begins applying a style.

virtual bool wxRichTextCtrl::BeginSuppressUndo ( ) `[virtual]`

Starts suppressing undo history for commands.

bool wxRichTextCtrl::BeginSymbolBullet (const wxString & *symbol,* int *leftIndent,* int *leftSubIndent,* int *bulletStyle =* wxTEXT_ATTR_BULLET_STYLE_SYMBOL)

Begins applying a symbol bullet, using a character from the current font.

See BeginNumberedBullet() for an explanation of how indentation is used to render the bulleted paragraph.

bool wxRichTextCtrl::BeginTextColour (const wxColour & *colour*)

Begins using this colour.

bool wxRichTextCtrl::BeginUnderline ( )

Begins using underlining.

bool wxRichTextCtrl::BeginURL (const wxString & *url,* const wxString & *characterStyle =* wxEmptyString)

Begins applying wxTEXT_ATTR_URL to the content.

Pass a URL and optionally, a character style to apply, since it is common to mark a URL with a familiar style such as blue text with underlining.

virtual bool wxRichTextCtrl::CanCopy ( ) const `[virtual]`

Returns true if selected content can be copied to the clipboard.

virtual bool wxRichTextCtrl::CanCut ( ) const `[virtual]`

Returns true if selected content can be copied to the clipboard and deleted.

virtual bool wxRichTextCtrl::CanDeleteRange (wxRichTextParagraphLayoutBox & *container,* const wxRichTextRange & *range*) const `[virtual]`

Can we delete this range? Sends an event to the control.

virtual bool wxRichTextCtrl::CanDeleteSelection ( ) const `[virtual]`

Returns true if selected content can be deleted.

virtual bool wxRichTextCtrl::CanEditProperties (wxRichTextObject * *obj*) const [virtual]

Returns true if we can edit the object's properties via a GUI.

virtual bool wxRichTextCtrl::CanInsertContent (wxRichTextParagraphLayoutBox & *container,* long *pos*) const [virtual]

Can we insert content at this position? Sends an event to the control.

virtual bool wxRichTextCtrl::CanPaste ( ) const [virtual]

Returns true if the clipboard content can be pasted to the buffer.

virtual bool wxRichTextCtrl::CanRedo ( ) const [virtual]

Returns true if there is a command in the command history that can be redone.

virtual bool wxRichTextCtrl::CanUndo ( ) const [virtual]

Returns true if there is a command in the command history that can be undone.

virtual void wxRichTextCtrl::Clear ( ) [virtual]

Clears the buffer content, leaving a single empty paragraph.

Cannot be undone.

static void wxRichTextCtrl::ClearAvailableFontNames ( ) [static]

Clears the cache of available font names.

virtual bool wxRichTextCtrl::ClearListStyle (const wxRichTextRange & *range,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO) [virtual]

Clears the list style from the given range, clearing list-related attributes and applying any named paragraph style associated with each paragraph.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.

See also

SetListStyle(), PromoteList(), NumberList().

void wxRichTextCtrl::Command (wxCommandEvent & *event*) [virtual]

Sends the event to the control.

Reimplemented from wxControl.

virtual void wxRichTextCtrl::Copy ( ) [virtual]

Copies the selected content (if any) to the clipboard.

bool wxRichTextCtrl::Create (wxWindow * *parent*, wxWindowID *id* = -1, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxRE_MULTILINE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxTextCtrlNameStr)

Creates the underlying window.

virtual void wxRichTextCtrl::Cut ( ) [virtual]

Copies the selected content (if any) to the clipboard and deletes the selection.

This is undoable.

virtual bool wxRichTextCtrl::Delete (const wxRichTextRange & *range*) [virtual]

Deletes the content within the given range.

bool wxRichTextCtrl::DeleteSelectedContent (long * *newPos* = NULL)

Deletes content if there is a selection, e.g.

when pressing a key. Returns the new caret position in *newPos*, or leaves it if there was no action. This is undoable.

wxPerl Note: In wxPerl this method takes no arguments and returns a 2-element list (ok, newPos).

virtual void wxRichTextCtrl::DeleteSelection ( ) [virtual]

Deletes the content in the selection, if any.

This is undoable.

virtual void wxRichTextCtrl::DiscardEdits ( ) [virtual]

Sets the buffer's modified status to false, and clears the buffer's command history.

virtual bool wxRichTextCtrl::DoesSelectionHaveTextEffectFlag (int *flag*) [virtual]

Returns true if all of the selection, or the content at the current caret position, has the supplied wxTextAttrEffects flag(s).

virtual wxSize wxRichTextCtrl::DoGetBestSize ( ) const [protected], [virtual]

Currently this simply returns `wxSize(10, 10)`.

Reimplemented from wxWindow.

virtual wxPoint wxRichTextCtrl::DoGetMargins ( ) const [protected], [virtual]

virtual wxString wxRichTextCtrl::DoGetValue ( ) const [virtual]

virtual void wxRichTextCtrl::DoLayoutBuffer (**wxRichTextBuffer** & *buffer,* **wxDC** & *dc,* **wxRichTextDrawingContext** & *context,* const **wxRect** & *rect,* const **wxRect** & *parentRect,* int *flags*) [virtual]

Implements layout.

An application may override this to perform operations before or after layout.

virtual bool wxRichTextCtrl::DoLoadFile (const **wxString** & *file,* int *fileType*) [virtual]

Helper function for LoadFile().

Loads content into the control's buffer using the given type.

If the specified type is wxRICHTEXT_TYPE_ANY, the type is deduced from the filename extension.

This function looks for a suitable wxRichTextFileHandler object.

virtual bool wxRichTextCtrl::DoSaveFile (const **wxString** & *file* = **wxEmptyString,** int *fileType* = **wxRICHTEXT_TYPE_ANY**) [virtual]

Helper function for SaveFile().

Saves the buffer content using the given type.

If the specified type is wxRICHTEXT_TYPE_ANY, the type is deduced from the filename extension.

This function looks for a suitable wxRichTextFileHandler object.

virtual bool wxRichTextCtrl::DoSetMargins (const **wxPoint** & *pt*) [protected], [virtual]

virtual void wxRichTextCtrl::DoSetValue (const **wxString** & *value,* int *flags* = 0) [protected], [virtual]

virtual void wxRichTextCtrl::DoThaw ( ) [protected], [virtual]

virtual void wxRichTextCtrl::DoWriteText (const **wxString** & *value,* int *flags* = 0) [virtual]

virtual bool wxRichTextCtrl::EditProperties (**wxRichTextObject** * *obj,* **wxWindow** * *parent*) [virtual]

Edits the object's properties via a GUI.

void wxRichTextCtrl::EnableDelayedImageLoading (bool *b*) [inline]

Enable or disable delayed image loading.

void wxRichTextCtrl::EnableImages (bool *b*)

Enable or disable images.

virtual void wxRichTextCtrl::EnableVerticalScrollbar (bool *enable*) [virtual]

Enable or disable the vertical scrollbar.

void wxRichTextCtrl::EnableVirtualAttributes (bool *b*)

Pass true to let the control use virtual attributes.

The default is false.

bool wxRichTextCtrl::EndAlignment ( )

Ends alignment.

virtual bool wxRichTextCtrl::EndAllStyles ( ) [virtual]

Ends application of all styles in the current style stack.

virtual bool wxRichTextCtrl::EndBatchUndo ( ) [virtual]

Ends batching undo command history.

bool wxRichTextCtrl::EndBold ( )

Ends using bold.

bool wxRichTextCtrl::EndCharacterStyle ( )

Ends application of a named character style.

bool wxRichTextCtrl::EndFont ( )

Ends using a font.

bool wxRichTextCtrl::EndFontSize ( )

Ends using a point size.

bool wxRichTextCtrl::EndItalic ( )

Ends using italic.

bool wxRichTextCtrl::EndLeftIndent ( )

Ends left indent.

bool wxRichTextCtrl::EndLineSpacing ( )

Ends line spacing.

bool wxRichTextCtrl::EndListStyle ( )

Ends using a specified list style.

bool wxRichTextCtrl::EndNumberedBullet ( )

Ends application of a numbered bullet.

bool wxRichTextCtrl::EndParagraphSpacing ( )

Ends paragraph spacing.

bool wxRichTextCtrl::EndParagraphStyle ( )

Ends application of a named paragraph style.

bool wxRichTextCtrl::EndRightIndent ( )

Ends right indent.

bool wxRichTextCtrl::EndStandardBullet ( )

Begins applying a standard bullet.

virtual bool wxRichTextCtrl::EndStyle ( ) `[virtual]`

Ends the current style.

virtual bool wxRichTextCtrl::EndSuppressUndo ( ) `[virtual]`

Ends suppressing undo command history.

bool wxRichTextCtrl::EndSymbolBullet ( )

Ends applying a symbol bullet.

bool wxRichTextCtrl::EndTextColour ( )

Ends applying a text colour.

bool wxRichTextCtrl::EndUnderline ( )

End applying underlining.

bool wxRichTextCtrl::EndURL ( )

Ends applying a URL.

virtual bool wxRichTextCtrl::ExtendCellSelection (wxRichTextTable * *table,* int *noRowSteps,* int *noColSteps*) `[virtual]`

Extends a table selection in the given direction.

virtual bool wxRichTextCtrl::ExtendSelection (long *oldPosition,* long *newPosition,* int *flags*) `[virtual]`

Helper function for extending the selection, returning true if the selection was changed.

Selections are in caret positions.

long wxRichTextCtrl::FindCaretPositionForCharacterPosition (long *position,* int *hitTestFlags,* wxRichTextParagraphLayoutBox * *container,* bool & *caretLineStart*)

Find the caret position for the combination of hit-test flags and character position.

Returns the caret position and also an indication of where to place the caret (caretLineStart) since this is ambiguous (same position used for end of line and start of next).

wxRichTextParagraphLayoutBox* wxRichTextCtrl::FindContainerAtPoint (const wxPoint *pt,* long & *position,* int & *hit,* wxRichTextObject * *hitObj,* int *flags* = 0)

Finds the container at the given point, which is assumed to be in client coordinates.

virtual long wxRichTextCtrl::FindNextWordPosition (int *direction* = 1) const [virtual]

Helper function for finding the caret position for the next word.

Direction is 1 (forward) or -1 (backwards).

virtual wxRichTextRange wxRichTextCtrl::FindRangeForList (long *pos,* bool & *isNumberedList*) [virtual]

Given a character position at which there is a list style, find the range encompassing the same list style by looking backwards and forwards.

void wxRichTextCtrl::ForceDelayedLayout ( )

long wxRichTextCtrl::GetAdjustedCaretPosition (long *caretPos*) const

The adjusted caret position is the character position adjusted to take into account whether we're at the start of a paragraph, in which case style information should be taken from the next position, not current one.

static const wxArrayString& wxRichTextCtrl::GetAvailableFontNames ( ) [static]

Font names take a long time to retrieve, so cache them (on demand).

virtual const wxRichTextAttr& wxRichTextCtrl::GetBasicStyle ( ) const [virtual]

Gets the basic (overall) style.

This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

wxRichTextBuffer& wxRichTextCtrl::GetBuffer ( )

Returns the buffer associated with the control.

const wxRichTextBuffer& wxRichTextCtrl::GetBuffer ( ) const

Returns the buffer associated with the control.

bool wxRichTextCtrl::GetCaretAtLineStart ( ) const

Returns true if we are showing the caret position at the start of a line instead of at the end of the previous one.

long wxRichTextCtrl::GetCaretPosition ( ) const

Returns the current caret position.

long wxRichTextCtrl::GetCaretPositionForDefaultStyle ( ) const

Returns the caret position since the default formatting was changed.

As soon as this position changes, we no longer reflect the default style in the UI. A value of -2 means that we should only reflect the style of the content under the caret.

bool wxRichTextCtrl::GetCaretPositionForIndex (long *position,* wxRect & *rect,* wxRichTextParagraphLayoutBox * *container* = NULL)

Returns the caret height and position for the given character position.

If container is null, the current focus object will be used.

wxPerl Note: In wxPerl this method is implemented as GetCaretPositionForIndex(*position*) returning a 2-element list (ok, rect).

wxCommandProcessor* wxRichTextCtrl::GetCommandProcessor ( ) const

Gets the command processor associated with the control's buffer.

wxMenu* wxRichTextCtrl::GetContextMenu ( ) const

Returns the current context menu.

wxRichTextContextMenuPropertiesInfo& wxRichTextCtrl::GetContextMenuPropertiesInfo ( )

Returns an object that stores information about context menu property item(s), in order to communicate between the context menu event handler and the code that responds to it.

The wxRichTextContextMenuPropertiesInfo stores one item for each object that could respond to a property-editing event. If objects are nested, several might be editable.

const wxRichTextContextMenuPropertiesInfo& wxRichTextCtrl::GetContextMenuPropertiesInfo ( ) const

Returns an object that stores information about context menu property item(s), in order to communicate between the context menu event handler and the code that responds to it.

The wxRichTextContextMenuPropertiesInfo stores one item for each object that could respond to a property-editing event. If objects are nested, several might be editable.

virtual const wxRichTextAttr& wxRichTextCtrl::GetDefaultStyleEx ( ) const `[virtual]`

Returns the current default style, which can be used to change how subsequently inserted text is displayed.

bool wxRichTextCtrl::GetDelayedImageLoading ( ) const `[inline]`

Returns true if delayed image loading is enabled.

bool wxRichTextCtrl::GetDelayedImageProcessingRequired ( ) const `[inline]`

Gets the flag indicating that delayed image processing is required.

wxLongLong wxRichTextCtrl::GetDelayedImageProcessingTime ( ) const `[inline]`

Returns the last time delayed image processing was performed.

long wxRichTextCtrl::GetDelayedLayoutThreshold ( ) const

Gets the size of the buffer beyond which layout is delayed during resizing.
This optimizes sizing for large buffers. The default is 20000.

double wxRichTextCtrl::GetDimensionScale ( ) const `[inline]`

Returns the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.

bool wxRichTextCtrl::GetDragging ( ) const

Returns true if we are extending a selection.

const wxPoint wxRichTextCtrl::GetDragStartPoint ( ) const

Get the possible Drag'n'Drop start point.

const wxDateTime wxRichTextCtrl::GetDragStartTime ( ) const

Get the possible Drag'n'Drop start time.

virtual wxWindow* wxRichTextCtrl::GetEditableWindow ( ) `[protected], [virtual]`

wxString wxRichTextCtrl::GetFilename ( ) const

Gets the current filename associated with the control.

wxPoint wxRichTextCtrl::GetFirstVisiblePoint ( ) const

Returns the first visible point in the window.

long wxRichTextCtrl::GetFirstVisiblePosition ( ) const

Returns the first visible position in the current view.

wxRichTextParagraphLayoutBox* wxRichTextCtrl::GetFocusObject ( ) const

Returns the wxRichTextObject object that currently has the editing focus.

If there are no composite objects, this will be the top-level buffer.

double wxRichTextCtrl::GetFontScale ( ) const `[inline]`

Returns the scale factor for displaying fonts, for example for more comfortable editing.

bool wxRichTextCtrl::GetFullLayoutRequired ( ) const

long wxRichTextCtrl::GetFullLayoutSavedPosition ( ) const

wxLongLong wxRichTextCtrl::GetFullLayoutTime ( ) const

int wxRichTextCtrl::GetHandlerFlags ( ) const

Returns flags that change the behaviour of loading or saving.

See the documentation for each handler class to see what flags are relevant for each handler.

bool wxRichTextCtrl::GetImagesEnabled ( ) const

Returns true if images are enabled.

virtual long wxRichTextCtrl::GetInsertionPoint ( ) const `[virtual]`

Returns the current insertion point.

wxRichTextRange wxRichTextCtrl::GetInternalSelectionRange ( ) const

Returns the selection range in character positions.

-2, -2 means no selection -1, -1 means select everything. The range is in internal format, i.e. a single character selection is denoted by (n, n)

virtual wxTextPos wxRichTextCtrl::GetLastPosition ( ) const `[virtual]`

Returns the last position in the buffer.

virtual int wxRichTextCtrl::GetLineLength (long *lineNo*) const `[virtual]`

Returns the length of the specified line in characters.

virtual wxString wxRichTextCtrl::GetLineText (long *lineNo*) const `[virtual]`

Returns the text for the given line.

wxPoint wxRichTextCtrl::GetLogicalPoint (const wxPoint & *ptPhysical*) const

Transforms physical window position to logical (unscrolled) position.

virtual int wxRichTextCtrl::GetNumberOfLines ( ) const [virtual]

Returns the number of lines in the buffer.

wxPoint wxRichTextCtrl::GetPhysicalPoint (const wxPoint & *ptLogical*) const

Transforms logical (unscrolled) position to physical window position.

bool wxRichTextCtrl::GetPreDrag ( ) const

Are we trying to start Drag'n'Drop?

virtual wxString wxRichTextCtrl::GetPropertiesMenuLabel (wxRichTextObject * *obj*) [virtual]

Gets the object's properties menu label.

virtual wxString wxRichTextCtrl::GetRange (long *from,* long *to*) const [virtual]

Gets the text for the given range.

The end point of range is specified as the last character position of the span of text, plus one.

double wxRichTextCtrl::GetScale ( ) const

Returns an overall scale factor for displaying and editing the content.

wxPoint wxRichTextCtrl::GetScaledPoint (const wxPoint & *pt*) const

Returns a scaled point.

wxRect wxRichTextCtrl::GetScaledRect (const wxRect & *rect*) const

Returns a scaled rectangle.

wxSize wxRichTextCtrl::GetScaledSize (const wxSize & *sz*) const

Returns a scaled size.

virtual void wxRichTextCtrl::GetSelection (long * *from,* long * *to*) const [virtual]

Returns the range of the current selection.

The end point of range is specified as the last character position of the span of text, plus one. If the return values *from* and *to* are the same, there is no selection.

const wxRichTextSelection& wxRichTextCtrl::GetSelection ( ) const

Returns the range of the current selection.

The end point of range is specified as the last character position of the span of text, plus one. If the return values *from* and *to* are the same, there is no selection.

wxRichTextSelection& wxRichTextCtrl::GetSelection ( )

Returns the range of the current selection.

The end point of range is specified as the last character position of the span of text, plus one. If the return values *from* and *to* are the same, there is no selection.

long wxRichTextCtrl::GetSelectionAnchor ( ) const

Returns an anchor so we know how to extend the selection.

It's a caret position since it's between two characters.

wxRichTextObject* wxRichTextCtrl::GetSelectionAnchorObject ( ) const

Returns the anchor object if selecting multiple containers.

wxRichTextRange wxRichTextCtrl::GetSelectionRange ( ) const

Returns the selection range in character positions.

-1, -1 means no selection.

The range is in API convention, i.e. a single character selection is denoted by (n, n+1)

virtual wxString wxRichTextCtrl::GetStringSelection ( ) const `[virtual]`

Returns the text within the current selection range, if any.

virtual bool wxRichTextCtrl::GetStyle (long *position,* wxTextAttr & *style*) `[virtual]`

Gets the attributes at the given position.

This function gets the combined style - that is, the style you see on the screen as a result of combining base style, paragraph style and character style attributes.

To get the character or paragraph style alone, use GetUncombinedStyle().

wxPerl Note: In wxPerl this method is implemented as GetStyle(*position*) returning a 2-element list (ok, attr).

virtual bool wxRichTextCtrl::GetStyle (long *position,* wxRichTextAttr & *style*) `[virtual]`

Gets the attributes at the given position.

This function gets the combined style - that is, the style you see on the screen as a result of combining base style, paragraph style and character style attributes.

To get the character or paragraph style alone, use GetUncombinedStyle().

wxPerl Note: In wxPerl this method is implemented as GetStyle(*position*) returning a 2-element list (ok, attr).

virtual bool wxRichTextCtrl::GetStyle (long *position,* wxRichTextAttr & *style,* wxRichTextParagraphLayoutBox * *container*) `[virtual]`

Gets the attributes at the given position.

This function gets the combined style - that is, the style you see on the screen as a result of combining base style, paragraph style and character style attributes.

To get the character or paragraph style alone, use GetUncombinedStyle().

wxPerl Note: In wxPerl this method is implemented as GetStyle(*position*) returning a 2-element list (ok, attr).

virtual bool wxRichTextCtrl::GetStyleForRange (const wxRichTextRange & *range,* wxTextAttr & *style*) `[virtual]`

Gets the attributes common to the specified range.

Attributes that differ in value within the range will not be included in *style* flags.

wxPerl Note: In wxPerl this method is implemented as GetStyleForRange(*position*) returning a 2-element list (ok, attr).

virtual bool wxRichTextCtrl::GetStyleForRange (const wxRichTextRange & *range,* wxRichTextAttr & *style*) `[virtual]`

Gets the attributes common to the specified range.

Attributes that differ in value within the range will not be included in *style* flags.

wxPerl Note: In wxPerl this method is implemented as GetStyleForRange(*position*) returning a 2-element list (ok, attr).

virtual bool wxRichTextCtrl::GetStyleForRange (const wxRichTextRange & *range,* wxRichTextAttr & *style,* wxRichTextParagraphLayoutBox * *container*) `[virtual]`

Gets the attributes common to the specified range.

Attributes that differ in value within the range will not be included in *style* flags.

wxPerl Note: In wxPerl this method is implemented as GetStyleForRange(*position*) returning a 2-element list (ok, attr).

wxRichTextStyleSheet* wxRichTextCtrl::GetStyleSheet ( ) const

Returns the style sheet associated with the control, if any.

A style sheet allows named character and paragraph styles to be applied.

wxCursor wxRichTextCtrl::GetTextCursor ( ) const

Returns the text (normal) cursor.

virtual bool wxRichTextCtrl::GetUncombinedStyle (long *position,* wxRichTextAttr & *style*) `[virtual]`

Gets the attributes at the given position.

This function gets the *uncombined* style - that is, the attributes associated with the paragraph or character content, and not necessarily the combined attributes you see on the screen. To get the combined attributes, use GetStyle().

If you specify (any) paragraph attribute in *style's* flags, this function will fetch the paragraph attributes. Otherwise, it will return the character attributes.

wxPerl Note: In wxPerl this method is implemented as GetUncombinedStyle(*position*) returning a 2-element list (ok, attr).

virtual bool wxRichTextCtrl::GetUncombinedStyle (long *position,* wxRichTextAttr & *style,* wxRichTextParagraphLayoutBox * *container*) [virtual]

Gets the attributes at the given position.

This function gets the *uncombined* style - that is, the attributes associated with the paragraph or character content, and not necessarily the combined attributes you see on the screen. To get the combined attributes, use GetStyle().

If you specify (any) paragraph attribute in *style's* flags, this function will fetch the paragraph attributes. Otherwise, it will return the character attributes.

wxPerl Note: In wxPerl this method is implemented as GetUncombinedStyle(*position*) returning a 2-element list (ok, attr).

wxPoint wxRichTextCtrl::GetUnscaledPoint (const wxPoint & *pt*) const

Returns an unscaled point.

wxRect wxRichTextCtrl::GetUnscaledRect (const wxRect & *rect*) const

Returns an unscaled rectangle.

wxSize wxRichTextCtrl::GetUnscaledSize (const wxSize & *sz*) const

Returns an unscaled size.

wxCursor wxRichTextCtrl::GetURLCursor ( ) const

Returns the cursor to be used over URLs.

virtual wxString wxRichTextCtrl::GetValue ( ) const [virtual]

Returns the content of the entire control as a string.

virtual bool wxRichTextCtrl::GetVerticalScrollbarEnabled ( ) const [virtual]

Returns true if the vertical scrollbar is enabled.

bool wxRichTextCtrl::GetVirtualAttributesEnabled ( ) const

Returns true if this control can use virtual attributes and virtual text.

The default is false.

wxRichTextLine* wxRichTextCtrl::GetVisibleLineForCaretPosition (long *caretPosition*) const

Internal helper function returning the line for the visible caret position.

If the caret is shown at the very end of the line, it means the next character is actually on the following line. So this function gets the line we're expecting to find if this is the case.

virtual bool wxRichTextCtrl::HasCharacterAttributes (const wxRichTextRange & *range,* const wxRichTextAttr & *style*) const [virtual]

Test if this whole range has character attributes of the specified kind.

If any of the attributes are different within the range, the test fails.

You can use this to implement, for example, bold button updating. *style* must have flags indicating which attributes are of interest.

virtual bool wxRichTextCtrl::HasParagraphAttributes (const wxRichTextRange & *range,* const wxRichTextAttr & *style*) const [virtual]

Test if this whole range has paragraph attributes of the specified kind.

If any of the attributes are different within the range, the test fails. You can use this to implement, for example, centering button updating. *style* must have flags indicating which attributes are of interest.

virtual bool wxRichTextCtrl::HasSelection ( ) const [virtual]

Returns true if there is a selection and the object containing the selection was the same as the current focus object.

virtual bool wxRichTextCtrl::HasUnfocusedSelection ( ) const [virtual]

Returns true if there was a selection, whether or not the current focus object is the same as the selection's container object.

virtual wxTextCtrlHitTestResult wxRichTextCtrl::HitTest (const wxPoint & *pt,* long * *pos*) const [virtual]

Finds the character at the given position in pixels.

pt is in device coords (not adjusted for the client area origin nor for scrolling).

virtual wxTextCtrlHitTestResult wxRichTextCtrl::HitTest (const wxPoint & *pt,* wxTextCoord * *col,* wxTextCoord * *row*) const [virtual]

Finds the character at the given position in pixels.

pt is in device coords (not adjusted for the client area origin nor for scrolling).

void wxRichTextCtrl::Init ( )

Initialises the members of the control.

void wxRichTextCtrl::Invalidate ( )

Invalidates the whole buffer to trigger painting later.

bool wxRichTextCtrl::IsDefaultStyleShowing ( ) const

Returns true if the user has recently set the default style without moving the caret, and therefore the UI needs to reflect the default style and not the style at the caret.

Below is an example of code that uses this function to determine whether the UI should show that the current style is bold.

See also

SetAndShowDefaultStyle().

virtual bool wxRichTextCtrl::IsEditable ( ) const `[virtual]`

Returns true if the control is editable.

virtual bool wxRichTextCtrl::IsModified ( ) const `[virtual]`

Returns true if the buffer has been modified.

bool wxRichTextCtrl::IsMultiLine ( ) const

Returns true if the control is multiline.

bool wxRichTextCtrl::IsPositionVisible (long *pos*) const

Returns true if the given position is visible on the screen.

virtual bool wxRichTextCtrl::IsSelectionAligned (wxTextAttrAlignment *alignment*) `[virtual]`

Returns true if all of the selection is aligned according to the specified flag.

virtual bool wxRichTextCtrl::IsSelectionBold ( ) `[virtual]`

Returns true if all of the selection, or the content at the caret position, is bold.

virtual bool wxRichTextCtrl::IsSelectionItalics ( ) `[virtual]`

Returns true if all of the selection, or the content at the caret position, is italic.

virtual bool wxRichTextCtrl::IsSelectionUnderlined ( ) `[virtual]`

Returns true if all of the selection, or the content at the caret position, is underlined.

bool wxRichTextCtrl::IsSingleLine ( ) const

Returns true if the control is single-line.

Currently wxRichTextCtrl does not support single-line editing.

virtual bool wxRichTextCtrl::KeyboardNavigate (int *keyCode,* int *flags*) `[virtual]`

Helper function implementing keyboard navigation.

virtual bool wxRichTextCtrl::LayoutContent (bool *onlyVisibleRect* = `false`) `[virtual]`

Lays out the buffer, which must be done before certain operations, such as setting the caret position.

This function should not normally be required by the application.

virtual bool wxRichTextCtrl::LineBreak ( ) [virtual]

Inserts a line break at the current insertion point.

A line break forces wrapping within a paragraph, and can be introduced by using this function, by appending the wxChar value **wxRichTextLineBreakChar** to text content, or by typing Shift-Return.

bool wxRichTextCtrl::LoadFile (const wxString & *file*, int *type* = wxRICHTEXT_TYPE_ANY)

Loads content into the control's buffer using the given type.

If the specified type is wxRICHTEXT_TYPE_ANY, the type is deduced from the filename extension.

This function looks for a suitable wxRichTextFileHandler object.

virtual void wxRichTextCtrl::MarkDirty ( ) [virtual]

Marks the buffer as modified.

virtual bool wxRichTextCtrl::MoveCaret (long *pos*, bool *showAtLineStart* = false, wxRichTextParagraphLayoutBox * *container* = NULL) [virtual]

Move the caret to the given character position.

Please note that this does not update the current editing style from the new position; to do that, call wxRichTextCtrl::SetInsertionPoint instead.

void wxRichTextCtrl::MoveCaretBack (long *oldPosition*)

Move the caret one visual step forward: this may mean setting a flag and keeping the same position if we're going from the end of one line to the start of the next, which may be the exact same caret position.

void wxRichTextCtrl::MoveCaretForward (long *oldPosition*)

Move the caret one visual step forward: this may mean setting a flag and keeping the same position if we're going from the end of one line to the start of the next, which may be the exact same caret position.

virtual bool wxRichTextCtrl::MoveDown (int *noLines* = 1, int *flags* = 0) [virtual]

Moves the caret down.

virtual bool wxRichTextCtrl::MoveEnd (int *flags* = 0) [virtual]

Moves to the end of the buffer.

virtual bool wxRichTextCtrl::MoveHome (int *flags* = 0) [virtual]

Moves to the start of the buffer.

virtual bool wxRichTextCtrl::MoveLeft (int *noPositions* = 1, int *flags* = 0) [virtual]

Moves left.

virtual bool wxRichTextCtrl::MoveRight (int *noPositions* = 1, int *flags* = 0) [virtual]

Moves right.

virtual bool wxRichTextCtrl::MoveToLineEnd (int *flags* = 0) [virtual]

Moves to the end of the line.

virtual bool wxRichTextCtrl::MoveToLineStart (int *flags* = 0) [virtual]

Moves to the start of the line.

virtual bool wxRichTextCtrl::MoveToParagraphEnd (int *flags* = 0) [virtual]

Moves to the end of the paragraph.

virtual bool wxRichTextCtrl::MoveToParagraphStart (int *flags* = 0) [virtual]

Moves to the start of the paragraph.

virtual bool wxRichTextCtrl::MoveUp (int *noLines* = 1, int *flags* = 0) [virtual]

Moves to the start of the paragraph.

virtual bool wxRichTextCtrl::Newline ( ) [virtual]

Inserts a new paragraph at the current insertion point.

See also

LineBreak().

virtual bool wxRichTextCtrl::NumberList (const wxRichTextRange & *range*, wxRichTextListStyleDefinition * *def* = NULL, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Numbers the paragraphs in the given range.

Pass flags to determine how the attributes are set.

Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

SetListStyle(), PromoteList(), ClearListStyle().

virtual bool wxRichTextCtrl::NumberList (const wxRichTextRange & *range*, const wxString & *defName*, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Numbers the paragraphs in the given range.

Pass flags to determine how the attributes are set.

Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

SetListStyle(), PromoteList(), ClearListStyle().

void wxRichTextCtrl::OnCaptureLost (wxMouseCaptureLostEvent & *event*)

void wxRichTextCtrl::OnChar (wxKeyEvent & *event*)

void wxRichTextCtrl::OnClear (wxCommandEvent & *event*)

Standard handler for the wxID_CLEAR command.

void wxRichTextCtrl::OnContextMenu (wxContextMenuEvent & *event*)

Shows a standard context menu with undo, redo, cut, copy, paste, clear, and select all commands.

void wxRichTextCtrl::OnCopy (wxCommandEvent & *event*)

Standard handler for the wxID_COPY command.

void wxRichTextCtrl::OnCut (wxCommandEvent & *event*)

Standard handler for the wxID_CUT command.

void wxRichTextCtrl::OnDropFiles (wxDropFilesEvent & *event*)

Loads the first dropped file.

void wxRichTextCtrl::OnEraseBackground (wxEraseEvent & *event*)

void wxRichTextCtrl::OnIdle (wxIdleEvent & *event*)

void wxRichTextCtrl::OnKillFocus (wxFocusEvent & *event*)

void wxRichTextCtrl::OnLeftClick (wxMouseEvent & *event*)

void wxRichTextCtrl::OnLeftDClick (wxMouseEvent & *event*)

void wxRichTextCtrl::OnLeftUp (wxMouseEvent & *event*)

void wxRichTextCtrl::OnMiddleClick (wxMouseEvent & *event*)

void wxRichTextCtrl::OnMoveMouse (wxMouseEvent & *event*)

void wxRichTextCtrl::OnPaint (wxPaintEvent & *event*)

void wxRichTextCtrl::OnPaste (wxCommandEvent & *event*)

Standard handler for the wxID_PASTE command.

void wxRichTextCtrl::OnProperties (wxCommandEvent & *event*)

Standard handler for property commands.

void wxRichTextCtrl::OnRedo (wxCommandEvent & *event*)

Standard handler for the wxID_REDO command.

void wxRichTextCtrl::OnRightClick (wxMouseEvent & *event*)

void wxRichTextCtrl::OnScroll (wxScrollWinEvent & *event*)

void wxRichTextCtrl::OnSelectAll (wxCommandEvent & *event*)

Standard handler for the wxID_SELECTALL command.

void wxRichTextCtrl::OnSetFocus (wxFocusEvent & *event*)

void wxRichTextCtrl::OnSize (wxSizeEvent & *event*)

void wxRichTextCtrl::OnSysColourChanged (wxSysColourChangedEvent & *event*)

void wxRichTextCtrl::OnTimer (wxTimerEvent & *event*)

Respond to timer events.

void wxRichTextCtrl::OnUndo (wxCommandEvent & *event*)

Standard handler for the wxID_UNDO command.

void wxRichTextCtrl::OnUpdateClear (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_CLEAR command.

void wxRichTextCtrl::OnUpdateCopy (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_COPY command.

void wxRichTextCtrl::OnUpdateCut (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_CUT command.

void wxRichTextCtrl::OnUpdatePaste (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_PASTE command.

void wxRichTextCtrl::OnUpdateProperties (wxUpdateUIEvent & *event*)

Standard update handler for property commands.

void wxRichTextCtrl::OnUpdateRedo (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_REDO command.

void wxRichTextCtrl::OnUpdateSelectAll (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_SELECTALL command.

void wxRichTextCtrl::OnUpdateUndo (wxUpdateUIEvent & *event*)

Standard update handler for the wxID_UNDO command.

virtual bool wxRichTextCtrl::PageDown (int *noPages* = 1, int *flags* = 0) [virtual]

Moves one or more pages down.

virtual bool wxRichTextCtrl::PageUp (int *noPages* = 1, int *flags* = 0) [virtual]

Moves one or more pages up.

virtual void wxRichTextCtrl::PaintAboveContent (wxDC & *WXUNUSEDdc*) [inline], [virtual]

Other user defined painting after everything else (i.e. all text) is painted.

Since

2.9.1

virtual void wxRichTextCtrl::PaintBackground (wxDC & *dc*) [virtual]

Paints the background.

virtual void wxRichTextCtrl::Paste ( ) [virtual]

Pastes content from the clipboard to the buffer.

wxRichTextStyleSheet* wxRichTextCtrl::PopStyleSheet ( )

Pops the style sheet from top of stack.

virtual void wxRichTextCtrl::PositionCaret (wxRichTextParagraphLayoutBox * *container* = NULL) [virtual]

Internal function to position the visible caret according to the current caret position.

virtual bool wxRichTextCtrl::PositionToXY (long *pos*, long * *x*, long * *y*) const [virtual]

Converts a text position to zero-based column and line numbers.

virtual void wxRichTextCtrl::PrepareContent (wxRichTextParagraphLayoutBox & *container*) [virtual]

Prepares the content just before insertion (or after buffer reset).

Called by the same function in wxRichTextBuffer. Currently is only called if undo mode is on.

virtual int wxRichTextCtrl::PrepareContextMenu (wxMenu * *menu*, const wxPoint & *pt*, bool *addPropertyCommands*) [virtual]

Prepares the context menu, optionally adding appropriate property-editing commands.

Returns the number of property commands added.

virtual bool wxRichTextCtrl::ProcessBackKey (wxKeyEvent & *event*, int *flags*) [virtual]

Processes the back key.

bool wxRichTextCtrl::ProcessDelayedImageLoading (bool *refresh*)

Do delayed image loading and garbage-collect other images.

bool wxRichTextCtrl::ProcessDelayedImageLoading (const wxRect & *screenRect*, wxRichTextParagraphLayoutBox * *box*, int & *loadCount*)

virtual bool wxRichTextCtrl::ProcessMouseMovement (wxRichTextParagraphLayoutBox * *container*, wxRichTextObject * *obj*, long *position*, const wxPoint & *pos*) [virtual]

Processes mouse movement in order to change the cursor.

virtual bool wxRichTextCtrl::PromoteList (int *promoteBy*, const wxRichTextRange & *range*, wxRichTextList↩ StyleDefinition * *def* = NULL, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *specifiedLevel* = -1) [virtual]

Promotes or demotes the paragraphs in the given range.

A positive *promoteBy* produces a smaller indent, and a negative number produces a larger indent. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.

- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

SetListStyle(),
SetListStyle(), ClearListStyle().

virtual bool wxRichTextCtrl::PromoteList (int *promoteBy,* const wxRichTextRange & *range,* const wxString & *defName,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *specifiedLevel* = −1) [virtual]

Promotes or demotes the paragraphs in the given range.

A positive *promoteBy* produces a smaller indent, and a negative number produces a larger indent. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

SetListStyle(),
SetListStyle(), ClearListStyle().

bool wxRichTextCtrl::PushStyleSheet (wxRichTextStyleSheet * *styleSheet*)

Push the style sheet to top of stack.

virtual void wxRichTextCtrl::Redo ( ) [virtual]

Redoes the current command.

bool wxRichTextCtrl::RefreshForSelectionChange (const wxRichTextSelection & *oldSelection,* const wxRichTextSelection & *newSelection*)

Refreshes the area affected by a selection change.

virtual void wxRichTextCtrl::Remove (long *from,* long *to*) [virtual]

Removes the content in the specified range.

virtual void wxRichTextCtrl::Replace (long *from,* long *to,* const wxString & *value*) [virtual]

Replaces the content in the specified range with the string specified by *value*.

void wxRichTextCtrl::RequestDelayedImageProcessing ( )

Request delayed image processing.

bool wxRichTextCtrl::SaveFile (const wxString & *file* = wxEmptyString, int *type* = wxRICHTEXT_TYPE_ANY)

Saves the buffer content using the given type.

If the specified type is wxRICHTEXT_TYPE_ANY, the type is deduced from the filename extension.

This function looks for a suitable wxRichTextFileHandler object.

virtual bool wxRichTextCtrl::ScrollIntoView (long *position*, int *keyCode*) [virtual]

Scrolls *position* into view.

This function takes a caret position.

virtual void wxRichTextCtrl::SelectAll ( ) [virtual]

Selects all the text in the buffer.

virtual void wxRichTextCtrl::SelectNone ( ) [virtual]

Cancels any selection.

virtual bool wxRichTextCtrl::SelectWord (long *position*) [virtual]

Selects the word at the given character position.

void wxRichTextCtrl::SetAndShowDefaultStyle (const wxRichTextAttr & *attr*)

Sets *attr* as the default style and tells the control that the UI should reflect this attribute until the user moves the caret.

See also

IsDefaultStyleShowing().

virtual void wxRichTextCtrl::SetBasicStyle (const wxRichTextAttr & *style*) [virtual]

Sets the basic (overall) style.

This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

void wxRichTextCtrl::SetCaretAtLineStart (bool *atStart*)

Sets a flag to remember that we are showing the caret position at the start of a line instead of at the end of the previous one.

void wxRichTextCtrl::SetCaretPosition (long *position,* bool *showAtLineStart* = `false`)

Sets the caret position.

The caret position is the character position just before the caret. A value of -1 means the caret is at the start of the buffer. Please note that this does not update the current editing style from the new position or cause the actual caret to be refreshed; to do that, call wxRichTextCtrl::SetInsertionPoint instead.

bool wxRichTextCtrl::SetCaretPositionAfterClick (wxRichTextParagraphLayoutBox * *container,* long *position,* int *hitTestFlags,* bool *extendSelection* = `false`)

Sets up the caret for the given position and container, after a mouse click.

void wxRichTextCtrl::SetCaretPositionForDefaultStyle (long *pos*)

Set the caret position for the default style that the user is selecting.

void wxRichTextCtrl::SetContextMenu (wxMenu * *menu*)

Sets the current context menu.

virtual bool wxRichTextCtrl::SetDefaultStyle (const wxTextAttr & *style*) `[virtual]`

Sets the current default style, which can be used to change how subsequently inserted text is displayed.

virtual bool wxRichTextCtrl::SetDefaultStyle (const wxRichTextAttr & *style*) `[virtual]`

Sets the current default style, which can be used to change how subsequently inserted text is displayed.

bool wxRichTextCtrl::SetDefaultStyleToCursorStyle ( )

Sets the default style to the style under the cursor.

void wxRichTextCtrl::SetDelayedImageProcessingRequired (bool *b*) `[inline]`

Sets the flag indicating that delayed image processing is required.

void wxRichTextCtrl::SetDelayedImageProcessingTime (wxLongLong *t*) `[inline]`

Sets the last time delayed image processing was performed.

void wxRichTextCtrl::SetDelayedLayoutThreshold (long *threshold*)

Sets the size of the buffer beyond which layout is delayed during resizing.

This optimizes sizing for large buffers. The default is 20000.

void wxRichTextCtrl::SetDimensionScale (double *dimScale,* bool *refresh* = `false`)

Sets the scale factor for displaying certain dimensions such as indentation and inter-paragraph spacing.

This can be useful when editing in a small control where you still want legible text, but a minimum of wasted white space.

void wxRichTextCtrl::SetDragging (bool *dragging*)

Sets a flag to remember if we are extending a selection.

void wxRichTextCtrl::SetDragStartPoint (wxPoint *sp*)

Set the possible Drag'n'Drop start point.

void wxRichTextCtrl::SetDragStartTime (wxDateTime *st*)

Set the possible Drag'n'Drop start time.

virtual void wxRichTextCtrl::SetEditable (bool *editable*) `[virtual]`

Makes the control editable, or not.

void wxRichTextCtrl::SetFilename (const wxString & *filename*)

Sets the current filename.

bool wxRichTextCtrl::SetFocusObject (wxRichTextParagraphLayoutBox * *obj*, bool *setCaretPosition* = `true`)

Sets the wxRichTextObject object that currently has the editing focus.

Parameters

obj	The wxRichTextObject to set focus on.
setCaretPosition	Optionally set the caret position.

virtual bool wxRichTextCtrl::SetFont (const wxFont & *font*) `[virtual]`

Sets the font, and also the basic and default attributes (see wxRichTextCtrl::SetDefaultStyle).

Reimplemented from wxWindow.

void wxRichTextCtrl::SetFontScale (double *fontScale*, bool *refresh* = `false`)

Sets the scale factor for displaying fonts, for example for more comfortable editing.

void wxRichTextCtrl::SetFullLayoutRequired (bool *b*)

void wxRichTextCtrl::SetFullLayoutSavedPosition (long *p*)

void wxRichTextCtrl::SetFullLayoutTime (wxLongLong *t*)

void wxRichTextCtrl::SetHandlerFlags (int *flags*)

Sets flags that change the behaviour of loading or saving.

See the documentation for each handler class to see what flags are relevant for each handler.

virtual void wxRichTextCtrl::SetInsertionPoint (long *pos*) `[virtual]`

Sets the insertion point and causes the current editing style to be taken from the new position (unlike wxRichTextCtrl::SetCaretPosition).

virtual void wxRichTextCtrl::SetInsertionPointEnd ( ) `[virtual]`

Sets the insertion point to the end of the text control.

void wxRichTextCtrl::SetInternalSelectionRange (const wxRichTextRange & *range*)

Sets the selection range in character positions.

-2, -2 means no selection -1, -1 means select everything. The range is in internal format, i.e. a single character selection is denoted by (n, n)

virtual bool wxRichTextCtrl::SetListStyle (const wxRichTextRange & *range*, wxRichTextListStyleDefinition * *def*, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) `[virtual]`

Sets the list attributes for the given range, passing flags to determine how the attributes are set.

Either the style definition or the name of the style definition (in the current sheet) can be passed. *flags* is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

NumberList(), PromoteList(), ClearListStyle().

virtual bool wxRichTextCtrl::SetListStyle (const wxRichTextRange & *range*, const wxString & *defName*, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) `[virtual]`

Sets the list attributes for the given range, passing flags to determine how the attributes are set.

Either the style definition or the name of the style definition (in the current sheet) can be passed. *flags* is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

NumberList(), PromoteList(), ClearListStyle().

virtual void wxRichTextCtrl::SetMaxLength (unsigned long *len*) `[virtual]`

Sets the maximum number of characters that may be entered in a single line text control.

For compatibility only; currently does nothing.

void wxRichTextCtrl::SetModified (bool *modified*)

void wxRichTextCtrl::SetPreDrag (bool *pd*)

Set if we're trying to start Drag'n'Drop.

virtual bool wxRichTextCtrl::SetProperties (const wxRichTextRange & *range,* const wxRichTextProperties & *properties,* int *flags* = wxRICHTEXT_SETPROPERTIES_WITH_UNDO) `[virtual]`

Sets the properties for the given range, passing flags to determine how the attributes are set.

You can merge properties or replace them.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the properties for a character at position 5, use the range (5,6).

flags may contain a bit list of the following values:

- wxRICHTEXT_SETSPROPERTIES_NONE: no flag.
- wxRICHTEXT_SETPROPERTIES_WITH_UNDO: specifies that this operation should be undoable.
- wxRICHTEXT_SETPROPERTIES_PARAGRAPHS_ONLY: specifies that the properties should only be applied to paragraphs, and not the content.
- wxRICHTEXT_SETPROPERTIES_CHARACTERS_ONLY: specifies that the properties should only be applied to characters, and not the paragraph.
- wxRICHTEXT_SETPROPERTIES_RESET: resets (clears) the existing properties before applying the new properties.
- wxRICHTEXT_SETPROPERTIES_REMOVE: removes the specified properties.

void wxRichTextCtrl::SetScale (double *scale,* bool *refresh* = `false`)

Sets an overall scale factor for displaying and editing the content.

virtual void wxRichTextCtrl::SetSelection (long *from,* long *to*) `[virtual]`

Sets the selection to the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the selection for a character at position 5, use the range (5,6).

void wxRichTextCtrl::SetSelection (const wxRichTextSelection & *sel*)

Sets the selection to the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the selection for a character at position 5, use the range (5,6).

void wxRichTextCtrl::SetSelectionAnchor (long *anchor*)

Sets an anchor so we know how to extend the selection.

It's a caret position since it's between two characters.

void wxRichTextCtrl::SetSelectionAnchorObject (wxRichTextObject * *anchor*)

Sets the anchor object if selecting multiple containers.

void wxRichTextCtrl::SetSelectionRange (const wxRichTextRange & *range*)

Sets the selection to the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the selection for a character at position 5, use the range (5,6).

virtual bool wxRichTextCtrl::SetStyle (long *start,* long *end,* const wxTextAttr & *style*) [virtual]

Sets the attributes for the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the style for a character at position 5, use the range (5,6).

virtual bool wxRichTextCtrl::SetStyle (long *start,* long *end,* const wxRichTextAttr & *style*) [virtual]

Sets the attributes for the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the style for a character at position 5, use the range (5,6).

virtual bool wxRichTextCtrl::SetStyle (const wxRichTextRange & *range,* const wxTextAttr & *style*) [virtual]

Sets the attributes for the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the style for a character at position 5, use the range (5,6).

virtual bool wxRichTextCtrl::SetStyle (const wxRichTextRange & *range,* const wxRichTextAttr & *style*) [virtual]

Sets the attributes for the given range.

The end point of range is specified as the last character position of the span of text, plus one.

So, for example, to set the style for a character at position 5, use the range (5,6).

virtual void wxRichTextCtrl::SetStyle (wxRichTextObject * *obj,* const wxRichTextAttr & *textAttr,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO) [virtual]

Sets the attributes for a single object.

virtual bool wxRichTextCtrl::SetStyleEx (const wxRichTextRange & *range,* const wxRichTextAttr & *style,* int *flags =* wxRICHTEXT_SETSTYLE_WITH_UNDO) `[virtual]`

Sets the attributes for the given range, passing flags to determine how the attributes are set.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the style for a character at position 5, use the range (5,6).

flags may contain a bit list of the following values:

- wxRICHTEXT_SETSTYLE_NONE: no style flag.
- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this operation should be undoable.
- wxRICHTEXT_SETSTYLE_OPTIMIZE: specifies that the style should not be applied if the combined style at this point is already the style in question.
- wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY: specifies that the style should only be applied to paragraphs, and not the content. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY: specifies that the style should only be applied to characters, and not the paragraph. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- wxRICHTEXT_SETSTYLE_RESET: resets (clears) the existing style before applying the new style.
- wxRICHTEXT_SETSTYLE_REMOVE: removes the specified style. Only the style flags are used in this operation.

void wxRichTextCtrl::SetStyleSheet (wxRichTextStyleSheet * *styleSheet*)

Sets the style sheet associated with the control.

A style sheet allows named character and paragraph styles to be applied.

void wxRichTextCtrl::SetTextCursor (const wxCursor & *cursor*)

Sets the text (normal) cursor.

virtual void wxRichTextCtrl::SetupScrollbars (bool *atTop =* `false`) `[virtual]`

A helper function setting up scrollbars, for example after a resize.

void wxRichTextCtrl::SetURLCursor (const wxCursor & *cursor*)

Sets the cursor to be used over URLs.

virtual void wxRichTextCtrl::SetValue (const wxString & *value*) `[virtual]`

Replaces existing content with the given text.

virtual bool wxRichTextCtrl::ShouldInheritColours ( ) const [virtual]

Return true from here to allow the colours of this window to be changed by InheritAttributes().

Returning false forbids inheriting them from the parent window.

The base class version returns false, but this method is overridden in wxControl where it returns true.

Reimplemented from wxWindow.

virtual bool wxRichTextCtrl::ShowContextMenu (wxMenu * *menu*, const wxPoint & *pt*, bool *addPropertyCommands*) [virtual]

Shows the given context menu, optionally adding appropriate property-editing commands for the current position in the object hierarchy.

virtual void wxRichTextCtrl::ShowPosition (long *pos*) [virtual]

Scrolls the buffer so that the given position is in view.

virtual bool wxRichTextCtrl::StartCellSelection (wxRichTextTable * *table*, wxRichTextParagraphLayoutBox * *newCell*) [virtual]

Starts selecting table cells.

void wxRichTextCtrl::StoreFocusObject (wxRichTextParagraphLayoutBox * *obj*)

Setter for m_focusObject.

virtual bool wxRichTextCtrl::SuppressingUndo ( ) const [virtual]

Returns true if undo history suppression is on.

virtual void wxRichTextCtrl::Undo ( ) [virtual]

Undoes the command at the top of the command history, if there is one.

virtual bool wxRichTextCtrl::WordLeft (int *noPages* = 1, int *flags* = 0) [virtual]

Moves a number of words to the left.

virtual bool wxRichTextCtrl::WordRight (int *noPages* = 1, int *flags* = 0) [virtual]

Move a nuber of words to the right.

virtual wxRichTextField* wxRichTextCtrl::WriteField (const wxString & *fieldType*, const wxRichTextProperties & *properties*, const wxRichTextAttr & *textAttr* = wxRichTextAttr()) [virtual]

Writes a field at the current insertion point.

Parameters

fieldType	The field type, matching an existing field type definition.
properties	Extra data for the field.
textAttr	Optional attributes.

See also

wxRichTextField, wxRichTextFieldType, wxRichTextFieldTypeStandard

virtual bool wxRichTextCtrl::WriteImage (const wxImage & *image*, wxBitmapType *bitmapType* = wxBITMAP_TYPE_PNG, const wxRichTextAttr & *textAttr* = wxRichTextAttr()) [virtual]

Write a bitmap or image at the current insertion point.

Supply an optional type to use for internal and file storage of the raw data.

virtual bool wxRichTextCtrl::WriteImage (const wxBitmap & *bitmap*, wxBitmapType *bitmapType* = wxBITMAP_TYPE_PNG, const wxRichTextAttr & *textAttr* = wxRichTextAttr()) [virtual]

Write a bitmap or image at the current insertion point.

Supply an optional type to use for internal and file storage of the raw data.

virtual bool wxRichTextCtrl::WriteImage (const wxString & *filename*, wxBitmapType *bitmapType*, const wxRichTextAttr & *textAttr* = wxRichTextAttr()) [virtual]

Loads an image from a file and writes it at the current insertion point.

virtual bool wxRichTextCtrl::WriteImage (const wxRichTextImageBlock & *imageBlock*, const wxRichTextAttr & *textAttr* = wxRichTextAttr()) [virtual]

Writes an image block at the current insertion point.

virtual wxRichTextTable* wxRichTextCtrl::WriteTable (int *rows*, int *cols*, const wxRichTextAttr & *tableAttr* = wxRichTextAttr(), const wxRichTextAttr & *cellAttr* = wxRichTextAttr()) [virtual]

Write a table at the current insertion point, returning the table.

You can then call SetFocusObject() to set the focus to the new object.

virtual void wxRichTextCtrl::WriteText (const wxString & *text*) [virtual]

Writes text at the current position.

virtual wxRichTextBox* wxRichTextCtrl::WriteTextBox (const wxRichTextAttr & *textAttr* = wxRichTextAttr()) [virtual]

Write a text box at the current insertion point, returning the text box.

You can then call SetFocusObject() to set the focus to the new object.

virtual long wxRichTextCtrl::XYToPosition (long *x*, long *y*) const [virtual]

Translates from column and line number to position.

21.617.4 Member Data Documentation

wxRichTextBuffer wxRichTextCtrl::m_buffer [protected]

Text buffer.

bool wxRichTextCtrl::m_caretAtLineStart [protected]

Are we showing the caret position at the start of a line instead of at the end of the previous one?

long wxRichTextCtrl::m_caretPosition [protected]

Caret position (1 less than the character position, so -1 is the first caret position).

long wxRichTextCtrl::m_caretPositionForDefaultStyle [protected]

Caret position when the default formatting has been changed.

As soon as this position changes, we no longer reflect the default style in the UI.

wxMenu* wxRichTextCtrl::m_contextMenu [protected]

wxRichTextContextMenuPropertiesInfo wxRichTextCtrl::m_contextMenuPropertiesInfo [protected]

bool wxRichTextCtrl::m_delayedImageProcessingRequired [protected]

wxLongLong wxRichTextCtrl::m_delayedImageProcessingTime [protected]

wxTimer wxRichTextCtrl::m_delayedImageProcessingTimer [protected]

long wxRichTextCtrl::m_delayedLayoutThreshold [protected]

Threshold for doing delayed layout.

bool wxRichTextCtrl::m_dragging [protected]

Are we dragging a selection?

bool wxRichTextCtrl::m_editable [protected]

Are we editable?

bool wxRichTextCtrl::m_enableDelayedImageLoading [protected]

Whether delayed image loading is enabled for this control.

bool wxRichTextCtrl::m_enableImages [protected]

Whether images are enabled for this control.

wxRichTextParagraphLayoutBox* **wxRichTextCtrl::m_focusObject** [protected]

The object that currently has the editing focus.

bool wxRichTextCtrl::m_fullLayoutRequired [protected]

Do we need full layout in idle?

long wxRichTextCtrl::m_fullLayoutSavedPosition [protected]

wxLongLong wxRichTextCtrl::m_fullLayoutTime [protected]

wxSize wxRichTextCtrl::m_lastWindowSize [protected]

Variables for scrollbar hysteresis detection.

double wxRichTextCtrl::m_scale [protected]

An overall scale factor.

wxRichTextSelection wxRichTextCtrl::m_selection [protected]

Selection range in character positions. -2, -2 means no selection.

long wxRichTextCtrl::m_selectionAnchor [protected]

Anchor so we know how to extend the selection It's a caret position since it's between two characters.

wxRichTextObject* **wxRichTextCtrl::m_selectionAnchorObject** [protected]

Anchor object if selecting multiple container objects, such as grid cells.

wxRichTextCtrlSelectionState wxRichTextCtrl::m_selectionState [protected]

int wxRichTextCtrl::m_setupScrollbarsCount [protected]

int wxRichTextCtrl::m_setupScrollbarsCountInOnSize [protected]

wxCursor wxRichTextCtrl::m_textCursor [protected]

Cursors.

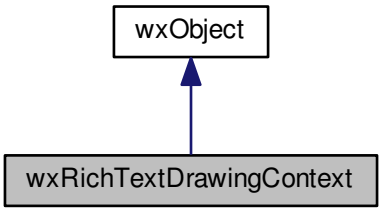
wxCursor wxRichTextCtrl::m_urlCursor [protected]

wxArrayString wxRichTextCtrl::sm_availableFontNames [static], [protected]

21.618 wxRichTextDrawingContext Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextDrawingContext:



21.618.1 Detailed Description

A class for passing information to drawing and measuring functions.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextDrawingContext (wxRichTextBuffer ∗buffer)

  Pass the buffer to the context so the context can retrieve information such as virtual attributes.
- void Init ()
- bool HasVirtualAttributes (wxRichTextObject ∗obj) const

  Does this object have virtual attributes? Virtual attributes can be provided for visual cues without affecting the actual styling.
- wxRichTextAttr GetVirtualAttributes (wxRichTextObject ∗obj) const

  Returns the virtual attributes for this object.
- bool ApplyVirtualAttributes (wxRichTextAttr &attr, wxRichTextObject ∗obj) const

  Applies any virtual attributes relevant to this object.
- int GetVirtualSubobjectAttributesCount (wxRichTextObject ∗obj) const

  Gets the count for mixed virtual attributes for individual positions within the object.
- int GetVirtualSubobjectAttributes (wxRichTextObject ∗obj, wxArrayInt &positions, wxRichTextAttrArray &attributes) const

  Gets the mixed virtual attributes for individual positions within the object.
- bool HasVirtualText (const wxRichTextPlainText ∗obj) const

  Do we have virtual text for this object? Virtual text allows an application to replace characters in an object for editing and display purposes, for example for highlighting special characters.
- bool GetVirtualText (const wxRichTextPlainText ∗obj, wxString &text) const

  Gets the virtual text for this object.
- void EnableVirtualAttributes (bool b)

Enables virtual attribute processing.

- bool GetVirtualAttributesEnabled () const

  Returns true if virtual attribute processing is enabled.

- void EnableImages (bool b)

  Enable or disable images.

- bool GetImagesEnabled () const

  Returns true if images are enabled.

- void SetLayingOut (bool b)

  Set laying out flag.

- bool GetLayingOut () const

  Returns true if laying out.

- void EnableDelayedImageLoading (bool b)

  Enable or disable delayed image loading.

- bool GetDelayedImageLoading () const

  Returns true if delayed image loading is enabled.

Public Attributes

- wxRichTextBuffer * m_buffer
- bool m_enableVirtualAttributes
- bool m_enableImages
- bool m_enableDelayedImageLoading
- bool m_layingOut

Additional Inherited Members

21.618.2 Constructor & Destructor Documentation

wxRichTextDrawingContext::wxRichTextDrawingContext (wxRichTextBuffer * *buffer*)

Pass the buffer to the context so the context can retrieve information such as virtual attributes.

21.618.3 Member Function Documentation

bool wxRichTextDrawingContext::ApplyVirtualAttributes (wxRichTextAttr & *attr,* wxRichTextObject * *obj*) const

Applies any virtual attributes relevant to this object.

void wxRichTextDrawingContext::EnableDelayedImageLoading (bool *b*) `[inline]`

Enable or disable delayed image loading.

void wxRichTextDrawingContext::EnableImages (bool *b*) `[inline]`

Enable or disable images.

void wxRichTextDrawingContext::EnableVirtualAttributes (bool *b*)

Enables virtual attribute processing.

bool wxRichTextDrawingContext::GetDelayedImageLoading ( ) const `[inline]`

Returns true if delayed image loading is enabled.

bool wxRichTextDrawingContext::GetImagesEnabled ( ) const `[inline]`

Returns true if images are enabled.

bool wxRichTextDrawingContext::GetLayingOut ( ) const `[inline]`

Returns true if laying out.

wxRichTextAttr wxRichTextDrawingContext::GetVirtualAttributes (wxRichTextObject * *obj*) const

Returns the virtual attributes for this object.

Virtual attributes can be provided for visual cues without affecting the actual styling.

bool wxRichTextDrawingContext::GetVirtualAttributesEnabled ( ) const

Returns true if virtual attribute processing is enabled.

int wxRichTextDrawingContext::GetVirtualSubobjectAttributes (wxRichTextObject * *obj*, wxArrayInt & *positions*, wxRichTextAttrArray & *attributes*) const

Gets the mixed virtual attributes for individual positions within the object.

For example, individual characters within a text object may require special highlighting. The function is passed the count returned by GetVirtualSubobjectAttributesCount.

int wxRichTextDrawingContext::GetVirtualSubobjectAttributesCount (wxRichTextObject * *obj*) const

Gets the count for mixed virtual attributes for individual positions within the object.

For example, individual characters within a text object may require special highlighting.

bool wxRichTextDrawingContext::GetVirtualText (const wxRichTextPlainText * *obj*, wxString & *text*) const

Gets the virtual text for this object.

bool wxRichTextDrawingContext::HasVirtualAttributes (wxRichTextObject * *obj*) const

Does this object have virtual attributes? Virtual attributes can be provided for visual cues without affecting the actual styling.

bool wxRichTextDrawingContext::HasVirtualText (const wxRichTextPlainText * *obj*) const

Do we have virtual text for this object? Virtual text allows an application to replace characters in an object for editing and display purposes, for example for highlighting special characters.

void wxRichTextDrawingContext::Init ( )

void wxRichTextDrawingContext::SetLayingOut (bool *b*) `[inline]`

Set laying out flag.

21.618.4 Member Data Documentation

wxRichTextBuffer* wxRichTextDrawingContext::m_buffer

bool wxRichTextDrawingContext::m_enableDelayedImageLoading

bool wxRichTextDrawingContext::m_enableImages

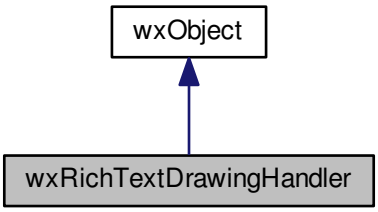
bool wxRichTextDrawingContext::m_enableVirtualAttributes

bool wxRichTextDrawingContext::m_layingOut

21.619 wxRichTextDrawingHandler Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextDrawingHandler:



21.619.1 Detailed Description

The base class for custom drawing handlers.

Currently, drawing handlers can provide virtual attributes.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextDrawingHandler (const wxString &name=wxEmptyString)

  Creates a drawing handler object.
- virtual bool HasVirtualAttributes (wxRichTextObject ∗obj) const =0

  Returns true if this object has virtual attributes that we can provide.
- virtual bool GetVirtualAttributes (wxRichTextAttr &attr, wxRichTextObject ∗obj) const =0

  Provides virtual attributes that we can provide.
- virtual int GetVirtualSubobjectAttributesCount (wxRichTextObject ∗obj) const =0

  Gets the count for mixed virtual attributes for individual positions within the object.
- virtual int GetVirtualSubobjectAttributes (wxRichTextObject ∗obj, wxArrayInt &positions, wxRichTextAttrArray &attributes) const =0

  Gets the mixed virtual attributes for individual positions within the object.
- virtual bool HasVirtualText (const wxRichTextPlainText ∗obj) const =0

  Do we have virtual text for this object? Virtual text allows an application to replace characters in an object for editing and display purposes, for example for highlighting special characters.
- virtual bool GetVirtualText (const wxRichTextPlainText ∗obj, wxString &text) const =0

  Gets the virtual text for this object.
- void SetName (const wxString &name)

  Sets the name of the handler.
- wxString GetName () const

  Returns the name of the handler.

Protected Attributes

- wxString m_name

Additional Inherited Members

21.619.2 Constructor & Destructor Documentation

wxRichTextDrawingHandler::wxRichTextDrawingHandler (const wxString & *name* = wxEmptyString) `[inline]`

Creates a drawing handler object.

21.619.3 Member Function Documentation

wxString wxRichTextDrawingHandler::GetName ( ) const `[inline]`

Returns the name of the handler.

virtual bool wxRichTextDrawingHandler::GetVirtualAttributes (wxRichTextAttr & *attr,* wxRichTextObject ∗ *obj*) const `[pure virtual]`

Provides virtual attributes that we can provide.

virtual int wxRichTextDrawingHandler::GetVirtualSubobjectAttributes (wxRichTextObject ∗ *obj,* wxArrayInt & *positions,* wxRichTextAttrArray & *attributes*) const `[pure virtual]`

Gets the mixed virtual attributes for individual positions within the object.

For example, individual characters within a text object may require special highlighting. Returns the number of virtual attributes found.

virtual int wxRichTextDrawingHandler::GetVirtualSubobjectAttributesCount (wxRichTextObject * *obj*) const [pure virtual]

Gets the count for mixed virtual attributes for individual positions within the object.

For example, individual characters within a text object may require special highlighting.

virtual bool wxRichTextDrawingHandler::GetVirtualText (const wxRichTextPlainText * *obj,* wxString & *text*) const [pure virtual]

Gets the virtual text for this object.

virtual bool wxRichTextDrawingHandler::HasVirtualAttributes (wxRichTextObject * *obj*) const [pure virtual]

Returns true if this object has virtual attributes that we can provide.

virtual bool wxRichTextDrawingHandler::HasVirtualText (const wxRichTextPlainText * *obj*) const [pure virtual]

Do we have virtual text for this object? Virtual text allows an application to replace characters in an object for editing and display purposes, for example for highlighting special characters.

void wxRichTextDrawingHandler::SetName (const wxString & *name*) [inline]

Sets the name of the handler.

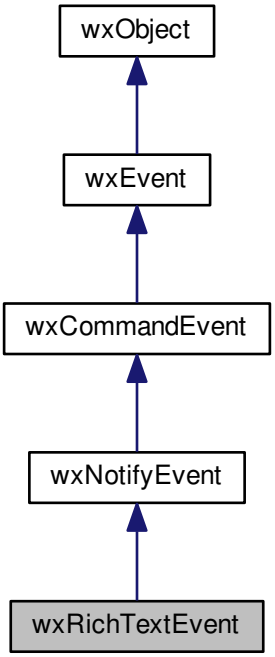
21.619.4 Member Data Documentation

wxString wxRichTextDrawingHandler::m_name [protected]

21.620 wxRichTextEvent Class Reference

```
#include <wx/richtext/richtextctrl.h>
```

Inheritance diagram for wxRichTextEvent:



21.620.1 Detailed Description

This is the event class for wxRichTextCtrl notifications.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxRichTextEvent& event)

Event macros:

- EVT_RICHTEXT_LEFT_CLICK(id, func): Process a `wxEVT_RICHTEXT_LEFT_CLICK` event, generated when the user releases the left mouse button over an object.
- EVT_RICHTEXT_RIGHT_CLICK(id, func): Process a `wxEVT_RICHTEXT_RIGHT_CLICK` event, generated when the user releases the right mouse button over an object.
- EVT_RICHTEXT_MIDDLE_CLICK(id, func): Process a `wxEVT_RICHTEXT_MIDDLE_CLICK` event, generated when the user releases the middle mouse button over an object.
- EVT_RICHTEXT_LEFT_DCLICK(id, func): Process a `wxEVT_RICHTEXT_LEFT_DCLICK` event, generated when the user double-clicks an object.
- EVT_RICHTEXT_RETURN(id, func): Process a `wxEVT_RICHTEXT_RETURN` event, generated when the user presses the return key. Valid event functions: GetFlags, GetPosition.
- EVT_RICHTEXT_CHARACTER(id, func): Process a `wxEVT_RICHTEXT_CHARACTER` event, generated when the user presses a character key. Valid event functions: GetFlags, GetPosition, GetCharacter.

- EVT_RICHTEXT_CONSUMING_CHARACTER(id, func): Process a wxEVT_RICHTEXT_CONSUMING_CHARACTER event, generated when the user presses a character key but before it is processed and inserted into the control. Call Veto to prevent normal processing. Valid event functions: GetFlags, GetPosition, GetCharacter, Veto.
- EVT_RICHTEXT_DELETE(id, func): Process a wxEVT_RICHTEXT_DELETE event, generated when the user presses the backspace or delete key. Valid event functions: GetFlags, GetPosition.
- EVT_RICHTEXT_RETURN(id, func): Process a wxEVT_RICHTEXT_RETURN event, generated when the user presses the return key. Valid event functions: GetFlags, GetPosition.
- EVT_RICHTEXT_STYLE_CHANGED(id, func): Process a wxEVT_RICHTEXT_STYLE_CHANGED event, generated when styling has been applied to the control. Valid event functions: GetPosition, GetRange.
- EVT_RICHTEXT_STYLESHEET_CHANGED(id, func): Process a wxEVT_RICHTEXT_STYLESHEET_CHANGING event, generated when the control's stylesheet has changed, for example the user added, edited or deleted a style. Valid event functions: GetRange, GetPosition.
- EVT_RICHTEXT_STYLESHEET_REPLACING(id, func): Process a wxEVT_RICHTEXT_STYLESHEET_REPLACING event, generated when the control's stylesheet is about to be replaced, for example when a file is loaded into the control. Valid event functions: Veto, GetOldStyleSheet, GetNewStyleSheet.
- EVT_RICHTEXT_STYLESHEET_REPLACED(id, func): Process a wxEVT_RICHTEXT_STYLESHEET_REPLACED event, generated when the control's stylesheet has been replaced, for example when a file is loaded into the control. Valid event functions: GetOldStyleSheet, GetNewStyleSheet.
- EVT_RICHTEXT_PROPERTIES_CHANGED(id, func): Process a wxEVT_RICHTEXT_PROPERTIES_CHANGED event, generated when properties have been applied to the control. Valid event functions: GetPosition, GetRange.
- EVT_RICHTEXT_CONTENT_INSERTED(id, func): Process a wxEVT_RICHTEXT_CONTENT_INSERTED event, generated when content has been inserted into the control. Valid event functions: GetPosition, GetRange.
- EVT_RICHTEXT_CONTENT_DELETED(id, func): Process a wxEVT_RICHTEXT_CONTENT_DELETED event, generated when content has been deleted from the control. Valid event functions: GetPosition, GetRange.
- EVT_RICHTEXT_BUFFER_RESET(id, func): Process a wxEVT_RICHTEXT_BUFFER_RESET event, generated when the buffer has been reset by deleting all content. You can use this to set a default style for the first new paragraph.
- EVT_RICHTEXT_SELECTION_CHANGED(id, func): Process a wxEVT_RICHTEXT_SELECTION_CHANGED event, generated when the selection range has changed.
- EVT_RICHTEXT_FOCUS_OBJECT_CHANGED(id, func): Process a wxEVT_RICHTEXT_FOCUS_OBJECT_CHANGED event, generated when the current focus object has changed.

Library: wxRichText

Category: Events, Rich Text

Public Member Functions

- wxRichTextEvent (wxEventType commandType=wxEVT_NULL, int winid=0)

  Constructor.
- wxRichTextEvent (const wxRichTextEvent &event)

  Copy constructor.
- long GetPosition () const

Returns the buffer position at which the event occurred.

- void SetPosition (long pos)

  Sets the buffer position variable.
- int GetFlags () const

  Returns flags indicating modifier keys pressed.
- void SetFlags (int flags)

  Sets flags indicating modifier keys pressed.
- wxRichTextStyleSheet * GetOldStyleSheet () const

  Returns the old style sheet.
- void SetOldStyleSheet (wxRichTextStyleSheet *sheet)

  Sets the old style sheet variable.
- wxRichTextStyleSheet * GetNewStyleSheet () const

  Returns the new style sheet.
- void SetNewStyleSheet (wxRichTextStyleSheet *sheet)

  Sets the new style sheet variable.
- const wxRichTextRange & GetRange () const

  Gets the range for the current operation.
- void SetRange (const wxRichTextRange &range)

  Sets the range variable.
- wxChar GetCharacter () const

  Returns the character pressed, within a `wxEVT_RICHTEXT_CHARACTER` event.
- void SetCharacter (wxChar ch)

  Sets the character variable.
- wxRichTextParagraphLayoutBox * GetContainer () const

  Returns the container for which the event is relevant.
- void SetContainer (wxRichTextParagraphLayoutBox *container)

  Sets the container for which the event is relevant.
- wxRichTextParagraphLayoutBox * GetOldContainer () const

  Returns the old container, for a focus change event.
- void SetOldContainer (wxRichTextParagraphLayoutBox *container)

  Sets the old container, for a focus change event.
- virtual wxEvent * Clone () const

  Returns a copy of the event.

Protected Attributes

- int m_flags
- long m_position
- wxRichTextStyleSheet * m_oldStyleSheet
- wxRichTextStyleSheet * m_newStyleSheet
- wxRichTextRange m_range
- wxChar m_char
- wxRichTextParagraphLayoutBox * m_container
- wxRichTextParagraphLayoutBox * m_oldContainer

Additional Inherited Members

21.620.2 Constructor & Destructor Documentation

wxRichTextEvent::wxRichTextEvent (wxEventType *commandType* = wxEVT_NULL, int *winid* = 0)

Constructor.

Parameters

commandType	The type of the event.
winid	Window identifier. The value `wxID_ANY` indicates a default value.

wxRichTextEvent::wxRichTextEvent (const wxRichTextEvent & *event*)

Copy constructor.

21.620.3 Member Function Documentation

virtual wxEvent* wxRichTextEvent::Clone ( ) const [virtual]

Returns a copy of the event.

Any event that is posted to the wxWidgets event system for later action (via wxEvtHandler::AddPendingEvent, wxEvtHandler::QueueEvent or wxPostEvent()) must implement this method.

All wxWidgets events fully implement this method, but any derived events implemented by the user should also implement this method just in case they (or some event derived from them) are ever posted.

All wxWidgets events implement a copy constructor, so the easiest way of implementing the Clone function is to implement a copy constructor for a new event (call it MyEvent) and then define the Clone function like this:

```
wxEvent *Clone() const { return new MyEvent(*this); }
```

Implements wxEvent.

wxChar wxRichTextEvent::GetCharacter ( ) const

Returns the character pressed, within a `wxEVT_RICHTEXT_CHARACTER` event.

wxRichTextParagraphLayoutBox* wxRichTextEvent::GetContainer ( ) const

Returns the container for which the event is relevant.

int wxRichTextEvent::GetFlags ( ) const

Returns flags indicating modifier keys pressed.

Possible values are `wxRICHTEXT_CTRL_DOWN`, `wxRICHTEXT_SHIFT_DOWN`, and `wxRICHTEXT_ALT_DOWN`.

wxRichTextStyleSheet* wxRichTextEvent::GetNewStyleSheet ( ) const

Returns the new style sheet.

Can be used in a `wxEVT_RICHTEXT_STYLESHEET_CHANGING` or `wxEVT_RICHTEXT_STYLESHEET_CHANGED` event handler.

wxRichTextParagraphLayoutBox* wxRichTextEvent::GetOldContainer ( ) const

Returns the old container, for a focus change event.

wxRichTextStyleSheet* wxRichTextEvent::GetOldStyleSheet ( ) const

Returns the old style sheet.

Can be used in a `wxEVT_RICHTEXT_STYLESHEET_CHANGING` or `wxEVT_RICHTEXT_STYLESHEET_CHANGED` event handler.

long wxRichTextEvent::GetPosition ( ) const

Returns the buffer position at which the event occurred.

const wxRichTextRange& wxRichTextEvent::GetRange ( ) const

Gets the range for the current operation.

void wxRichTextEvent::SetCharacter (wxChar *ch*)

Sets the character variable.

void wxRichTextEvent::SetContainer (wxRichTextParagraphLayoutBox * *container*)

Sets the container for which the event is relevant.

void wxRichTextEvent::SetFlags (int *flags*)

Sets flags indicating modifier keys pressed.

Possible values are `wxRICHTEXT_CTRL_DOWN`, `wxRICHTEXT_SHIFT_DOWN`, and `wxRICHTEXT_ALT_DOWN`.

void wxRichTextEvent::SetNewStyleSheet (wxRichTextStyleSheet * *sheet*)

Sets the new style sheet variable.

void wxRichTextEvent::SetOldContainer (wxRichTextParagraphLayoutBox * *container*)

Sets the old container, for a focus change event.

void wxRichTextEvent::SetOldStyleSheet (wxRichTextStyleSheet * *sheet*)

Sets the old style sheet variable.

void wxRichTextEvent::SetPosition (long *pos*)

Sets the buffer position variable.

void wxRichTextEvent::SetRange (const wxRichTextRange & *range*)

Sets the range variable.

21.620.4 Member Data Documentation

wxChar wxRichTextEvent::m_char [protected]

wxRichTextParagraphLayoutBox* wxRichTextEvent::m_container [protected]

int wxRichTextEvent::m_flags [protected]

wxRichTextStyleSheet* wxRichTextEvent::m_newStyleSheet [protected]

wxRichTextParagraphLayoutBox* wxRichTextEvent::m_oldContainer [protected]

wxRichTextStyleSheet* wxRichTextEvent::m_oldStyleSheet [protected]

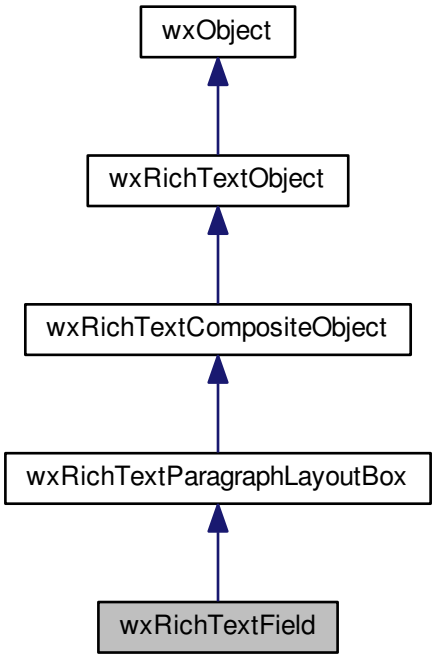
long wxRichTextEvent::m_position [protected]

wxRichTextRange wxRichTextEvent::m_range [protected]

21.621 wxRichTextField Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextField:



21.621.1 Detailed Description

This class implements the general concept of a field, an object that represents additional functionality such as a footnote, a bookmark, a page number, a table of contents, and so on.

Extra information (such as a bookmark name) can be stored in the object properties.

Drawing, layout, and property editing is delegated to classes derived from wxRichTextFieldType, such as instances of wxRichTextFieldTypeStandard; this makes the use of fields an efficient method of introducing extra functionality, since most of the information required to draw a field (such as a bitmap) is kept centrally in a single field type definition.

The FieldType property, accessed by SetFieldType/GetFieldType, is used to retrieve the field type definition. So be careful not to overwrite this property.

wxRichTextField is derived from wxRichTextParagraphLayoutBox, which means that it can contain its own read-only content, refreshed when the application calls the UpdateField function. Whether a field is treated as a composite or a single graphic is determined by the field type definition. If using wxRichTextFieldTypeStandard, passing the display type wxRICHTEXT_FIELD_STYLE_COMPOSITE to the field type definition causes the field to behave like a composite; the other display styles display a simple graphic. When implementing a composite field, you will still need to derive from wxRichTextFieldTypeStandard or wxRichTextFieldType, if only to implement UpdateField to refresh the field content appropriately. wxRichTextFieldTypeStandard is only one possible implementation, but covers common needs especially for simple, static fields using text or a bitmap.

Register field types on application initialisation with the static function wxRichTextBuffer::AddFieldType. They will be deleted automatically on application exit.

An application can write a field to a control with wxRichTextCtrl::WriteField, taking a field type, the properties for the field, and optional attributes.

Library: wxRichText

Category: Rich Text

See also

wxRichTextFieldTypeStandard, wxRichTextFieldType, wxRichTextParagraphLayoutBox, wxRichTextProperties, wxRichTextCtrl

Public Member Functions

- wxRichTextField (const wxString &fieldType=wxEmptyString, wxRichTextObject *parent=NULL)

  Default constructor; optionally pass the parent object.
- wxRichTextField (const wxRichTextField &obj)

  Copy constructor.
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

  Lay the item out at the specified position with the given size constraint.
- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- virtual bool CanEditProperties () const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxWindow *parent, wxRichTextBuffer *buffer)

  Edits the object's properties via a GUI.

- virtual wxString GetPropertiesMenuLabel () const

  Returns the label to be used for the properties context menu item.
- virtual bool AcceptsFocus () const

  Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.
- virtual void CalculateRange (long start, long &end)

  Calculates the range of the object.
- virtual bool IsAtomic () const

  If a field has children, we don't want the user to be able to edit it.
- virtual bool IsEmpty () const

  Returns true if the buffer is empty.
- virtual bool IsTopLevel () const

  Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.
- void SetFieldType (const wxString &fieldType)
- wxString GetFieldType () const
- virtual bool UpdateField (wxRichTextBuffer *buffer)

  Update the field; delegated to the associated field type.
- virtual wxRichTextObject * Clone () const

  Clones the object.
- void Copy (const wxRichTextField &obj)

Additional Inherited Members

21.621.2 Constructor & Destructor Documentation

wxRichTextField::wxRichTextField (const wxString & *fieldType* = wxEmptyString, wxRichTextObject * *parent* = NULL)

Default constructor; optionally pass the parent object.

wxRichTextField::wxRichTextField (const wxRichTextField & *obj*) `[inline]`

Copy constructor.

21.621.3 Member Function Documentation

virtual bool wxRichTextField::AcceptsFocus ( ) const `[inline], [virtual]`

Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.

For example, containers supporting text, such as a text box object, can accept the focus, but a table can't (set the focus to individual cells instead).

Reimplemented from wxRichTextParagraphLayoutBox.

virtual void wxRichTextField::CalculateRange (long *start*, long & *end*) `[virtual]`

Calculates the range of the object.

By default, guess that the object is 1 unit long.

Reimplemented from wxRichTextCompositeObject.

virtual bool wxRichTextField::CanEditProperties ( ) const `[virtual]`

Returns true if we can edit the object's properties via a GUI.

Reimplemented from wxRichTextObject.

virtual wxRichTextObject* wxRichTextField::Clone ( ) const `[inline], [virtual]`

Clones the object.

Reimplemented from wxRichTextParagraphLayoutBox.

void wxRichTextField::Copy (const wxRichTextField & *obj*)

virtual bool wxRichTextField::Draw (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRichTextRange & *range*, const wxRichTextSelection & *selection*, const wxRect & *rect*, int *descent*, int *style*) `[virtual]`

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextField::EditProperties (wxWindow * *parent*, wxRichTextBuffer * *buffer*) `[virtual]`

Edits the object's properties via a GUI.

Reimplemented from wxRichTextObject.

wxString wxRichTextField::GetFieldType ( ) const `[inline]`

virtual wxString wxRichTextField::GetPropertiesMenuLabel ( ) const `[virtual]`

Returns the label to be used for the properties context menu item.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextField::GetRangeSize (const wxRichTextRange & *range*, wxSize & *size*, int & *descent*, wxDC & *dc*, wxRichTextDrawingContext & *context*, int *flags*, const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const `[virtual]`

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual wxString wxRichTextField::GetXMLNodeName ( ) const `[inline], [virtual]`

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextField::IsAtomic ( ) const `[inline], [virtual]`

If a field has children, we don't want the user to be able to edit it.

Reimplemented from wxRichTextCompositeObject.

virtual bool wxRichTextField::IsEmpty ( ) const `[inline], [virtual]`

Returns true if the buffer is empty.

Reimplemented from wxRichTextCompositeObject.

virtual bool wxRichTextField::IsTopLevel ( ) const `[virtual]`

Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextField::Layout (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style*) `[virtual]`

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Reimplemented from wxRichTextParagraphLayoutBox.

void wxRichTextField::SetFieldType (const wxString & *fieldType*) `[inline]`

virtual bool wxRichTextField::UpdateField (wxRichTextBuffer * *buffer*) `[virtual]`

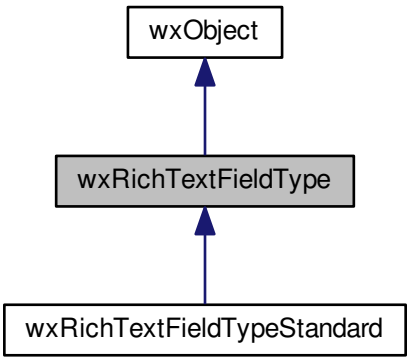
Update the field; delegated to the associated field type.

This would typically expand the field to its value, if this is a dynamically changing and/or composite field.

21.622 wxRichTextFieldType Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextFieldType:



21.622.1 Detailed Description

The base class for custom field types.

Each type definition handles one field type. Override functions to provide drawing, layout, updating and property editing functionality for a field.

Register field types on application initialisation with the static function wxRichTextBuffer::AddFieldType. They will be deleted automatically on application exit.

Library: wxRichText

Category: Rich Text

See also

wxRichTextFieldTypeStandard, wxRichTextField, wxRichTextCtrl

Public Member Functions

- wxRichTextFieldType (const wxString &name=wxEmptyString)

  Creates a field type definition.
- wxRichTextFieldType (const wxRichTextFieldType &fieldType)

  Copy constructor.
- void Copy (const wxRichTextFieldType &fieldType)
- virtual bool Draw (wxRichTextField *obj, wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)=0

  Draw the item, within the given range.
- virtual bool Layout (wxRichTextField *obj, wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)=0

  Lay the item out at the specified position with the given size constraint.

- virtual bool GetRangeSize (wxRichTextField ∗obj, const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt ∗partialExtents=NULL) const =0

  Returns the object size for the given range.
- virtual bool CanEditProperties (wxRichTextField ∗obj) const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxRichTextField ∗obj, wxWindow ∗parent, wxRichTextBuffer ∗buffer)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel (wxRichTextField ∗obj) const

  Returns the label to be used for the properties context menu item.
- virtual bool UpdateField (wxRichTextBuffer ∗buffer, wxRichTextField ∗obj)

  Update the field.
- virtual bool IsTopLevel (wxRichTextField ∗obj) const

  Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.
- void SetName (const wxString &name)

  Sets the field type name.
- wxString GetName () const

  Returns the field type name.

Protected Attributes

- wxString m_name

Additional Inherited Members

21.622.2 Constructor & Destructor Documentation

wxRichTextFieldType::wxRichTextFieldType (const wxString & *name* = wxEmptyString) `[inline]`

Creates a field type definition.

wxRichTextFieldType::wxRichTextFieldType (const wxRichTextFieldType & *fieldType*) `[inline]`

Copy constructor.

21.622.3 Member Function Documentation

virtual bool wxRichTextFieldType::CanEditProperties (wxRichTextField ∗ *obj*) const `[inline], [virtual]`

Returns true if we can edit the object's properties via a GUI.

void wxRichTextFieldType::Copy (const wxRichTextFieldType & *fieldType*) `[inline]`

virtual bool wxRichTextFieldType::Draw (wxRichTextField ∗ *obj,* wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) `[pure virtual]`

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implemented in wxRichTextFieldTypeStandard.

virtual bool wxRichTextFieldType::EditProperties (wxRichTextField * *obj*, wxWindow * *parent*, wxRichTextBuffer * *buffer*) [inline], [virtual]

Edits the object's properties via a GUI.

wxString wxRichTextFieldType::GetName ( ) const [inline]

Returns the field type name.

There should be a unique name per field type object.

virtual wxString wxRichTextFieldType::GetPropertiesMenuLabel (wxRichTextField * *obj*) const [inline], [virtual]

Returns the label to be used for the properties context menu item.

virtual bool wxRichTextFieldType::GetRangeSize (wxRichTextField * *obj*, const wxRichTextRange & *range*, wxSize & *size*, int & *descent*, wxDC & *dc*, wxRichTextDrawingContext & *context*, int *flags*, const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [pure virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Implemented in wxRichTextFieldTypeStandard.

virtual bool wxRichTextFieldType::IsTopLevel (wxRichTextField * *obj*) const [inline], [virtual]

Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.

Reimplemented in wxRichTextFieldTypeStandard.

virtual bool wxRichTextFieldType::Layout (wxRichTextField * *obj*, wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRect & *rect*, const wxRect & *parentRect*, int *style*) [pure virtual]

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implemented in wxRichTextFieldTypeStandard.

void wxRichTextFieldType::SetName (const wxString & *name*) [inline]

Sets the field type name.

There should be a unique name per field type object.

virtual bool wxRichTextFieldType::UpdateField (wxRichTextBuffer * *buffer*, wxRichTextField * *obj*) [inline], [virtual]

Update the field.

This would typically expand the field to its value, if this is a dynamically changing and/or composite field.

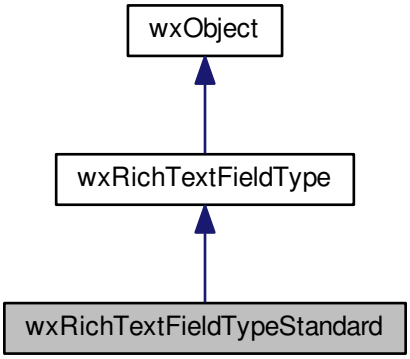
21.622.4 Member Data Documentation

wxString wxRichTextFieldType::m_name `[protected]`

21.623 wxRichTextFieldTypeStandard Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextFieldTypeStandard:



21.623.1 Detailed Description

A field type that can handle fields with text or bitmap labels, with a small range of styles for implementing rectangular fields and fields that can be used for start and end tags.

The border, text and background colours can be customised; the default is white text on a black background.

The following display styles can be used.

Styles

This class supports the following styles:

- wxRICHTEXT_FIELD_STYLE_COMPOSITE: Creates a composite field; you will probably need to derive a new class to implement UpdateField.
- wxRICHTEXT_FIELD_STYLE_RECTANGLE: Shows a rounded rectangle background.
- wxRICHTEXT_FIELD_STYLE_NO_BORDER: Suppresses the background and border; mostly used with a bitmap label.
- wxRICHTEXT_FIELD_STYLE_START_TAG: Shows a start tag background, with the pointy end facing right.
- wxRICHTEXT_FIELD_STYLE_END_TAG: Shows an end tag background, with the pointy end facing left.

Library: wxRichText

Category: Rich Text

See also

wxRichTextFieldType, wxRichTextField, wxRichTextBuffer, wxRichTextCtrl

Public Types

- enum {
  wxRICHTEXT_FIELD_STYLE_COMPOSITE = 0x01,
  wxRICHTEXT_FIELD_STYLE_RECTANGLE = 0x02,
  wxRICHTEXT_FIELD_STYLE_NO_BORDER = 0x04,
  wxRICHTEXT_FIELD_STYLE_START_TAG = 0x08,
  wxRICHTEXT_FIELD_STYLE_END_TAG = 0x10 }

Public Member Functions

- wxRichTextFieldTypeStandard (const wxString &name, const wxString &label, int displayStyle=wxRICHTEXT_FIELD_STYLE_RECTANGLE)

  Constructor, creating a field type definition with a text label.
- wxRichTextFieldTypeStandard (const wxString &name, const wxBitmap &bitmap, int displayStyle=wxRICHTEXT_FIELD_STYLE_NO_BORDER)

  Constructor, creating a field type definition with a bitmap label.
- wxRichTextFieldTypeStandard ()

  The default constructor.
- wxRichTextFieldTypeStandard (const wxRichTextFieldTypeStandard &field)

  The copy constructor.
- void Init ()

  Initialises the object.
- void Copy (const wxRichTextFieldTypeStandard &field)

  Copies the object.
- void operator= (const wxRichTextFieldTypeStandard &field)

  The assignment operator.
- virtual bool Draw (wxRichTextField *obj, wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual bool Layout (wxRichTextField *obj, wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

  Lay the item out at the specified position with the given size constraint.
- virtual bool GetRangeSize (wxRichTextField *obj, const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.
- wxSize GetSize (wxRichTextField *obj, wxDC &dc, wxRichTextDrawingContext &context, int style) const

  Get the size of the field, given the label, font size, and so on.
- virtual bool IsTopLevel (wxRichTextField *obj) const

  Returns true if the display type is wxRICHTEXT_FIELD_STYLE_COMPOSITE, false otherwise.
- void SetLabel (const wxString &label)

  Sets the text label for fields of this type.
- const wxString & GetLabel () const

  Returns the text label for fields of this type.
- void SetBitmap (const wxBitmap &bitmap)

  Sets the bitmap label for fields of this type.

- const wxBitmap & GetBitmap () const

  Gets the bitmap label for fields of this type.
- int GetDisplayStyle () const

  Gets the display style for fields of this type.
- void SetDisplayStyle (int displayStyle)

  Sets the display style for fields of this type.
- const wxFont & GetFont () const

  Gets the font used for drawing the text label.
- void SetFont (const wxFont &font)

  Sets the font used for drawing the text label.
- const wxColour & GetTextColour () const

  Gets the colour used for drawing the text label.
- void SetTextColour (const wxColour &colour)

  Sets the colour used for drawing the text label.
- const wxColour & GetBorderColour () const

  Gets the colour used for drawing the field border.
- void SetBorderColour (const wxColour &colour)

  Sets the colour used for drawing the field border.
- const wxColour & GetBackgroundColour () const

  Gets the colour used for drawing the field background.
- void SetBackgroundColour (const wxColour &colour)

  Sets the colour used for drawing the field background.
- void SetVerticalPadding (int padding)

  Sets the vertical padding (the distance between the border and the text).
- int GetVerticalPadding () const

  Gets the vertical padding (the distance between the border and the text).
- void SetHorizontalPadding (int padding)

  Sets the horizontal padding (the distance between the border and the text).
- int GetHorizontalPadding () const

  Sets the horizontal padding (the distance between the border and the text).
- void SetHorizontalMargin (int margin)

  Sets the horizontal margin surrounding the field object.
- int GetHorizontalMargin () const

  Gets the horizontal margin surrounding the field object.
- void SetVerticalMargin (int margin)

  Sets the vertical margin surrounding the field object.
- int GetVerticalMargin () const

  Gets the vertical margin surrounding the field object.

Protected Attributes

- wxString m_label
- int m_displayStyle
- wxFont m_font
- wxColour m_textColour
- wxColour m_borderColour
- wxColour m_backgroundColour
- int m_verticalPadding
- int m_horizontalPadding
- int m_horizontalMargin
- int m_verticalMargin
- wxBitmap m_bitmap

Additional Inherited Members

21.623.2 Member Enumeration Documentation

anonymous enum

Enumerator

wxRICHTEXT_FIELD_STYLE_COMPOSITE

wxRICHTEXT_FIELD_STYLE_RECTANGLE

wxRICHTEXT_FIELD_STYLE_NO_BORDER

wxRICHTEXT_FIELD_STYLE_START_TAG

wxRICHTEXT_FIELD_STYLE_END_TAG

21.623.3 Constructor & Destructor Documentation

wxRichTextFieldTypeStandard::wxRichTextFieldTypeStandard (const wxString & *name,* const wxString & *label,* int *displayStyle* = wxRICHTEXT_FIELD_STYLE_RECTANGLE)

Constructor, creating a field type definition with a text label.

Parameters

name	The name of the type definition. This must be unique, and is the type name used when adding a field to a control.
label	The text label to be shown on the field.
displayStyle	The display style: one of wxRICHTEXT_FIELD_STYLE_RECTANGLE, wxRICHTEXT_FIELD_STYLE_NO_BORDER, wxRICHTEXT_FIELD_STYLE_START_TAG, wxRICHTEXT_FIELD_STYLE_END_TAG.

wxRichTextFieldTypeStandard::wxRichTextFieldTypeStandard (const wxString & *name,* const wxBitmap & *bitmap,* int *displayStyle* = wxRICHTEXT_FIELD_STYLE_NO_BORDER)

Constructor, creating a field type definition with a bitmap label.

Parameters

name	The name of the type definition. This must be unique, and is the type name used when adding a field to a control.
bitmap	The bitmap label to be shown on the field.
displayStyle	The display style: one of wxRICHTEXT_FIELD_STYLE_RECTANGLE, wxRICHTEXT_FIELD_STYLE_NO_BORDER, wxRICHTEXT_FIELD_STYLE_START_TAG, wxRICHTEXT_FIELD_STYLE_END_TAG.

wxRichTextFieldTypeStandard::wxRichTextFieldTypeStandard ( ) [inline]

The default constructor.

wxRichTextFieldTypeStandard::wxRichTextFieldTypeStandard (const wxRichTextFieldTypeStandard & *field*) [inline]

The copy constructor.

21.623.4 Member Function Documentation

void wxRichTextFieldTypeStandard::Copy (const wxRichTextFieldTypeStandard & *field*)

Copies the object.

virtual bool wxRichTextFieldTypeStandard::Draw (wxRichTextField * *obj,* wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) [virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implements wxRichTextFieldType.

const wxColour& wxRichTextFieldTypeStandard::GetBackgroundColour ( ) const [inline]

Gets the colour used for drawing the field background.

const wxBitmap& wxRichTextFieldTypeStandard::GetBitmap ( ) const [inline]

Gets the bitmap label for fields of this type.

const wxColour& wxRichTextFieldTypeStandard::GetBorderColour ( ) const [inline]

Gets the colour used for drawing the field border.

int wxRichTextFieldTypeStandard::GetDisplayStyle ( ) const [inline]

Gets the display style for fields of this type.

const wxFont& wxRichTextFieldTypeStandard::GetFont ( ) const [inline]

Gets the font used for drawing the text label.

int wxRichTextFieldTypeStandard::GetHorizontalMargin ( ) const [inline]

Gets the horizontal margin surrounding the field object.

int wxRichTextFieldTypeStandard::GetHorizontalPadding ( ) const [inline]

Sets the horizontal padding (the distance between the border and the text).

const wxString& wxRichTextFieldTypeStandard::GetLabel ( ) const [inline]

Returns the text label for fields of this type.

virtual bool wxRichTextFieldTypeStandard::GetRangeSize (wxRichTextField * *obj*, const wxRichTextRange & *range*, wxSize & *size*, int & *descent*, wxDC & *dc*, wxRichTextDrawingContext & *context*, int *flags*, const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Implements wxRichTextFieldType.

wxSize wxRichTextFieldTypeStandard::GetSize (wxRichTextField * *obj*, wxDC & *dc*, wxRichTextDrawingContext & *context*, int *style*) const

Get the size of the field, given the label, font size, and so on.

const wxColour& wxRichTextFieldTypeStandard::GetTextColour ( ) const [inline]

Gets the colour used for drawing the text label.

int wxRichTextFieldTypeStandard::GetVerticalMargin ( ) const [inline]

Gets the vertical margin surrounding the field object.

int wxRichTextFieldTypeStandard::GetVerticalPadding ( ) const [inline]

Gets the vertical padding (the distance between the border and the text).

void wxRichTextFieldTypeStandard::Init ( )

Initialises the object.

virtual bool wxRichTextFieldTypeStandard::IsTopLevel (wxRichTextField * *obj*) const [inline], [virtual]

Returns true if the display type is wxRICHTEXT_FIELD_STYLE_COMPOSITE, false otherwise.

Reimplemented from wxRichTextFieldType.

virtual bool wxRichTextFieldTypeStandard::Layout (wxRichTextField * *obj*, wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRect & *rect*, const wxRect & *parentRect*, int *style*) [virtual]

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implements wxRichTextFieldType.

void wxRichTextFieldTypeStandard::operator= (const wxRichTextFieldTypeStandard & *field*) [inline]

The assignment operator.

void wxRichTextFieldTypeStandard::SetBackgroundColour (const wxColour & *colour*) `[inline]`

Sets the colour used for drawing the field background.

void wxRichTextFieldTypeStandard::SetBitmap (const wxBitmap & *bitmap*) `[inline]`

Sets the bitmap label for fields of this type.

void wxRichTextFieldTypeStandard::SetBorderColour (const wxColour & *colour*) `[inline]`

Sets the colour used for drawing the field border.

void wxRichTextFieldTypeStandard::SetDisplayStyle (int *displayStyle*) `[inline]`

Sets the display style for fields of this type.

void wxRichTextFieldTypeStandard::SetFont (const wxFont & *font*) `[inline]`

Sets the font used for drawing the text label.

void wxRichTextFieldTypeStandard::SetHorizontalMargin (int *margin*) `[inline]`

Sets the horizontal margin surrounding the field object.

void wxRichTextFieldTypeStandard::SetHorizontalPadding (int *padding*) `[inline]`

Sets the horizontal padding (the distance between the border and the text).

void wxRichTextFieldTypeStandard::SetLabel (const wxString & *label*) `[inline]`

Sets the text label for fields of this type.

void wxRichTextFieldTypeStandard::SetTextColour (const wxColour & *colour*) `[inline]`

Sets the colour used for drawing the text label.

void wxRichTextFieldTypeStandard::SetVerticalMargin (int *margin*) `[inline]`

Sets the vertical margin surrounding the field object.

void wxRichTextFieldTypeStandard::SetVerticalPadding (int *padding*) `[inline]`

Sets the vertical padding (the distance between the border and the text).

21.623.5 Member Data Documentation

wxColour wxRichTextFieldTypeStandard::m_backgroundColour `[protected]`

wxBitmap wxRichTextFieldTypeStandard::m_bitmap `[protected]`

wxColour wxRichTextFieldTypeStandard::m_borderColour [protected]

int wxRichTextFieldTypeStandard::m_displayStyle [protected]

wxFont wxRichTextFieldTypeStandard::m_font [protected]

int wxRichTextFieldTypeStandard::m_horizontalMargin [protected]

int wxRichTextFieldTypeStandard::m_horizontalPadding [protected]

wxString wxRichTextFieldTypeStandard::m_label [protected]

wxColour wxRichTextFieldTypeStandard::m_textColour [protected]

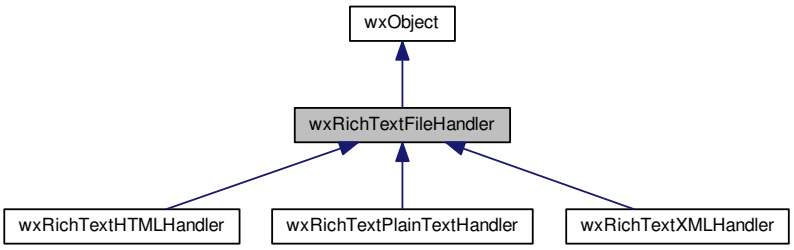
int wxRichTextFieldTypeStandard::m_verticalMargin [protected]

int wxRichTextFieldTypeStandard::m_verticalPadding [protected]

21.624 wxRichTextFileHandler Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextFileHandler:



21.624.1 Detailed Description

The base class for file handlers.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextFileHandler (const wxString &name=wxEmptyString, const wxString &ext=wxEmptyString, int type=0)

Creates a file handler object.

- bool LoadFile (wxRichTextBuffer *buffer, wxInputStream &stream)

  Loads the buffer from a stream.
- bool SaveFile (wxRichTextBuffer *buffer, wxOutputStream &stream)

  Saves the buffer to a stream.
- virtual bool LoadFile (wxRichTextBuffer *buffer, const wxString &filename)

  Loads the buffer from a file.
- virtual bool SaveFile (wxRichTextBuffer *buffer, const wxString &filename)

  Saves the buffer to a file.
- virtual bool CanHandle (const wxString &filename) const

  Returns true if we handle this filename (if using files).
- virtual bool CanSave () const

  Returns true if we can save using this handler.
- virtual bool CanLoad () const

  Returns true if we can load using this handler.
- virtual bool IsVisible () const

  Returns true if this handler should be visible to the user.
- virtual void SetVisible (bool visible)

  Sets whether the handler should be visible to the user (via the application's load and save dialogs).
- void SetName (const wxString &name)

  Sets the name of the handler.
- wxString GetName () const

  Returns the name of the handler.
- void SetExtension (const wxString &ext)

  Sets the default extension to recognise.
- wxString GetExtension () const

  Returns the default extension to recognise.
- void SetType (int type)

  Sets the handler type.
- int GetType () const

  Returns the handler type.
- void SetFlags (int flags)

  Sets flags that change the behaviour of loading or saving.
- int GetFlags () const

  Returns flags controlling how loading and saving is done.
- void SetEncoding (const wxString &encoding)

  Sets the encoding to use when saving a file.
- const wxString & GetEncoding () const

  Returns the encoding to use when saving a file.

Protected Member Functions

- virtual bool DoLoadFile (wxRichTextBuffer *buffer, wxInputStream &stream)=0

  Override to load content from stream into buffer.
- virtual bool DoSaveFile (wxRichTextBuffer *buffer, wxOutputStream &stream)=0

  Override to save content to stream from buffer.

Protected Attributes

- wxString m_name
- wxString m_encoding
- wxString m_extension
- int m_type
- int m_flags
- bool m_visible

21.624.2 Constructor & Destructor Documentation

wxRichTextFileHandler::wxRichTextFileHandler (const wxString & *name* = wxEmptyString, const wxString & *ext* = wxEmptyString, int *type* = 0) `[inline]`

Creates a file handler object.

21.624.3 Member Function Documentation

virtual bool wxRichTextFileHandler::CanHandle (const wxString & *filename*) const `[virtual]`

Returns true if we handle this filename (if using files).

By default, checks the extension.

virtual bool wxRichTextFileHandler::CanLoad ( ) const `[inline]`, `[virtual]`

Returns true if we can load using this handler.

Reimplemented in wxRichTextPlainTextHandler, and wxRichTextXMLHandler.

virtual bool wxRichTextFileHandler::CanSave ( ) const `[inline]`, `[virtual]`

Returns true if we can save using this handler.

Reimplemented in wxRichTextPlainTextHandler, and wxRichTextXMLHandler.

virtual bool wxRichTextFileHandler::DoLoadFile (wxRichTextBuffer * *buffer*, wxInputStream & *stream*) `[protected]`, `[pure virtual]`

Override to load content from *stream* into *buffer*.

Implemented in wxRichTextPlainTextHandler, and wxRichTextXMLHandler.

virtual bool wxRichTextFileHandler::DoSaveFile (wxRichTextBuffer * *buffer*, wxOutputStream & *stream*) `[protected]`, `[pure virtual]`

Override to save content to *stream* from *buffer*.

Implemented in wxRichTextPlainTextHandler, wxRichTextHTMLHandler, and wxRichTextXMLHandler.

const wxString& wxRichTextFileHandler::GetEncoding ( ) const `[inline]`

Returns the encoding to use when saving a file.

If empty, a suitable encoding is chosen.

wxString wxRichTextFileHandler::GetExtension ( ) const [inline]

Returns the default extension to recognise.

int wxRichTextFileHandler::GetFlags ( ) const [inline]

Returns flags controlling how loading and saving is done.

wxString wxRichTextFileHandler::GetName ( ) const [inline]

Returns the name of the handler.

int wxRichTextFileHandler::GetType ( ) const [inline]

Returns the handler type.

virtual bool wxRichTextFileHandler::IsVisible ( ) const [inline], [virtual]

Returns true if this handler should be visible to the user.

bool wxRichTextFileHandler::LoadFile (wxRichTextBuffer * *buffer,* wxInputStream & *stream*) [inline]

Loads the buffer from a stream.

Not all handlers will implement file loading.

virtual bool wxRichTextFileHandler::LoadFile (wxRichTextBuffer * *buffer,* const wxString & *filename*) [virtual]

Loads the buffer from a file.

bool wxRichTextFileHandler::SaveFile (wxRichTextBuffer * *buffer,* wxOutputStream & *stream*) [inline]

Saves the buffer to a stream.

Not all handlers will implement file saving.

virtual bool wxRichTextFileHandler::SaveFile (wxRichTextBuffer * *buffer,* const wxString & *filename*) [virtual]

Saves the buffer to a file.

void wxRichTextFileHandler::SetEncoding (const wxString & *encoding*) [inline]

Sets the encoding to use when saving a file.

If empty, a suitable encoding is chosen.

void wxRichTextFileHandler::SetExtension (const wxString & *ext*) [inline]

Sets the default extension to recognise.

void wxRichTextFileHandler::SetFlags (int *flags*) [inline]

Sets flags that change the behaviour of loading or saving.

See the documentation for each handler class to see what flags are relevant for each handler.

You call this function directly if you are using a file handler explicitly (without going through the text control or buffer LoadFile/SaveFile API). Or, you can call the control or buffer's SetHandlerFlags function to set the flags that will be used for subsequent load and save operations.

void wxRichTextFileHandler::SetName (const wxString & *name*) [inline]

Sets the name of the handler.

void wxRichTextFileHandler::SetType (int *type*) [inline]

Sets the handler type.

virtual void wxRichTextFileHandler::SetVisible (bool *visible*) [inline], [virtual]

Sets whether the handler should be visible to the user (via the application's load and save dialogs).

21.624.4 Member Data Documentation

wxString wxRichTextFileHandler::m_encoding [protected]

wxString wxRichTextFileHandler::m_extension [protected]

int wxRichTextFileHandler::m_flags [protected]

wxString wxRichTextFileHandler::m_name [protected]

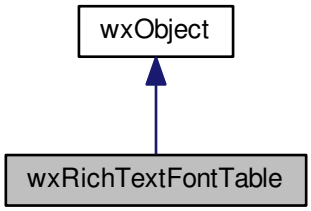
int wxRichTextFileHandler::m_type [protected]

bool wxRichTextFileHandler::m_visible [protected]

21.625 wxRichTextFontTable Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextFontTable:



21.625.1 Detailed Description

Manages quick access to a pool of fonts for rendering rich text.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextFontTable ()

  Default constructor.
- wxRichTextFontTable (const wxRichTextFontTable &table)

  Copy constructor.
- virtual ~wxRichTextFontTable ()
- bool IsOk () const

  Returns true if the font table is valid.
- wxFont FindFont (const wxRichTextAttr &fontSpec)

  Finds a font for the given attribute object.
- void Clear ()

  Clears the font table.
- void operator= (const wxRichTextFontTable &table)

  Assignment operator.
- bool operator== (const wxRichTextFontTable &table) const

  Equality operator.
- bool operator!= (const wxRichTextFontTable &table) const

  Inequality operator.
- void SetFontScale (double fontScale)

  Set the font scale factor.

Protected Attributes

- double m_fontScale

Additional Inherited Members

21.625.2 Constructor & Destructor Documentation

wxRichTextFontTable::wxRichTextFontTable ( )

Default constructor.

wxRichTextFontTable::wxRichTextFontTable (const wxRichTextFontTable & *table*)

Copy constructor.

virtual wxRichTextFontTable::~wxRichTextFontTable ( ) `[virtual]`

21.625.3 Member Function Documentation

void wxRichTextFontTable::Clear ( )

Clears the font table.

wxFont wxRichTextFontTable::FindFont (const wxRichTextAttr & *fontSpec*)

Finds a font for the given attribute object.

bool wxRichTextFontTable::IsOk ( ) const `[inline]`

Returns true if the font table is valid.

bool wxRichTextFontTable::operator!= (const wxRichTextFontTable & *table*) const `[inline]`

Inequality operator.

void wxRichTextFontTable::operator= (const wxRichTextFontTable & *table*)

Assignment operator.

bool wxRichTextFontTable::operator== (const wxRichTextFontTable & *table*) const

Equality operator.

void wxRichTextFontTable::SetFontScale (double *fontScale*)

Set the font scale factor.

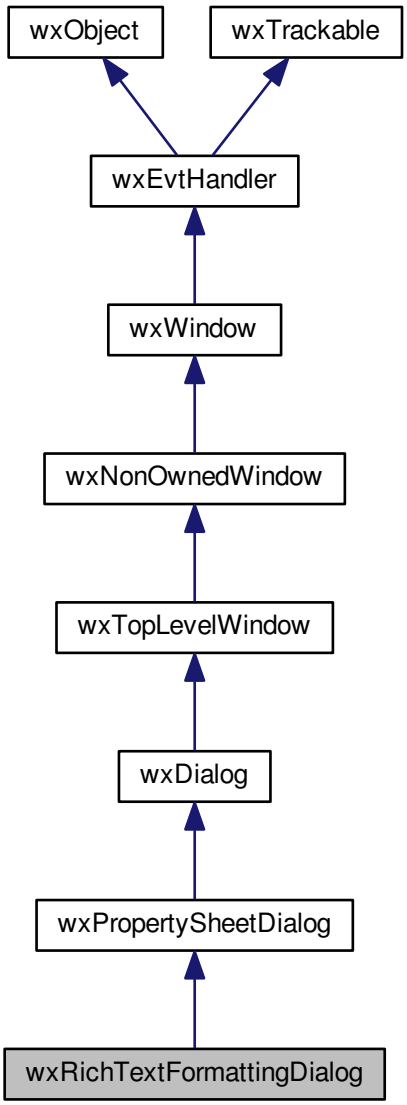
21.625.4 Member Data Documentation

double wxRichTextFontTable::m_fontScale `[protected]`

21.626 wxRichTextFormattingDialog Class Reference

`#include <wx/richtext/richtextformatdlg.h>`

Inheritance diagram for wxRichTextFormattingDialog:



21.626.1 Detailed Description

This dialog allows the user to edit a character and/or paragraph style.

In the constructor, specify the pages that will be created. Use wxRichTextFormattingDialog::GetStyle() to retrieve the common style for a given range, and then use wxRichTextFormattingDialog::ApplyStyle() to apply the user-selected formatting to a control.

For example:

```
wxRichTextRange range;
if (m_richTextCtrl-HasSelection())
    range = m_richTextCtrl-GetSelectionRange();
else
    range = wxRichTextRange(0, m_richTextCtrl-GetLastPosition()+1);

int pages = wxRICHTEXT_FORMAT_FONT|
      wxRICHTEXT_FORMAT_INDENTS_SPACING| \
            wxRICHTEXT_FORMAT_TABS|
      wxRICHTEXT_FORMAT_BULLETS;

wxRichTextFormattingDialog formatDlg(pages, this);
formatDlg.GetStyle(m_richTextCtrl, range);

if (formatDlg.ShowModal() == wxID_OK)
{
    formatDlg.ApplyStyle(m_richTextCtrl, range);
}
```

Library: wxRichText

Category: Rich Text

Public Types

- enum { Option_AllowPixelFontSize = 0x0001 }

Public Member Functions

- wxRichTextFormattingDialog ()

  Default ctor.
- wxRichTextFormattingDialog (long flags, wxWindow *parent, const wxString &title="Formatting", wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &sz=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE)

  Constructors.
- virtual ~wxRichTextFormattingDialog ()

  Destructor.
- virtual bool ApplyStyle (wxRichTextCtrl *ctrl, const wxRichTextRange &range, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO|wxRICHTEXT_SETSTYLE_OPTIMIZE)

  Apply attributes to the given range, only changing attributes that need to be changed.
- bool Create (long flags, wxWindow *parent, const wxString &title=wxGetTranslation("Formatting"), wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &sz=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE)

  Creation: see wxRichTextFormattingDialog() "the constructor" for details about the parameters.
- wxImageList * GetImageList () const

  Returns the image list associated with the dialog, used for example if showing the dialog as a toolbook.
- virtual bool GetStyle (wxRichTextCtrl *ctrl, const wxRichTextRange &range)

  Gets common attributes from the given range and calls SetAttributes().
- virtual wxRichTextStyleDefinition * GetStyleDefinition () const

  Gets the associated style definition, if any.
- virtual wxRichTextStyleSheet * GetStyleSheet () const

  Gets the associated style sheet, if any.

- void SetAttributes (const wxTextAttr &attr)

  Sets the attributes to be edited.
- void SetOptions (int options)

  Sets the dialog options, determining what the interface presents to the user.
- int GetOptions () const

  Gets the dialog options, determining what the interface presents to the user.
- bool HasOption (int option) const

  Returns true if the given option is present.
- void SetImageList (wxImageList *imageList)

  Sets the image list associated with the dialog's property sheet.
- virtual bool SetStyle (const wxTextAttr &style, bool update=true)

  Sets the attributes and optionally updates the display, if update is true.
- virtual bool SetStyleDefinition (const wxRichTextStyleDefinition &styleDef, wxRichTextStyleSheet *sheet, bool update=true)

  Sets the style definition and optionally update the display, if update is true.
- virtual bool UpdateDisplay ()

  Updates the display.

- const wxTextAttr & GetAttributes () const

  Gets the attributes being edited.
- wxTextAttr & GetAttributes ()

  Gets the attributes being edited.

Static Public Member Functions

- static wxRichTextFormattingDialog * GetDialog (wxWindow *win)

  Helper for pages to get the top-level dialog.
- static wxTextAttr * GetDialogAttributes (wxWindow *win)

  Helper for pages to get the attributes.
- static wxRichTextStyleDefinition * GetDialogStyleDefinition (wxWindow *win)

  Helper for pages to get the style.
- static wxRichTextFormattingDialogFactory * GetFormattingDialogFactory ()

  Returns the object to be used to customize the dialog and provide pages.
- static void SetFormattingDialogFactory (wxRichTextFormattingDialogFactory *factory)

  Sets the formatting factory object to be used for customization and page creation.
- static bool GetRestoreLastPage ()

  Returns true if the dialog will restore the last-selected page.
- static void SetRestoreLastPage (bool b)

  Pass true if the dialog should restore the last-selected page.
- static int GetLastPage ()

  Returns the page identifier of the last page selected (not the control id).
- static void SetLastPage (int lastPage)

  Sets the page identifier of the last page selected (not the control id).
- static void SetColourData (const wxColourData &colourData)

  Sets the custom colour data for use by the colour dialog.
- static wxColourData GetColourData ()

  Returns the custom colour data for use by the colour dialog.

Additional Inherited Members

21.626.2 Member Enumeration Documentation

anonymous enum

Enumerator

Option_AllowPixelFontSize

21.626.3 Constructor & Destructor Documentation

wxRichTextFormattingDialog::wxRichTextFormattingDialog ( )

Default ctor.

wxRichTextFormattingDialog::wxRichTextFormattingDialog (long *flags*, wxWindow * *parent*, const wxString & *title* = "Formatting", wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *sz* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE)

Constructors.

Parameters

flags	The pages to show.
parent	The dialog's parent.
title	The dialog's title.
id	The dialog's ID.
pos	The dialog's position.
sz	The dialog's size.
style	The dialog's window style.

virtual wxRichTextFormattingDialog::~wxRichTextFormattingDialog ( ) [virtual]

Destructor.

21.626.4 Member Function Documentation

virtual bool wxRichTextFormattingDialog::ApplyStyle (wxRichTextCtrl * *ctrl*, const wxRichTextRange & *range*, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO|wxRICHTEXT_SETSTYLE_OPTIMIZE) [virtual]

Apply attributes to the given range, only changing attributes that need to be changed.

bool wxRichTextFormattingDialog::Create (long *flags*, wxWindow * *parent*, const wxString & *title* = wxGetTranslation("Formatting"), wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *sz* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE)

Creation: see wxRichTextFormattingDialog() "the constructor" for details about the parameters.

const wxTextAttr& wxRichTextFormattingDialog::GetAttributes ( ) const

Gets the attributes being edited.

wxTextAttr& wxRichTextFormattingDialog::GetAttributes ( )

Gets the attributes being edited.

static wxColourData wxRichTextFormattingDialog::GetColourData ( ) [static]

Returns the custom colour data for use by the colour dialog.

static wxRichTextFormattingDialog* wxRichTextFormattingDialog::GetDialog (wxWindow * *win*) [static]

Helper for pages to get the top-level dialog.

static wxTextAttr* wxRichTextFormattingDialog::GetDialogAttributes (wxWindow * *win*) [static]

Helper for pages to get the attributes.

static wxRichTextStyleDefinition* wxRichTextFormattingDialog::GetDialogStyleDefinition (wxWindow * *win*) [static]

Helper for pages to get the style.

static wxRichTextFormattingDialogFactory* wxRichTextFormattingDialog::GetFormattingDialogFactory ( ) [static]

Returns the object to be used to customize the dialog and provide pages.

wxImageList* wxRichTextFormattingDialog::GetImageList ( ) const

Returns the image list associated with the dialog, used for example if showing the dialog as a toolbook.

static int wxRichTextFormattingDialog::GetLastPage ( ) [static]

Returns the page identifier of the last page selected (not the control id).

int wxRichTextFormattingDialog::GetOptions ( ) const [inline]

Gets the dialog options, determining what the interface presents to the user.

Currently the only option is Option_AllowPixelFontSize.

static bool wxRichTextFormattingDialog::GetRestoreLastPage ( ) [static]

Returns true if the dialog will restore the last-selected page.

virtual bool wxRichTextFormattingDialog::GetStyle (wxRichTextCtrl * *ctrl*, const wxRichTextRange & *range*) [virtual]

Gets common attributes from the given range and calls SetAttributes().

Attributes that do not have common values in the given range will be omitted from the style's flags.

virtual wxRichTextStyleDefinition* wxRichTextFormattingDialog::GetStyleDefinition ( ) const [virtual]

Gets the associated style definition, if any.

virtual wxRichTextStyleSheet* wxRichTextFormattingDialog::GetStyleSheet ( ) const [virtual]

Gets the associated style sheet, if any.

bool wxRichTextFormattingDialog::HasOption (int *option*) const [inline]

Returns true if the given option is present.

void wxRichTextFormattingDialog::SetAttributes (const wxTextAttr & *attr*)

Sets the attributes to be edited.

static void wxRichTextFormattingDialog::SetColourData (const wxColourData & *colourData*) [static]

Sets the custom colour data for use by the colour dialog.

static void wxRichTextFormattingDialog::SetFormattingDialogFactory (wxRichTextFormattingDialogFactory * *factory*) [static]

Sets the formatting factory object to be used for customization and page creation.

It deletes the existing factory object.

void wxRichTextFormattingDialog::SetImageList (wxImageList * *imageList*)

Sets the image list associated with the dialog's property sheet.

static void wxRichTextFormattingDialog::SetLastPage (int *lastPage*) [static]

Sets the page identifier of the last page selected (not the control id).

void wxRichTextFormattingDialog::SetOptions (int *options*) [inline]

Sets the dialog options, determining what the interface presents to the user.

Currently the only option is Option_AllowPixelFontSize.

static void wxRichTextFormattingDialog::SetRestoreLastPage (bool *b*) [static]

Pass true if the dialog should restore the last-selected page.

virtual bool wxRichTextFormattingDialog::SetStyle (const wxTextAttr & *style,* bool *update* = true) [virtual]

Sets the attributes and optionally updates the display, if *update* is true.

virtual bool wxRichTextFormattingDialog::SetStyleDefinition (const wxRichTextStyleDefinition & *styleDef,* wxRichTextStyleSheet * *sheet,* bool *update* = `true`) [virtual]

Sets the style definition and optionally update the display, if *update* is true.

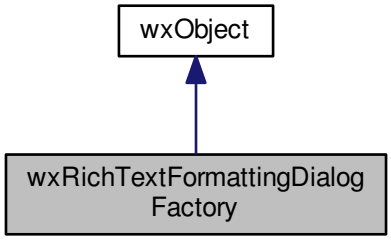
virtual bool wxRichTextFormattingDialog::UpdateDisplay ( ) [virtual]

Updates the display.

21.627 wxRichTextFormattingDialogFactory Class Reference

`#include <wx/richtext/richtextformatdlg.h>`

Inheritance diagram for wxRichTextFormattingDialogFactory:



21.627.1 Detailed Description

This class provides pages for wxRichTextFormattingDialog, and allows other customization of the dialog.

A default instance of this class is provided automatically. If you wish to change the behaviour of the formatting dialog (for example add or replace a page), you may derive from this class, override one or more functions, and call the static function wxRichTextFormattingDialog::SetFormattingDialogFactory.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextFormattingDialogFactory ()

  Constructor.
- virtual ~wxRichTextFormattingDialogFactory ()

  Destructor.
- virtual bool CreateButtons (wxRichTextFormattingDialog *dialog)

  Creates the main dialog buttons.
- virtual wxPanel * CreatePage (int page, wxString &title, wxRichTextFormattingDialog *dialog)

Creates a page, given a page identifier.

- virtual bool CreatePages (long pages, wxRichTextFormattingDialog *dialog)

  Creates all pages under the dialog's book control, also calling AddPage().
- virtual int GetPageId (int i) const

  Enumerate all available page identifiers.
- virtual int GetPageIdCount () const

  Gets the number of available page identifiers.
- virtual int GetPageImage (int id) const

  Gets the image index for the given page identifier.
- virtual bool SetSheetStyle (wxRichTextFormattingDialog *dialog)

  Set the property sheet style, called at the start of wxRichTextFormattingDialog::Create.
- virtual bool ShowHelp (int page, wxRichTextFormattingDialog *dialog)

  Invokes help for the dialog.

Additional Inherited Members

21.627.2 Constructor & Destructor Documentation

wxRichTextFormattingDialogFactory::wxRichTextFormattingDialogFactory ( )

Constructor.

virtual wxRichTextFormattingDialogFactory::~wxRichTextFormattingDialogFactory ( ) [virtual]

Destructor.

21.627.3 Member Function Documentation

virtual bool wxRichTextFormattingDialogFactory::CreateButtons (wxRichTextFormattingDialog * *dialog*) [virtual]

Creates the main dialog buttons.

virtual wxPanel* wxRichTextFormattingDialogFactory::CreatePage (int *page,* wxString & *title,* wxRichTextFormattingDialog * *dialog*) [virtual]

Creates a page, given a page identifier.

virtual bool wxRichTextFormattingDialogFactory::CreatePages (long *pages,* wxRichTextFormattingDialog * *dialog*) [virtual]

Creates all pages under the dialog's book control, also calling AddPage().

virtual int wxRichTextFormattingDialogFactory::GetPageId (int *i*) const [virtual]

Enumerate all available page identifiers.

virtual int wxRichTextFormattingDialogFactory::GetPageIdCount ( ) const [virtual]

Gets the number of available page identifiers.

virtual int wxRichTextFormattingDialogFactory::GetPageImage (int *id*) const `[virtual]`

Gets the image index for the given page identifier.

virtual bool wxRichTextFormattingDialogFactory::SetSheetStyle (wxRichTextFormattingDialog * *dialog*) `[virtual]`

Set the property sheet style, called at the start of wxRichTextFormattingDialog::Create.

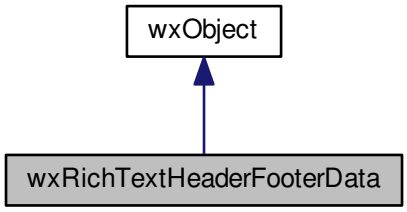
virtual bool wxRichTextFormattingDialogFactory::ShowHelp (int *page*, wxRichTextFormattingDialog * *dialog*) `[virtual]`

Invokes help for the dialog.

21.628 wxRichTextHeaderFooterData Class Reference

`#include <wx/richtext/richtextprint.h>`

Inheritance diagram for wxRichTextHeaderFooterData:



21.628.1 Detailed Description

This class represents header and footer data to be passed to the wxRichTextPrinting and wxRichTextPrintout classes.

Headers and footers can be specified independently for odd, even or both page sides. Different text can be specified for left, centre and right locations on the page, and the font and text colour can also be specified.

You can specify the following keywords in header and footer text, which will be substituted for the actual values during printing and preview.

- @DATE@: the current date.
- @PAGESCNT@: the total number of pages.
- @PAGENUM@: the current page number.
- @TIME@: the current time.
- @TITLE@: the title of the document, as passed to the wxRichTextPrinting or wxRichTextLayout constructor.

Library: wxRichText

Category: Rich Text

Public Member Functions

- void Clear ()

  Clears all text.
- void Copy (const wxRichTextHeaderFooterData &data)

  Copies the data.
- const wxFont & GetFont () const

  Returns the font specified for printing the header and footer.
- int GetFooterMargin () const

  Returns the margin between the text and the footer.
- wxString GetFooterText (wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE) const

  Returns the footer text on odd or even pages, and at a given position on the page (left, centre or right).
- int GetHeaderMargin () const

  Returns the margin between the text and the header.
- wxString GetHeaderText (wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE) const

  Returns the header text on odd or even pages, and at a given position on the page (left, centre or right).
- bool GetShowOnFirstPage () const

  Returns true if the header and footer will be shown on the first page.
- wxString GetText (int headerFooter, wxRichTextOddEvenPage page, wxRichTextPageLocation location) const

  Helper function for getting the header or footer text, odd or even pages, and at a given position on the page (left, centre or right).
- const wxColour & GetTextColour () const

  Returns the text colour for drawing the header and footer.
- void Init ()

  Initialises the object.
- void SetFont (const wxFont &font)

  Sets the font for drawing the header and footer.
- void SetFooterText (const wxString &text, wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE)

  Sets the footer text on odd or even pages, and at a given position on the page (left, centre or right).
- void SetHeaderText (const wxString &text, wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE)

  Sets the header text on odd or even pages, and at a given position on the page (left, centre or right).
- void SetMargins (int headerMargin, int footerMargin)

  Sets the margins between text and header or footer, in tenths of a millimeter.
- void SetShowOnFirstPage (bool showOnFirstPage)

  Pass true to show the header or footer on first page (the default).
- void SetText (const wxString &text, int headerFooter, wxRichTextOddEvenPage page, wxRichTextPageLocation location)

  Helper function for setting the header or footer text, odd or even pages, and at a given position on the page (left, centre or right).
- void SetTextColour (const wxColour &col)

  Sets the text colour for drawing the header and footer.
- void operator operator= (const wxRichTextHeaderFooterData &data)

Assignment operator.

- wxRichTextHeaderFooterData ()
  Constructors.
- wxRichTextHeaderFooterData (const wxRichTextHeaderFooterData &data)
  Constructors.

Additional Inherited Members

21.628.2 Constructor & Destructor Documentation

wxRichTextHeaderFooterData::wxRichTextHeaderFooterData ( )

Constructors.

wxRichTextHeaderFooterData::wxRichTextHeaderFooterData (const wxRichTextHeaderFooterData & *data*)

Constructors.

21.628.3 Member Function Documentation

void wxRichTextHeaderFooterData::Clear ( )

Clears all text.

void wxRichTextHeaderFooterData::Copy (const wxRichTextHeaderFooterData & *data*)

Copies the data.

const wxFont& wxRichTextHeaderFooterData::GetFont ( ) const

Returns the font specified for printing the header and footer.

int wxRichTextHeaderFooterData::GetFooterMargin ( ) const

Returns the margin between the text and the footer.

wxString wxRichTextHeaderFooterData::GetFooterText (wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE) const

Returns the footer text on odd or even pages, and at a given position on the page (left, centre or right).

int wxRichTextHeaderFooterData::GetHeaderMargin ( ) const

Returns the margin between the text and the header.

wxString wxRichTextHeaderFooterData::GetHeaderText (wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE) const

Returns the header text on odd or even pages, and at a given position on the page (left, centre or right).

bool wxRichTextHeaderFooterData::GetShowOnFirstPage ( ) const

Returns true if the header and footer will be shown on the first page.

wxString wxRichTextHeaderFooterData::GetText (int *headerFooter,* wxRichTextOddEvenPage *page,* wxRichTextPageLocation *location*) const

Helper function for getting the header or footer text, odd or even pages, and at a given position on the page (left, centre or right).

const wxColour& wxRichTextHeaderFooterData::GetTextColour ( ) const

Returns the text colour for drawing the header and footer.

void wxRichTextHeaderFooterData::Init ( )

Initialises the object.

void wxRichTextHeaderFooterData::operator operator= (const wxRichTextHeaderFooterData & *data*)

Assignment operator.

void wxRichTextHeaderFooterData::SetFont (const wxFont & *font*)

Sets the font for drawing the header and footer.

void wxRichTextHeaderFooterData::SetFooterText (const wxString & *text,* wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE)

Sets the footer text on odd or even pages, and at a given position on the page (left, centre or right).

void wxRichTextHeaderFooterData::SetHeaderText (const wxString & *text,* wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE)

Sets the header text on odd or even pages, and at a given position on the page (left, centre or right).

void wxRichTextHeaderFooterData::SetMargins (int *headerMargin,* int *footerMargin*)

Sets the margins between text and header or footer, in tenths of a millimeter.

void wxRichTextHeaderFooterData::SetShowOnFirstPage (bool *showOnFirstPage*)

Pass true to show the header or footer on first page (the default).

void wxRichTextHeaderFooterData::SetText (const wxString & *text,* int *headerFooter,* wxRichTextOddEvenPage *page,* wxRichTextPageLocation *location*)

Helper function for setting the header or footer text, odd or even pages, and at a given position on the page (left, centre or right).

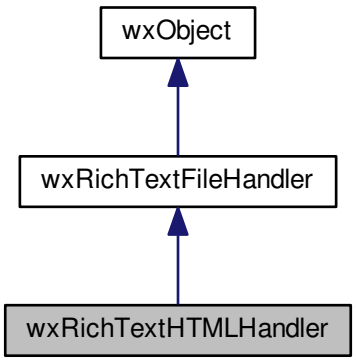
void wxRichTextHeaderFooterData::SetTextColour (const wxColour & *col*)

Sets the text colour for drawing the header and footer.

21.629 wxRichTextHTMLHandler Class Reference

`#include <wx/richtext/richtexthtml.h>`

Inheritance diagram for wxRichTextHTMLHandler:



21.629.1 Detailed Description

Handles HTML output (only) for wxRichTextCtrl content.

The most flexible way to use this class is to create a temporary object and call its functions directly, rather than use wxRichTextBuffer::SaveFile or wxRichTextCtrl::SaveFile.

Image handling requires a little extra work from the application, to choose an appropriate image format for the target HTML viewer and to clean up the temporary images later. If you are planning to load the HTML into a standard web browser, you can specify the handler flag wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_BASE64 (the default) and no extra work is required: the images will be written with the HTML.

However, if you want wxHTML compatibility, you will need to use `wxRICHTEXT_HANDLER_SAVE_IMAGES_↩TO_MEMORY` or `wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_FILES`.

In this case, you must either call wxRichTextHTMLHandler::DeleteTemporaryImages before the next load operation, or you must store the image locations and delete them yourself when appropriate.

You can call wxRichTextHTMLHandler::GetTemporaryImageLocations to get the array of temporary image names.

21.629.2 Handler flags

The following flags can be used with this handler, via the handler's SetFlags() function or the buffer or control's SetHandlerFlags() function:

- wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_MEMORY Images are saved to the memory filesystem↩ : suitable for showing wxHTML windows.

- wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_FILES Images are saved to temporary files: suitable for showing in wxHTML windows.
- wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_BASE64 Images are written with the HTML files in Base 64 format: suitable for showing in web browsers.
- wxRICHTEXT_HANDLER_NO_HEADER_FOOTER Don't include header and footer tags (HTML, HEAD, BODY), so that the HTML can be used as part of a larger document.
- wxRICHTEXT_HANDLER_USE_CSS Use CSS where possible, otherwise use workarounds that will show in wxHtmlWindow.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextHTMLHandler (const wxString &name="HTML", const wxString &ext="html", int type=wxRICHTEXT_TYPE_HTML)

  Constructor.
- void ClearTemporaryImageLocations ()

  Clears the image locations generated by the last operation.
- bool DeleteTemporaryImages ()

  Deletes the in-memory or temporary files generated by the last operation.
- wxArrayInt GetFontSizeMapping () const

  Returns the mapping for converting point sizes to HTML font sizes.
- const wxString & GetTempDir () const

  Returns the directory used to store temporary image files.
- const wxArrayString & GetTemporaryImageLocations () const

  Returns the image locations for the last operation.
- void SetFontSizeMapping (const wxArrayInt &fontSizeMapping)

  Sets the mapping for converting point sizes to HTML font sizes.
- void SetTempDir (const wxString &tempDir)

  Sets the directory for storing temporary files.
- void SetTemporaryImageLocations (const wxArrayString &locations)

  Sets the list of image locations generated by the last operation.

Static Public Member Functions

- static bool DeleteTemporaryImages (int flags, const wxArrayString &imageLocations)

  Delete the in-memory or temporary files generated by the last operation.
- static void SetFileCounter (int counter)

  Reset the file counter, in case, for example, the same names are required each time.

Protected Member Functions

- virtual bool DoSaveFile (wxRichTextBuffer *buffer, wxOutputStream &stream)

  Saves the buffer content to the HTML stream.

Additional Inherited Members

21.629.3 Constructor & Destructor Documentation

wxRichTextHTMLHandler::wxRichTextHTMLHandler (const wxString & *name* = "HTML", const wxString & *ext* = "html", int *type* = wxRICHTEXT_TYPE_HTML)

Constructor.

21.629.4 Member Function Documentation

void wxRichTextHTMLHandler::ClearTemporaryImageLocations ( )

Clears the image locations generated by the last operation.

bool wxRichTextHTMLHandler::DeleteTemporaryImages ( )

Deletes the in-memory or temporary files generated by the last operation.

static bool wxRichTextHTMLHandler::DeleteTemporaryImages (int *flags*, const wxArrayString & *imageLocations*) [static]

Delete the in-memory or temporary files generated by the last operation.

This is a static function that can be used to delete the saved locations from an earlier operation, for example after the user has viewed the HTML file.

virtual bool wxRichTextHTMLHandler::DoSaveFile (wxRichTextBuffer * *buffer*, wxOutputStream & *stream*) [protected], [virtual]

Saves the buffer content to the HTML stream.

Implements wxRichTextFileHandler.

wxArrayInt wxRichTextHTMLHandler::GetFontSizeMapping ( ) const

Returns the mapping for converting point sizes to HTML font sizes.

const wxString& wxRichTextHTMLHandler::GetTempDir ( ) const

Returns the directory used to store temporary image files.

const wxArrayString& wxRichTextHTMLHandler::GetTemporaryImageLocations ( ) const

Returns the image locations for the last operation.

static void wxRichTextHTMLHandler::SetFileCounter (int *counter*) [static]

Reset the file counter, in case, for example, the same names are required each time.

void wxRichTextHTMLHandler::SetFontSizeMapping (const wxArrayInt & *fontSizeMapping*)

Sets the mapping for converting point sizes to HTML font sizes.

There should be 7 elements, one for each HTML font size, each element specifying the maximum point size for that HTML font size. For example:

```
wxArrayInt fontSizeMapping;
fontSizeMapping.Add(7);
fontSizeMapping.Add(9);
fontSizeMapping.Add(11);
fontSizeMapping.Add(12);
fontSizeMapping.Add(14);
fontSizeMapping.Add(22);
fontSizeMapping.Add(100);

htmlHandler.SetFontSizeMapping(fontSizeMapping);
```

void wxRichTextHTMLHandler::SetTempDir (const wxString & *tempDir*)

Sets the directory for storing temporary files.

If empty, the system temporary directory will be used.

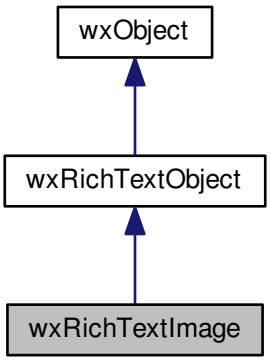
void wxRichTextHTMLHandler::SetTemporaryImageLocations (const wxArrayString & *locations*)

Sets the list of image locations generated by the last operation.

21.630 wxRichTextImage Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextImage:



21.630.1 Detailed Description

This class implements a graphic object.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl, wxRichTextImageBlock

Public Member Functions

- wxRichTextImage (wxRichTextObject *parent=NULL)

  Default constructor.
- wxRichTextImage (const wxImage &image, wxRichTextObject *parent=NULL, wxRichTextAttr *charStyle=NULL)

  Creates a wxRichTextImage from a wxImage.
- wxRichTextImage (const wxRichTextImageBlock &imageBlock, wxRichTextObject *parent=NULL, wxRichTextAttr *charStyle=NULL)

  Creates a wxRichTextImage from an image block.
- wxRichTextImage (const wxRichTextImage &obj)

  Copy constructor.
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

  Lay the item out at the specified position with the given size constraint.
- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.
- virtual wxTextAttrSize GetNaturalSize () const

  Returns the 'natural' size for this object - the image size.
- virtual bool IsEmpty () const

  Returns true if the object is empty.
- virtual bool CanEditProperties () const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxWindow *parent, wxRichTextBuffer *buffer)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel () const

  Returns the label to be used for the properties context menu item.
- virtual bool UsesParagraphAttributes () const

  Returns true if this object takes note of paragraph attributes (text and image objects don't).
- virtual bool ImportFromXML (wxRichTextBuffer *buffer, wxXmlNode *node, wxRichTextXMLHandler *handler, bool *recurse)

  Imports this object from XML.
- virtual bool IsFloatable () const

  Returns true if this class of object is floatable.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- const wxBitmap & GetImageCache () const

  Returns the image cache (a scaled bitmap).

- void SetImageCache (const wxBitmap &bitmap)

  Sets the image cache.
- void ResetImageCache ()

  Resets the image cache.
- wxRichTextImageBlock & GetImageBlock ()

  Returns the image block containing the raw data.
- wxSize GetOriginalImageSize () const

  Gets the original image size.
- void SetOriginalImageSize (const wxSize &sz)

  Sets the original image size.
- void Copy (const wxRichTextImage &obj)

  Copies the image object.
- virtual wxRichTextObject * Clone () const

  Clones the image object.
- virtual bool LoadImageCache (wxDC &dc, wxRichTextDrawingContext &context, wxSize &retImageSize, bool resetCache=false, const wxSize &parentSize=wxDefaultSize)

  Creates a cached image at the required size.
- virtual bool LoadAndScaleImageCache (wxImage &image, const wxSize &sz, bool delayLoading, bool &changed)

  Do the loading and scaling.
- int GetImageState () const

  Gets the image state.
- void SetImageState (int state)

  Sets the image state.

Protected Attributes

- wxRichTextImageBlock m_imageBlock
- wxBitmap m_imageCache
- wxSize m_originalImageSize
- int m_imageState

Additional Inherited Members

21.630.2 Constructor & Destructor Documentation

wxRichTextImage::wxRichTextImage (wxRichTextObject * *parent* = `NULL`) `[inline]`

Default constructor.

wxRichTextImage::wxRichTextImage (const wxImage & *image*, wxRichTextObject * *parent* = `NULL`, wxRichTextAttr * *charStyle* = `NULL`)

Creates a wxRichTextImage from a wxImage.

wxRichTextImage::wxRichTextImage (const wxRichTextImageBlock & *imageBlock*, wxRichTextObject * *parent* = `NULL`, wxRichTextAttr * *charStyle* = `NULL`)

Creates a wxRichTextImage from an image block.

wxRichTextImage::wxRichTextImage (const wxRichTextImage & *obj*) `[inline]`

Copy constructor.

21.630.3 Member Function Documentation

virtual bool wxRichTextImage::CanEditProperties ( ) const `[inline], [virtual]`

Returns true if we can edit the object's properties via a GUI.

Reimplemented from wxRichTextObject.

virtual wxRichTextObject* wxRichTextImage::Clone ( ) const `[inline], [virtual]`

Clones the image object.

Reimplemented from wxRichTextObject.

void wxRichTextImage::Copy (const wxRichTextImage & *obj*)

Copies the image object.

virtual bool wxRichTextImage::Draw (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRichTextRange & *range*, const wxRichTextSelection & *selection*, const wxRect & *rect*, int *descent*, int *style*) `[virtual]`

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implements wxRichTextObject.

virtual bool wxRichTextImage::EditProperties (wxWindow * *parent*, wxRichTextBuffer * *buffer*) `[virtual]`

Edits the object's properties via a GUI.

Reimplemented from wxRichTextObject.

wxRichTextImageBlock& wxRichTextImage::GetImageBlock ( ) `[inline]`

Returns the image block containing the raw data.

const wxBitmap& wxRichTextImage::GetImageCache ( ) const `[inline]`

Returns the image cache (a scaled bitmap).

int wxRichTextImage::GetImageState ( ) const `[inline]`

Gets the image state.

virtual wxTextAttrSize wxRichTextImage::GetNaturalSize ( ) const `[virtual]`

Returns the 'natural' size for this object - the image size.

Reimplemented from wxRichTextObject.

wxSize wxRichTextImage::GetOriginalImageSize ( ) const

Gets the original image size.

virtual wxString wxRichTextImage::GetPropertiesMenuLabel ( ) const `[inline], [virtual]`

Returns the label to be used for the properties context menu item.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextImage::GetRangeSize (const wxRichTextRange & *range,* wxSize & *size,* int & *descent,* wxDC & *dc,* wxRichTextDrawingContext & *context,* int *flags,* const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const `[virtual]`

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Implements wxRichTextObject.

virtual wxString wxRichTextImage::GetXMLNodeName ( ) const `[inline], [virtual]`

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextImage::ImportFromXML (wxRichTextBuffer * *buffer,* wxXmlNode * *node,* wxRichTextXMLHandler * *handler,* bool * *recurse*) `[virtual]`

Imports this object from XML.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextImage::IsEmpty ( ) const `[inline], [virtual]`

Returns true if the object is empty.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextImage::IsFloatable ( ) const `[inline], [virtual]`

Returns true if this class of object is floatable.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextImage::Layout (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style*) `[virtual]`

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implements wxRichTextObject.

virtual bool wxRichTextImage::LoadAndScaleImageCache (wxImage & *image*, const wxSize & *sz*, bool *delayLoading*, bool & *changed*) [virtual]

Do the loading and scaling.

virtual bool wxRichTextImage::LoadImageCache (wxDC & *dc*, wxRichTextDrawingContext & *context*, wxSize & *retImageSize*, bool *resetCache* = false, const wxSize & *parentSize* = wxDefaultSize) [virtual]

Creates a cached image at the required size.

void wxRichTextImage::ResetImageCache ( ) [inline]

Resets the image cache.

void wxRichTextImage::SetImageCache (const wxBitmap & *bitmap*) [inline]

Sets the image cache.

void wxRichTextImage::SetImageState (int *state*) [inline]

Sets the image state.

void wxRichTextImage::SetOriginalImageSize (const wxSize & *sz*)

Sets the original image size.

virtual bool wxRichTextImage::UsesParagraphAttributes ( ) const [inline], [virtual]

Returns true if this object takes note of paragraph attributes (text and image objects don't).

Reimplemented from wxRichTextObject.

21.630.4 Member Data Documentation

wxRichTextImageBlock wxRichTextImage::m_imageBlock [protected]

wxBitmap wxRichTextImage::m_imageCache [protected]

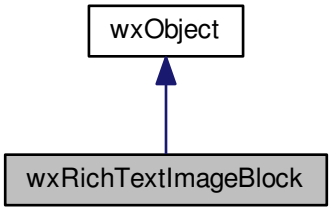
int wxRichTextImage::m_imageState [protected]

wxSize wxRichTextImage::m_originalImageSize [protected]

21.631 wxRichTextImageBlock Class Reference

```
#include <wx/richtext/richtextbuffer.h>
```

Inheritance diagram for wxRichTextImageBlock:



21.631.1 Detailed Description

This class stores information about an image, in binary in-memory form.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextImageBlock ()

  Constructor.
- wxRichTextImageBlock (const wxRichTextImageBlock &block)

  Copy constructor.
- virtual ~wxRichTextImageBlock ()
- void Init ()

  Initialises the block.
- void Clear ()

  Clears the block.
- virtual bool MakeImageBlock (const wxString &filename, wxBitmapType imageType, wxImage &image, bool convertToJPEG=true)

  Load the original image into a memory block.
- virtual bool MakeImageBlock (wxImage &image, wxBitmapType imageType, int quality=80)

  Make an image block from the wxImage in the given format.
- virtual bool MakeImageBlockDefaultQuality (const wxImage &image, wxBitmapType imageType)

  Uses a const wxImage for efficiency, but can't set quality (only relevant for JPEG)
- virtual bool DoMakeImageBlock (const wxImage &image, wxBitmapType imageType)

  Makes the image block.
- bool Write (const wxString &filename)

  Writes the block to a file.
- bool WriteHex (wxOutputStream &stream)

Writes the data in hex to a stream.

- bool ReadHex (wxInputStream &stream, int length, wxBitmapType imageType)

  Reads the data in hex from a stream.
- void Copy (const wxRichTextImageBlock &block)

  Copy from block.
- bool Load (wxImage &image)
- void operator= (const wxRichTextImageBlock &block)

  Assignment operation.
- unsigned char * GetData () const

  Returns the raw data.
- size_t GetDataSize () const

  Returns the data size in bytes.
- wxBitmapType GetImageType () const

  Returns the image type.
- void SetData (unsigned char *image)
- void SetDataSize (size_t size)

  Sets the data size.
- void SetImageType (wxBitmapType imageType)

  Sets the image type.
- bool IsOk () const

  Returns true if the data is non-NULL.
- bool Ok () const
- wxString GetExtension () const

  Gets the extension for the block's type.

Static Public Member Functions

- static unsigned char * ReadBlock (wxInputStream &stream, size_t size)

  Implementation.
- static unsigned char * ReadBlock (const wxString &filename, size_t size)

  Allocates and reads from a file as a block of memory.
- static bool WriteBlock (wxOutputStream &stream, unsigned char *block, size_t size)

  Writes a memory block to stream.
- static bool WriteBlock (const wxString &filename, unsigned char *block, size_t size)

  Writes a memory block to a file.

Protected Attributes

- unsigned char * m_data
- size_t m_dataSize
- wxBitmapType m_imageType

Additional Inherited Members

21.631.2 Constructor & Destructor Documentation

wxRichTextImageBlock::wxRichTextImageBlock ( )

Constructor.

wxRichTextImageBlock::wxRichTextImageBlock (const wxRichTextImageBlock & *block*)

Copy constructor.

virtual wxRichTextImageBlock::~wxRichTextImageBlock ( ) [virtual]

21.631.3 Member Function Documentation

void wxRichTextImageBlock::Clear ( )

Clears the block.

void wxRichTextImageBlock::Copy (const wxRichTextImageBlock & *block*)

Copy from *block*.

virtual bool wxRichTextImageBlock::DoMakeImageBlock (const wxImage & *image,* wxBitmapType *imageType*) [virtual]

Makes the image block.

unsigned char* wxRichTextImageBlock::GetData ( ) const [inline]

Returns the raw data.

size_t wxRichTextImageBlock::GetDataSize ( ) const [inline]

Returns the data size in bytes.

wxString wxRichTextImageBlock::GetExtension ( ) const

Gets the extension for the block's type.

wxBitmapType wxRichTextImageBlock::GetImageType ( ) const [inline]

Returns the image type.

void wxRichTextImageBlock::Init ( )

Initialises the block.

bool wxRichTextImageBlock::IsOk ( ) const [inline]

Returns true if the data is non-NULL.

bool wxRichTextImageBlock::Load (wxImage & *image*)

virtual bool wxRichTextImageBlock::MakeImageBlock (const wxString & *filename,* wxBitmapType *imageType,* wxImage & *image,* bool *convertToJPEG* = `true`) `[virtual]`

Load the original image into a memory block.

If the image is not a JPEG, we must convert it into a JPEG to conserve space. If it's not a JPEG we can make use of *image*, already scaled, so we don't have to load the image a second time.

virtual bool wxRichTextImageBlock::MakeImageBlock (wxImage & *image,* wxBitmapType *imageType,* int *quality* = `80`) `[virtual]`

Make an image block from the wxImage in the given format.

virtual bool wxRichTextImageBlock::MakeImageBlockDefaultQuality (const wxImage & *image,* wxBitmapType *imageType*) `[virtual]`

Uses a const wxImage for efficiency, but can't set quality (only relevant for JPEG)

bool wxRichTextImageBlock::Ok ( ) const `[inline]`

void wxRichTextImageBlock::operator= (const wxRichTextImageBlock & *block*)

Assignment operation.

static unsigned char* wxRichTextImageBlock::ReadBlock (wxInputStream & *stream,* size_t *size*) `[static]`

Implementation.

Allocates and reads from a stream as a block of memory.

static unsigned char* wxRichTextImageBlock::ReadBlock (const wxString & *filename,* size_t *size*) `[static]`

Allocates and reads from a file as a block of memory.

bool wxRichTextImageBlock::ReadHex (wxInputStream & *stream,* int *length,* wxBitmapType *imageType*)

Reads the data in hex from a stream.

void wxRichTextImageBlock::SetData (unsigned char * *image*) `[inline]`

void wxRichTextImageBlock::SetDataSize (size_t *size*) `[inline]`

Sets the data size.

void wxRichTextImageBlock::SetImageType (wxBitmapType *imageType*) `[inline]`

Sets the image type.

bool wxRichTextImageBlock::Write (const wxString & *filename*)

Writes the block to a file.

static bool wxRichTextImageBlock::WriteBlock (wxOutputStream & *stream*, unsigned char * *block*, size_t *size*) `[static]`

Writes a memory block to stream.

static bool wxRichTextImageBlock::WriteBlock (const wxString & *filename*, unsigned char * *block*, size_t *size*) `[static]`

Writes a memory block to a file.

bool wxRichTextImageBlock::WriteHex (wxOutputStream & *stream*)

Writes the data in hex to a stream.

21.631.4 Member Data Documentation

unsigned char* wxRichTextImageBlock::m_data `[protected]`

size_t wxRichTextImageBlock::m_dataSize `[protected]`

wxBitmapType wxRichTextImageBlock::m_imageType `[protected]`

21.632 wxRichTextLine Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.632.1 Detailed Description

This object represents a line in a paragraph, and stores offsets from the start of the paragraph representing the start and end positions of the line.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextLine (wxRichTextParagraph *parent)
- wxRichTextLine (const wxRichTextLine &obj)
- virtual ~wxRichTextLine ()
- void SetRange (const wxRichTextRange &range)

  Sets the range associated with this line.
- void SetRange (long from, long to)

  Sets the range associated with this line.
- wxRichTextParagraph * GetParent ()

  Returns the parent paragraph.

- const wxRichTextRange & GetRange () const

  Returns the range.
- wxRichTextRange & GetRange ()

  Returns the range.
- wxRichTextRange GetAbsoluteRange () const

  Returns the absolute range.
- virtual wxSize GetSize () const

  Returns the line size as calculated by Layout.
- virtual void SetSize (const wxSize &sz)

  Sets the line size as calculated by Layout.
- virtual wxPoint GetPosition () const

  Returns the object position relative to the parent.
- virtual void SetPosition (const wxPoint &pos)

  Sets the object position relative to the parent.
- virtual wxPoint GetAbsolutePosition () const

  Returns the absolute object position.
- virtual wxRect GetRect () const

  Returns the rectangle enclosing the line.
- void SetDescent (int descent)

  Sets the stored descent.
- int GetDescent () const

  Returns the stored descent.
- void Init (wxRichTextParagraph *parent)

  Initialises the object.
- void Copy (const wxRichTextLine &obj)

  Copies from obj.
- virtual wxRichTextLine * Clone () const

Protected Attributes

- wxRichTextRange m_range
- wxPoint m_pos
- wxSize m_size
- int m_descent
- wxRichTextParagraph * m_parent

21.632.2 Constructor & Destructor Documentation

wxRichTextLine::wxRichTextLine (wxRichTextParagraph * *parent*)

wxRichTextLine::wxRichTextLine (const wxRichTextLine & *obj*) `[inline]`

virtual wxRichTextLine::~wxRichTextLine ( ) `[inline]`, `[virtual]`

21.632.3 Member Function Documentation

virtual wxRichTextLine* wxRichTextLine::Clone ( ) const `[inline]`, `[virtual]`

void wxRichTextLine::Copy (const wxRichTextLine & *obj*)

Copies from *obj*.

virtual wxPoint wxRichTextLine::GetAbsolutePosition ( ) const [virtual]

Returns the absolute object position.

wxRichTextRange wxRichTextLine::GetAbsoluteRange ( ) const

Returns the absolute range.

int wxRichTextLine::GetDescent ( ) const [inline]

Returns the stored descent.

wxRichTextParagraph* wxRichTextLine::GetParent ( ) [inline]

Returns the parent paragraph.

virtual wxPoint wxRichTextLine::GetPosition ( ) const [inline], [virtual]

Returns the object position relative to the parent.

const wxRichTextRange& wxRichTextLine::GetRange ( ) const [inline]

Returns the range.

wxRichTextRange& wxRichTextLine::GetRange ( ) [inline]

Returns the range.

virtual wxRect wxRichTextLine::GetRect ( ) const [inline], [virtual]

Returns the rectangle enclosing the line.

virtual wxSize wxRichTextLine::GetSize ( ) const [inline], [virtual]

Returns the line size as calculated by Layout.

void wxRichTextLine::Init (wxRichTextParagraph * *parent*)

Initialises the object.

void wxRichTextLine::SetDescent (int *descent*) [inline]

Sets the stored descent.

virtual void wxRichTextLine::SetPosition (const wxPoint & *pos*) [inline], [virtual]

Sets the object position relative to the parent.

void wxRichTextLine::SetRange (const wxRichTextRange & *range*) `[inline]`

Sets the range associated with this line.

void wxRichTextLine::SetRange (long *from,* long *to*) `[inline]`

Sets the range associated with this line.

virtual void wxRichTextLine::SetSize (const wxSize & *sz*) `[inline], [virtual]`

Sets the line size as calculated by Layout.

21.632.4 Member Data Documentation

int wxRichTextLine::m_descent `[protected]`

wxRichTextParagraph* wxRichTextLine::m_parent `[protected]`

wxPoint wxRichTextLine::m_pos `[protected]`

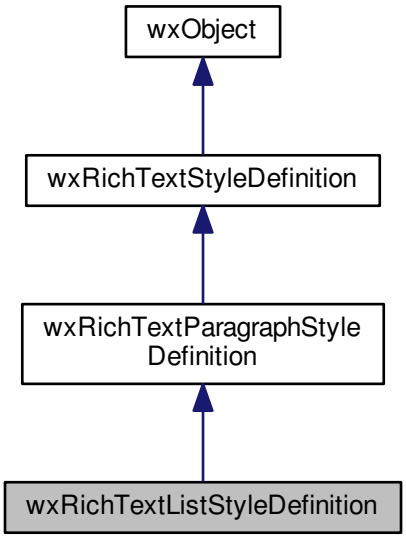
wxRichTextRange wxRichTextLine::m_range `[protected]`

wxSize wxRichTextLine::m_size `[protected]`

21.633 wxRichTextListStyleDefinition Class Reference

`#include <wx/richtext/richtextstyles.h>`

Inheritance diagram for wxRichTextListStyleDefinition:



21.633.1 Detailed Description

This class represents a list style definition, usually added to a wxRichTextStyleSheet.

The class inherits paragraph attributes from wxRichTextStyleParagraphDefinition, and adds 10 further attribute objects, one for each level of a list. When applying a list style to a paragraph, the list style's base and appropriate level attributes are merged with the paragraph's existing attributes.

You can apply a list style to one or more paragraphs using wxRichTextCtrl::SetListStyle. You can also use the functions wxRichTextCtrl::NumberList, wxRichTextCtrl::PromoteList and wxRichTextCtrl::ClearListStyle.

As usual, there are wxRichTextBuffer versions of these functions so that you can apply them directly to a buffer without requiring a control.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextListStyleDefinition (const wxString &name=wxEmptyString)

  Constructor.
- virtual ~wxRichTextListStyleDefinition ()

  Destructor.
- wxRichTextAttr CombineWithParagraphStyle (int indent, const wxRichTextAttr &paraStyle, wxRichTextStyleSheet *styleSheet=NULL)

  This function combines the given paragraph style with the list style's base attributes and level style matching the given indent, returning the combined attributes.
- int FindLevelForIndent (int indent) const

  This function finds the level (from 0 to 9) whose indentation attribute mostly closely matches indent (expressed in tenths of a millimetre).
- wxRichTextAttr GetCombinedStyle (int indent, wxRichTextStyleSheet *styleSheet=NULL)

  This function combines the list style's base attributes and the level style matching the given indent, returning the combined attributes.
- wxRichTextAttr GetCombinedStyleForLevel (int level, wxRichTextStyleSheet *styleSheet=NULL)

  This function combines the list style's base attributes and the style for the specified level, returning the combined attributes.
- const wxRichTextAttr * GetLevelAttributes (int level) const

  Returns the style for the given level.
- int GetLevelCount () const

  Returns the number of levels.
- bool IsNumbered (int level) const

  Returns true if the given level has numbered list attributes.
- void SetLevelAttributes (int level, const wxRichTextAttr &attr)

  Sets the style for the given level.

Additional Inherited Members

21.633.2 Constructor & Destructor Documentation

wxRichTextListStyleDefinition::wxRichTextListStyleDefinition (const wxString & *name* = wxEmptyString)

Constructor.

virtual wxRichTextListStyleDefinition::~wxRichTextListStyleDefinition ( ) [virtual]

Destructor.

21.633.3 Member Function Documentation

wxRichTextAttr wxRichTextListStyleDefinition::CombineWithParagraphStyle (int *indent,* const wxRichTextAttr & *paraStyle,* wxRichTextStyleSheet * *styleSheet* = NULL)

This function combines the given paragraph style with the list style's base attributes and level style matching the given indent, returning the combined attributes.

If *styleSheet* is specified, the base style for this definition will also be included in the result.

int wxRichTextListStyleDefinition::FindLevelForIndent (int *indent*) const

This function finds the level (from 0 to 9) whose indentation attribute mostly closely matches *indent* (expressed in tenths of a millimetre).

wxRichTextAttr wxRichTextListStyleDefinition::GetCombinedStyle (int *indent,* wxRichTextStyleSheet * *styleSheet* = NULL)

This function combines the list style's base attributes and the level style matching the given indent, returning the combined attributes.

If *styleSheet* is specified, the base style for this definition will also be included in the result.

wxRichTextAttr wxRichTextListStyleDefinition::GetCombinedStyleForLevel (int *level,* wxRichTextStyleSheet * *styleSheet* = NULL)

This function combines the list style's base attributes and the style for the specified level, returning the combined attributes.

If *styleSheet* is specified, the base style for this definition will also be included in the result.

const wxRichTextAttr* wxRichTextListStyleDefinition::GetLevelAttributes (int *level*) const

Returns the style for the given level.

level is a number between 0 and 9.

int wxRichTextListStyleDefinition::GetLevelCount ( ) const

Returns the number of levels.

This is hard-wired to 10. Returns the style for the given level. *level* is a number between 0 and 9.

bool wxRichTextListStyleDefinition::IsNumbered (int *level*) const

Returns true if the given level has numbered list attributes.

void wxRichTextListStyleDefinition::SetLevelAttributes (int *level,* const wxRichTextAttr & *attr*)

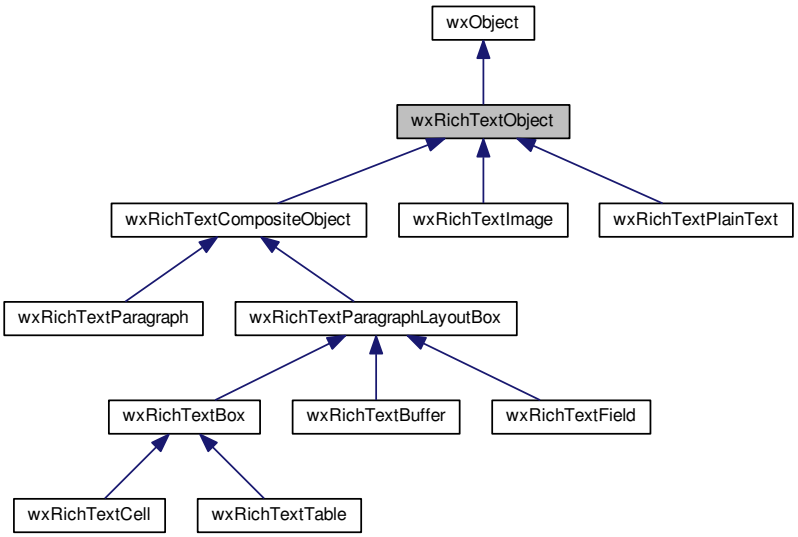
Sets the style for the given level.

level is a number between 0 and 9. The first and most flexible form uses a wxTextAttr object, while the second form is for convenient setting of the most commonly-used attributes.

21.634 wxRichTextObject Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextObject:



21.634.1 Detailed Description

This is the base for drawable rich text objects.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextObject (wxRichTextObject *parent=NULL)

  Constructor, taking an optional parent pointer.
- virtual ~wxRichTextObject ()
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)=0

  Draw the item, within the given range.

- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)=0

  Lay the item out at the specified position with the given size constraint.
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

  Hit-testing: returns a flag indicating hit test details, plus information about position.
- virtual bool FindPosition (wxDC &dc, wxRichTextDrawingContext &context, long index, wxPoint &pt, int *height, bool forceLineStart)

  Finds the absolute position and row height for the given character position.
- virtual wxSize GetBestSize () const

  Returns the best size, i.e. the ideal starting size for this object irrespective of available space.
- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const =0

  Returns the object size for the given range.
- virtual wxRichTextObject * DoSplit (long pos)

  Do a split from pos, returning an object containing the second part, and setting the first part in 'this'.
- virtual void CalculateRange (long start, long &end)

  Calculates the range of the object.
- virtual bool DeleteRange (const wxRichTextRange &range)

  Deletes the given range.
- virtual bool IsEmpty () const

  Returns true if the object is empty.
- virtual bool IsFloatable () const

  Returns true if this class of object is floatable.
- virtual bool IsFloating () const

  Returns true if this object is currently floating.
- virtual int GetFloatDirection () const

  Returns the floating direction.
- virtual wxString GetTextForRange (const wxRichTextRange &range) const

  Returns any text in this object for the given range.
- virtual bool CanMerge (wxRichTextObject *object, wxRichTextDrawingContext &context) const

  Returns true if this object can merge itself with the given one.
- virtual bool Merge (wxRichTextObject *object, wxRichTextDrawingContext &context)

  Returns true if this object merged itself with the given one.
- virtual bool CanSplit (wxRichTextDrawingContext &context) const

  Returns true if this object can potentially be split, by virtue of having different virtual attributes for individual sub-objects.
- virtual wxRichTextObject * Split (wxRichTextDrawingContext &context)

  Returns the final object in the split objects if this object was split due to differences between sub-object virtual attributes.
- virtual void Dump (wxTextOutputStream &stream)

  Dump object data to the given output stream for debugging.
- virtual bool CanEditProperties () const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxWindow *parent, wxRichTextBuffer *buffer)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel () const

  Returns the label to be used for the properties context menu item.
- virtual bool AcceptsFocus () const

  Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.

- virtual bool ImportFromXML (wxRichTextBuffer *buffer, wxXmlNode *node, wxRichTextXMLHandler *handler, bool *recurse)

  Imports this object from XML.
- virtual bool UsesParagraphAttributes () const

  Returns true if this object takes note of paragraph attributes (text and image objects don't).
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- virtual void Invalidate (const wxRichTextRange &invalidRange=wxRICHTEXT_ALL)

  Invalidates the object at the given range.
- virtual bool HandlesChildSelections () const

  Returns true if this object can handle the selections of its children, fOr example a table.
- virtual wxRichTextSelection GetSelection (long start, long end) const

  Returns a selection object specifying the selections between start and end character positions.
- virtual wxSize GetCachedSize () const

  Gets the cached object size as calculated by Layout.
- virtual void SetCachedSize (const wxSize &sz)

  Sets the cached object size as calculated by Layout.
- virtual wxSize GetMaxSize () const

  Gets the maximum object size as calculated by Layout.
- virtual void SetMaxSize (const wxSize &sz)

  Sets the maximum object size as calculated by Layout.
- virtual wxSize GetMinSize () const

  Gets the minimum object size as calculated by Layout.
- virtual void SetMinSize (const wxSize &sz)

  Sets the minimum object size as calculated by Layout.
- virtual wxTextAttrSize GetNaturalSize () const

  Gets the 'natural' size for an object.
- virtual wxPoint GetPosition () const

  Returns the object position in pixels.
- virtual void SetPosition (const wxPoint &pos)

  Sets the object position in pixels.
- virtual wxPoint GetAbsolutePosition () const

  Returns the absolute object position, by traversing up the child/parent hierarchy.
- virtual wxRect GetRect () const

  Returns the rectangle enclosing the object.
- void SetRange (const wxRichTextRange &range)

  Sets the object's range within its container.
- const wxRichTextRange & GetRange () const

  Returns the object's range.
- wxRichTextRange & GetRange ()

  Returns the object's range.
- void SetOwnRange (const wxRichTextRange &range)

  Set the object's own range, for a top-level object with its own position space.
- const wxRichTextRange & GetOwnRange () const

  Returns the object's own range (valid if top-level).
- wxRichTextRange & GetOwnRange ()

  Returns the object's own range (valid if top-level).
- wxRichTextRange GetOwnRangeIfTopLevel () const

  Returns the object's own range only if a top-level object.
- virtual bool IsComposite () const

  Returns true if this object is composite.

- virtual bool IsAtomic () const

  Returns true if no user editing can be done inside the object.
- virtual wxRichTextObject * GetParent () const

  Returns a pointer to the parent object.
- virtual void SetParent (wxRichTextObject *parent)

  Sets the pointer to the parent object.
- virtual wxRichTextParagraphLayoutBox * GetContainer () const

  Returns the top-level container of this object.
- virtual wxRichTextParagraphLayoutBox * GetParentContainer () const

  Returns the top-level container of this object.
- virtual void SetMargins (int margin)

  Set the margin around the object, in pixels.
- virtual void SetMargins (int leftMargin, int rightMargin, int topMargin, int bottomMargin)

  Set the margin around the object, in pixels.
- virtual int GetLeftMargin () const

  Returns the left margin of the object, in pixels.
- virtual int GetRightMargin () const

  Returns the right margin of the object, in pixels.
- virtual int GetTopMargin () const

  Returns the top margin of the object, in pixels.
- virtual int GetBottomMargin () const

  Returns the bottom margin of the object, in pixels.
- virtual wxRect GetAvailableContentArea (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &outerRect) const

  Calculates the available content space in the given rectangle, given the margins, border and padding specified in the object's attributes.
- virtual bool LayoutToBestSize (wxDC &dc, wxRichTextDrawingContext &context, wxRichTextBuffer *buffer, const wxRichTextAttr &parentAttr, const wxRichTextAttr &attr, const wxRect &availableParentSpace, const wxRect &availableContainerSpace, int style)

  Lays out the object first with a given amount of space, and then if no width was specified in attr, lays out the object again using the minimum size.
- virtual bool AdjustAttributes (wxRichTextAttr &attr, wxRichTextDrawingContext &context)

  Adjusts the attributes for virtual attribute provision, collapsed borders, etc.
- void SetAttributes (const wxRichTextAttr &attr)

  Sets the object's attributes.
- const wxRichTextAttr & GetAttributes () const

  Returns the object's attributes.
- wxRichTextAttr & GetAttributes ()

  Returns the object's attributes.
- wxRichTextProperties & GetProperties ()

  Returns the object's properties.
- const wxRichTextProperties & GetProperties () const

  Returns the object's properties.
- void SetProperties (const wxRichTextProperties &props)

  Sets the object's properties.
- void SetDescent (int descent)

  Sets the stored descent value.
- int GetDescent () const

  Returns the stored descent value.
- wxRichTextBuffer * GetBuffer () const

Returns the containing buffer.

- void SetName (const wxString &name)

  Sets the identifying name for this object as a property using the "name" key.

- wxString GetName () const

  Returns the identifying name for this object from the properties, using the "name" key.

- virtual bool IsTopLevel () const

  Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.

- bool IsShown () const

  Returns true if the object will be shown, false otherwise.

- virtual void Show (bool show)

  Call to show or hide this object.

- virtual wxRichTextObject * Clone () const

  Clones the object.

- void Copy (const wxRichTextObject &obj)

  Copies the object.

- void Reference ()

  Reference-counting allows us to use the same object in multiple lists (not yet used).

- void Dereference ()

  Reference-counting allows us to use the same object in multiple lists (not yet used).

- virtual void Move (const wxPoint &pt)

  Moves the object recursively, by adding the offset from old to new.

- int ConvertTenthsMMToPixels (wxDC &dc, int units) const

  Converts units in tenths of a millimetre to device units.

- int ConvertPixelsToTenthsMM (wxDC &dc, int pixels) const

  Convert units in pixels to tenths of a millimetre.

Static Public Member Functions

- static int ConvertTenthsMMToPixels (int ppi, int units, double scale=1.0)

  Converts units in tenths of a millimetre to device units.

- static int ConvertPixelsToTenthsMM (int ppi, int pixels, double scale=1.0)

  Convert units in pixels to tenths of a millimetre.

- static bool DrawBoxAttributes (wxDC &dc, wxRichTextBuffer *buffer, const wxRichTextAttr &attr, const wxRect &boxRect, int flags=0, wxRichTextObject *obj=NULL)

  Draws the borders and background for the given rectangle and attributes.

- static bool DrawBorder (wxDC &dc, wxRichTextBuffer *buffer, const wxRichTextAttr &attr, const wxTextAttr↩ Borders &borders, const wxRect &rect, int flags=0)

  Draws a border.

- static bool GetBoxRects (wxDC &dc, wxRichTextBuffer *buffer, const wxRichTextAttr &attr, wxRect &margin↩ Rect, wxRect &borderRect, wxRect &contentRect, wxRect &paddingRect, wxRect &outlineRect)

  Returns the various rectangles of the box model in pixels.

- static bool GetTotalMargin (wxDC &dc, wxRichTextBuffer *buffer, const wxRichTextAttr &attr, int &leftMargin, int &rightMargin, int &topMargin, int &bottomMargin)

  Returns the total margin for the object in pixels, taking into account margin, padding and border size.

- static wxRect AdjustAvailableSpace (wxDC &dc, wxRichTextBuffer *buffer, const wxRichTextAttr &parentAttr, const wxRichTextAttr &childAttr, const wxRect &availableParentSpace, const wxRect &availableContainer↩ Space)

  Returns the rectangle which the child has available to it given restrictions specified in the child attribute, e.g.

Protected Attributes

- wxSize m_size
- wxSize m_maxSize
- wxSize m_minSize
- wxPoint m_pos
- int m_descent
- int m_refCount
- bool m_show
- wxRichTextObject * m_parent
- wxRichTextRange m_range
- wxRichTextRange m_ownRange
- wxRichTextAttr m_attributes
- wxRichTextProperties m_properties

Additional Inherited Members

21.634.2 Constructor & Destructor Documentation

wxRichTextObject::wxRichTextObject (wxRichTextObject * *parent* = NULL)

Constructor, taking an optional parent pointer.

virtual wxRichTextObject::~wxRichTextObject ( ) [virtual]

21.634.3 Member Function Documentation

virtual bool wxRichTextObject::AcceptsFocus ( ) const [virtual]

Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.

For example, containers supporting text, such as a text box object, can accept the focus, but a table can't (set the focus to individual cells instead).

Reimplemented in wxRichTextTable, wxRichTextField, and wxRichTextParagraphLayoutBox.

virtual bool wxRichTextObject::AdjustAttributes (wxRichTextAttr & *attr,* wxRichTextDrawingContext & *context*) [virtual]

Adjusts the attributes for virtual attribute provision, collapsed borders, etc.

static wxRect wxRichTextObject::AdjustAvailableSpace (wxDC & *dc,* wxRichTextBuffer * *buffer,* const wxRichTextAttr & *parentAttr,* const wxRichTextAttr & *childAttr,* const wxRect & *availableParentSpace,* const wxRect & *availableContainerSpace*) [static]

Returns the rectangle which the child has available to it given restrictions specified in the child attribute, e.g.

50% width of the parent, 400 pixels, x position 20% of the parent, etc. availableContainerSpace might be a parent that the cell has to compute its width relative to. E.g. a cell that's 50% of its parent.

virtual void wxRichTextObject::CalculateRange (long *start,* long & *end*) [virtual]

Calculates the range of the object.

By default, guess that the object is 1 unit long.

Reimplemented in wxRichTextTable, wxRichTextPlainText, wxRichTextParagraph, wxRichTextField, and wxRichTextCompositeObject.

virtual bool wxRichTextObject::CanEditProperties ( ) const `[virtual]`

Returns true if we can edit the object's properties via a GUI.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextImage, wxRichTextField, and wxRichTextBox.

virtual bool wxRichTextObject::CanMerge (wxRichTextObject * *object,* wxRichTextDrawingContext & *context*) const `[virtual]`

Returns true if this object can merge itself with the given one.

Reimplemented in wxRichTextPlainText.

virtual bool wxRichTextObject::CanSplit (wxRichTextDrawingContext & *context*) const `[virtual]`

Returns true if this object can potentially be split, by virtue of having different virtual attributes for individual sub-objects.

Reimplemented in wxRichTextPlainText.

virtual wxRichTextObject* wxRichTextObject::Clone ( ) const `[virtual]`

Clones the object.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextBuffer, wxRichTextImage, wxRichTextPlainText, wxRichTextParagraph, wxRichTextField, wxRichTextBox, and wxRichTextParagraphLayoutBox.

int wxRichTextObject::ConvertPixelsToTenthsMM (wxDC & *dc,* int *pixels*) const

Convert units in pixels to tenths of a millimetre.

static int wxRichTextObject::ConvertPixelsToTenthsMM (int *ppi,* int *pixels,* double *scale* = `1.0`) `[static]`

Convert units in pixels to tenths of a millimetre.

int wxRichTextObject::ConvertTenthsMMToPixels (wxDC & *dc,* int *units*) const

Converts units in tenths of a millimetre to device units.

static int wxRichTextObject::ConvertTenthsMMToPixels (int *ppi,* int *units,* double *scale* = `1.0`) `[static]`

Converts units in tenths of a millimetre to device units.

void wxRichTextObject::Copy (const wxRichTextObject & *obj*)

Copies the object.

virtual bool wxRichTextObject::DeleteRange (const wxRichTextRange & *range*) [virtual]

Deletes the given range.

Reimplemented in wxRichTextTable, wxRichTextPlainText, wxRichTextParagraphLayoutBox, and wxRichTextCompositeObject.

void wxRichTextObject::Dereference ( )

Reference-counting allows us to use the same object in multiple lists (not yet used).

virtual wxRichTextObject* wxRichTextObject::DoSplit (long *pos*) [virtual]

Do a split from *pos*, returning an object containing the second part, and setting the first part in 'this'.

Reimplemented in wxRichTextPlainText.

virtual bool wxRichTextObject::Draw (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) [pure virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implemented in wxRichTextTable, wxRichTextCell, wxRichTextImage, wxRichTextPlainText, wxRichTextParagraph, wxRichTextField, wxRichTextBox, and wxRichTextParagraphLayoutBox.

static bool wxRichTextObject::DrawBorder (wxDC & *dc,* wxRichTextBuffer * *buffer,* const wxRichTextAttr & *attr,* const wxTextAttrBorders & *borders,* const wxRect & *rect,* int *flags* = 0) [static]

Draws a border.

static bool wxRichTextObject::DrawBoxAttributes (wxDC & *dc,* wxRichTextBuffer * *buffer,* const wxRichTextAttr & *attr,* const wxRect & *boxRect,* int *flags* = 0, wxRichTextObject * *obj* = NULL) [static]

Draws the borders and background for the given rectangle and attributes.

boxRect is taken to be the outer margin box, not the box around the content.

virtual void wxRichTextObject::Dump (wxTextOutputStream & *stream*) [virtual]

Dump object data to the given output stream for debugging.

Reimplemented in wxRichTextBuffer, wxRichTextPlainText, and wxRichTextCompositeObject.

virtual bool wxRichTextObject::EditProperties (wxWindow * *parent,* wxRichTextBuffer * *buffer*) [virtual]

Edits the object's properties via a GUI.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextImage, wxRichTextField, and wxRichTextBox.

virtual bool wxRichTextObject::FindPosition (wxDC & *dc,* wxRichTextDrawingContext & *context,* long *index,* wxPoint & *pt,* int * *height,* bool *forceLineStart*) [virtual]

Finds the absolute position and row height for the given character position.

Reimplemented in wxRichTextTable, wxRichTextParagraph, and wxRichTextCompositeObject.

virtual wxPoint wxRichTextObject::GetAbsolutePosition ( ) const [virtual]

Returns the absolute object position, by traversing up the child/parent hierarchy.

TODO: may not be needed, if all object positions are in fact relative to the top of the coordinate space.

const wxRichTextAttr& wxRichTextObject::GetAttributes ( ) const

Returns the object's attributes.

wxRichTextAttr& wxRichTextObject::GetAttributes ( )

Returns the object's attributes.

virtual wxRect wxRichTextObject::GetAvailableContentArea (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *outerRect*) const [virtual]

Calculates the available content space in the given rectangle, given the margins, border and padding specified in the object's attributes.

virtual wxSize wxRichTextObject::GetBestSize ( ) const [virtual]

Returns the best size, i.e. the ideal starting size for this object irrespective of available space.

For a short text string, it will be the size that exactly encloses the text. For a longer string, it might use the parent width for example.

virtual int wxRichTextObject::GetBottomMargin ( ) const [virtual]

Returns the bottom margin of the object, in pixels.

static bool wxRichTextObject::GetBoxRects (wxDC & *dc,* wxRichTextBuffer * *buffer,* const wxRichTextAttr & *attr,* wxRect & *marginRect,* wxRect & *borderRect,* wxRect & *contentRect,* wxRect & *paddingRect,* wxRect & *outlineRect*) [static]

Returns the various rectangles of the box model in pixels.

You can either specify *contentRect* (inner) or *marginRect* (outer), and the other must be the default rectangle (no width or height). Note that the outline doesn't affect the position of the rectangle, it's drawn in whatever space is available.

wxRichTextBuffer* wxRichTextObject::GetBuffer ( ) const

Returns the containing buffer.

virtual wxSize wxRichTextObject::GetCachedSize ( ) const [virtual]

Gets the cached object size as calculated by Layout.

virtual wxRichTextParagraphLayoutBox* wxRichTextObject::GetContainer ( ) const [virtual]

Returns the top-level container of this object.

May return itself if it's a container; use GetParentContainer to return a different container.

int wxRichTextObject::GetDescent ( ) const

Returns the stored descent value.

virtual int wxRichTextObject::GetFloatDirection ( ) const [virtual]

Returns the floating direction.

virtual int wxRichTextObject::GetLeftMargin ( ) const [virtual]

Returns the left margin of the object, in pixels.

virtual wxSize wxRichTextObject::GetMaxSize ( ) const [virtual]

Gets the maximum object size as calculated by Layout.

This allows us to fit an object to its contents or allocate extra space if required.

virtual wxSize wxRichTextObject::GetMinSize ( ) const [virtual]

Gets the minimum object size as calculated by Layout.

This allows us to constrain an object to its absolute minimum size if necessary.

wxString wxRichTextObject::GetName ( ) const

Returns the identifying name for this object from the properties, using the "name" key.

virtual wxTextAttrSize wxRichTextObject::GetNaturalSize ( ) const [virtual]

Gets the 'natural' size for an object.

For an image, it would be the image size.

Reimplemented in wxRichTextImage.

const wxRichTextRange& wxRichTextObject::GetOwnRange ( ) const

Returns the object's own range (valid if top-level).

wxRichTextRange& wxRichTextObject::GetOwnRange ( )

Returns the object's own range (valid if top-level).

wxRichTextRange wxRichTextObject::GetOwnRangeIfTopLevel ( ) const

Returns the object's own range only if a top-level object.

virtual wxRichTextObject* wxRichTextObject::GetParent ( ) const [virtual]

Returns a pointer to the parent object.

virtual wxRichTextParagraphLayoutBox* wxRichTextObject::GetParentContainer ( ) const [virtual]

Returns the top-level container of this object.

Returns a different container than itself, unless there's no parent, in which case it will return NULL.

virtual wxPoint wxRichTextObject::GetPosition ( ) const [virtual]

Returns the object position in pixels.

wxRichTextProperties& wxRichTextObject::GetProperties ( )

Returns the object's properties.

const wxRichTextProperties& wxRichTextObject::GetProperties ( ) const

Returns the object's properties.

virtual wxString wxRichTextObject::GetPropertiesMenuLabel ( ) const [virtual]

Returns the label to be used for the properties context menu item.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextImage, wxRichTextField, and wxRichTextBox.

const wxRichTextRange& wxRichTextObject::GetRange ( ) const

Returns the object's range.

wxRichTextRange& wxRichTextObject::GetRange ( )

Returns the object's range.

virtual bool wxRichTextObject::GetRangeSize (const wxRichTextRange & *range,* wxSize & *size,* int & *descent,* wxDC & *dc,* wxRichTextDrawingContext & *context,* int *flags,* const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [pure virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Implemented in wxRichTextTable, wxRichTextImage, wxRichTextPlainText, wxRichTextParagraph, wxRichTextField, wxRichTextParagraphLayoutBox, and wxRichTextCompositeObject.

virtual wxRect wxRichTextObject::GetRect ( ) const [virtual]

Returns the rectangle enclosing the object.

virtual int wxRichTextObject::GetRightMargin ( ) const [virtual]

Returns the right margin of the object, in pixels.

virtual wxRichTextSelection wxRichTextObject::GetSelection (long *start*, long *end*) const [virtual]

Returns a selection object specifying the selections between start and end character positions.

For example, a table would deduce what cells (of range length 1) are selected when dragging across the table.

Reimplemented in wxRichTextTable.

virtual wxString wxRichTextObject::GetTextForRange (const wxRichTextRange & *range*) const [virtual]

Returns any text in this object for the given range.

Reimplemented in wxRichTextTable, wxRichTextPlainText, wxRichTextParagraphLayoutBox, and wxRichTextCompositeObject.

virtual int wxRichTextObject::GetTopMargin ( ) const [virtual]

Returns the top margin of the object, in pixels.

static bool wxRichTextObject::GetTotalMargin (wxDC & *dc*, wxRichTextBuffer * *buffer*, const wxRichTextAttr & *attr*, int & *leftMargin*, int & *rightMargin*, int & *topMargin*, int & *bottomMargin*) [static]

Returns the total margin for the object in pixels, taking into account margin, padding and border size.

virtual wxString wxRichTextObject::GetXMLNodeName ( ) const [virtual]

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextImage, wxRichTextPlainText, wxRichTextParagraph, wxRichTextField, wxRichTextBox, and wxRichTextParagraphLayoutBox.

virtual bool wxRichTextObject::HandlesChildSelections ( ) const [virtual]

Returns true if this object can handle the selections of its children, fOr example a table.

Required for composite selection handling to work.

Reimplemented in wxRichTextTable.

virtual int wxRichTextObject::HitTest (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxPoint & *pt*, long & *textPosition*, wxRichTextObject ** *obj*, wxRichTextObject ** *contextObj*, int *flags* = 0) [virtual]

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextBuffer, wxRichTextParagraph, wxRichTextParagraphLayoutBox, and wxRichTextCompositeObject.

virtual bool wxRichTextObject::ImportFromXML (wxRichTextBuffer * *buffer,* wxXmlNode * *node,* wxRichTextXMLHandler * *handler,* bool * *recurse*) [virtual]

Imports this object from XML.

Reimplemented in wxRichTextTable, wxRichTextImage, wxRichTextPlainText, and wxRichTextParagraphLayoutBox.

virtual void wxRichTextObject::Invalidate (const wxRichTextRange & *invalidRange* = wxRICHTEXT_ALL) [virtual]

Invalidates the object at the given range.

With no argument, invalidates the whole object.

Reimplemented in wxRichTextParagraphLayoutBox, and wxRichTextCompositeObject.

virtual bool wxRichTextObject::IsAtomic ( ) const [virtual]

Returns true if no user editing can be done inside the object.

This returns true for simple objects, false for most composite objects, but true for fields, which if composite, should not be user-edited.

Reimplemented in wxRichTextField, and wxRichTextCompositeObject.

virtual bool wxRichTextObject::IsComposite ( ) const [virtual]

Returns true if this object is composite.

Reimplemented in wxRichTextCompositeObject.

virtual bool wxRichTextObject::IsEmpty ( ) const [virtual]

Returns true if the object is empty.

Reimplemented in wxRichTextImage, wxRichTextPlainText, wxRichTextField, and wxRichTextCompositeObject.

virtual bool wxRichTextObject::IsFloatable ( ) const [virtual]

Returns true if this class of object is floatable.

Reimplemented in wxRichTextImage.

virtual bool wxRichTextObject::IsFloating ( ) const [virtual]

Returns true if this object is currently floating.

bool wxRichTextObject::IsShown ( ) const

Returns true if the object will be shown, false otherwise.

virtual bool wxRichTextObject::IsTopLevel ( ) const [virtual]

Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.

Reimplemented in wxRichTextField, and wxRichTextParagraphLayoutBox.

virtual bool wxRichTextObject::Layout (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style*) [pure virtual]

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implemented in wxRichTextTable, wxRichTextImage, wxRichTextPlainText, wxRichTextParagraph, wxRichTextField, and wxRichTextParagraphLayoutBox.

virtual bool wxRichTextObject::LayoutToBestSize (wxDC & *dc,* wxRichTextDrawingContext & *context,* wxRichTextBuffer * *buffer,* const wxRichTextAttr & *parentAttr,* const wxRichTextAttr & *attr,* const wxRect & *availableParentSpace,* const wxRect & *availableContainerSpace,* int *style*) [virtual]

Lays out the object first with a given amount of space, and then if no width was specified in attr, lays out the object again using the minimum size.

availableParentSpace is the maximum space for the object, whereas *availableContainerSpace* is the container with which relative positions and sizes should be computed. For example, a text box whose space has already been constrained in a previous layout pass to *availableParentSpace*, but should have a width of 50% of *availableContainerSpace*. (If these two rects were the same, a 2nd pass could see the object getting too small.)

virtual bool wxRichTextObject::Merge (wxRichTextObject * *object,* wxRichTextDrawingContext & *context*) [virtual]

Returns true if this object merged itself with the given one.

The calling code will then delete the given object.

Reimplemented in wxRichTextPlainText.

virtual void wxRichTextObject::Move (const wxPoint & *pt*) [virtual]

Moves the object recursively, by adding the offset from old to new.

Reimplemented in wxRichTextCompositeObject.

void wxRichTextObject::Reference ( )

Reference-counting allows us to use the same object in multiple lists (not yet used).

void wxRichTextObject::SetAttributes (const wxRichTextAttr & *attr*)

Sets the object's attributes.

virtual void wxRichTextObject::SetCachedSize (const wxSize & *sz*) [virtual]

Sets the cached object size as calculated by Layout.

void wxRichTextObject::SetDescent (int *descent*)

Sets the stored descent value.

virtual void wxRichTextObject::SetMargins (int *margin*) [virtual]

Set the margin around the object, in pixels.

virtual void wxRichTextObject::SetMargins (int *leftMargin,* int *rightMargin,* int *topMargin,* int *bottomMargin*) [virtual]

Set the margin around the object, in pixels.

virtual void wxRichTextObject::SetMaxSize (const wxSize & *sz*) [virtual]

Sets the maximum object size as calculated by Layout.

This allows us to fit an object to its contents or allocate extra space if required.

virtual void wxRichTextObject::SetMinSize (const wxSize & *sz*) [virtual]

Sets the minimum object size as calculated by Layout.

This allows us to constrain an object to its absolute minimum size if necessary.

void wxRichTextObject::SetName (const wxString & *name*)

Sets the identifying name for this object as a property using the "name" key.

void wxRichTextObject::SetOwnRange (const wxRichTextRange & *range*)

Set the object's own range, for a top-level object with its own position space.

virtual void wxRichTextObject::SetParent (wxRichTextObject * *parent*) [virtual]

Sets the pointer to the parent object.

virtual void wxRichTextObject::SetPosition (const wxPoint & *pos*) [virtual]

Sets the object position in pixels.

void wxRichTextObject::SetProperties (const wxRichTextProperties & *props*)

Sets the object's properties.

void wxRichTextObject::SetRange (const wxRichTextRange & *range*)

Sets the object's range within its container.

virtual void wxRichTextObject::Show (bool *show*) `[virtual]`

Call to show or hide this object.

This function does not cause the content to be laid out or redrawn.

virtual wxRichTextObject* wxRichTextObject::Split (wxRichTextDrawingContext & *context*) `[virtual]`

Returns the final object in the split objects if this object was split due to differences between sub-object virtual attributes.

Returns itself if it was not split.

Reimplemented in wxRichTextPlainText.

virtual bool wxRichTextObject::UsesParagraphAttributes ( ) const `[virtual]`

Returns true if this object takes note of paragraph attributes (text and image objects don't).

Reimplemented in wxRichTextImage, and wxRichTextPlainText.

21.634.4 Member Data Documentation

wxRichTextAttr wxRichTextObject::m_attributes `[protected]`

int wxRichTextObject::m_descent `[protected]`

wxSize wxRichTextObject::m_maxSize `[protected]`

wxSize wxRichTextObject::m_minSize `[protected]`

wxRichTextRange wxRichTextObject::m_ownRange `[protected]`

wxRichTextObject* wxRichTextObject::m_parent `[protected]`

wxPoint wxRichTextObject::m_pos `[protected]`

wxRichTextProperties wxRichTextObject::m_properties `[protected]`

wxRichTextRange wxRichTextObject::m_range `[protected]`

int wxRichTextObject::m_refCount `[protected]`

bool wxRichTextObject::m_show `[protected]`

wxSize wxRichTextObject::m_size `[protected]`

21.635 wxRichTextObjectAddress Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.635.1 Detailed Description

A class for specifying an object anywhere in an object hierarchy, without using a pointer, necessary since wxRTC commands may delete and recreate sub-objects so physical object addresses change.

An array of positions (one per hierarchy level) is used.

Library: wxRichText

Category: Rich Text

See also

wxRichTextCommand

Public Member Functions

- wxRichTextObjectAddress (wxRichTextParagraphLayoutBox *topLevelContainer, wxRichTextObject *obj)
  Creates the address given a container and an object.
- wxRichTextObjectAddress ()
- wxRichTextObjectAddress (const wxRichTextObjectAddress &address)
- void Init ()
- void Copy (const wxRichTextObjectAddress &address)
  Copies the address.
- void operator= (const wxRichTextObjectAddress &address)
  Assignment operator.
- wxRichTextObject * GetObject (wxRichTextParagraphLayoutBox *topLevelContainer) const
  Returns the object specified by the address, given a top level container.
- bool Create (wxRichTextParagraphLayoutBox *topLevelContainer, wxRichTextObject *obj)
  Creates the address given a container and an object.
- wxArrayInt & GetAddress ()
  Returns the array of integers representing the object address.
- const wxArrayInt & GetAddress () const
  Returns the array of integers representing the object address.
- void SetAddress (const wxArrayInt &address)
  Sets the address from an array of integers.

Protected Attributes

- wxArrayInt m_address

21.635.2 Constructor & Destructor Documentation

wxRichTextObjectAddress::wxRichTextObjectAddress (wxRichTextParagraphLayoutBox * *topLevelContainer,* wxRichTextObject * *obj*) `[inline]`

Creates the address given a container and an object.

wxRichTextObjectAddress::wxRichTextObjectAddress ( ) `[inline]`

wxRichTextObjectAddress::wxRichTextObjectAddress (const wxRichTextObjectAddress & *address*) `[inline]`

21.635.3 Member Function Documentation

void wxRichTextObjectAddress::Copy (const wxRichTextObjectAddress & *address*) `[inline]`

Copies the address.

bool wxRichTextObjectAddress::Create (wxRichTextParagraphLayoutBox * *topLevelContainer,* wxRichTextObject * *obj*)

Creates the address given a container and an object.

wxArrayInt& wxRichTextObjectAddress::GetAddress ( ) [inline]

Returns the array of integers representing the object address.

const wxArrayInt& wxRichTextObjectAddress::GetAddress ( ) const [inline]

Returns the array of integers representing the object address.

wxRichTextObject* wxRichTextObjectAddress::GetObject (wxRichTextParagraphLayoutBox * *topLevelContainer*) const

Returns the object specified by the address, given a top level container.

void wxRichTextObjectAddress::Init ( ) [inline]

void wxRichTextObjectAddress::operator= (const wxRichTextObjectAddress & *address*) [inline]

Assignment operator.

void wxRichTextObjectAddress::SetAddress (const wxArrayInt & *address*) [inline]

Sets the address from an array of integers.

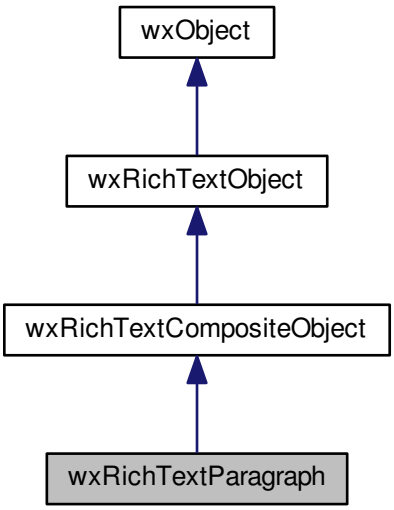
21.635.4 Member Data Documentation

wxArrayInt wxRichTextObjectAddress::m_address [protected]

21.636 wxRichTextParagraph Class Reference

```
#include <wx/richtext/richtextbuffer.h>
```

Inheritance diagram for wxRichTextParagraph:



21.636.1 Detailed Description

This object represents a single paragraph containing various objects such as text content, images, and further paragraph layout objects.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextParagraph (wxRichTextObject *parent=NULL, wxRichTextAttr *style=NULL)

  Constructor taking a parent and style.
- wxRichTextParagraph (const wxString &text, wxRichTextObject *parent=NULL, wxRichTextAttr *paraStyle=NULL, wxRichTextAttr *charStyle=NULL)

  Constructor taking a text string, a parent and paragraph and character attributes.
- virtual ~wxRichTextParagraph ()
- wxRichTextParagraph (const wxRichTextParagraph &obj)
- void Init ()
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

Lay the item out at the specified position with the given size constraint.

- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.
- virtual bool FindPosition (wxDC &dc, wxRichTextDrawingContext &context, long index, wxPoint &pt, int *height, bool forceLineStart)

  Finds the absolute position and row height for the given character position.
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

  Hit-testing: returns a flag indicating hit test details, plus information about position.
- virtual void CalculateRange (long start, long &end)

  Calculates the range of the object.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- wxRichTextLineList & GetLines ()

  Returns the cached lines.
- void Copy (const wxRichTextParagraph &obj)

  Copies the object.
- virtual wxRichTextObject * Clone () const

  Clones the object.
- void ClearLines ()

  Clears the cached lines.
- virtual void ApplyParagraphStyle (wxRichTextLine *line, const wxRichTextAttr &attr, const wxRect &rect, wxDC &dc)

  Applies paragraph styles such as centering to the wrapped lines.
- virtual bool InsertText (long pos, const wxString &text)

  Inserts text at the given position.
- virtual wxRichTextObject * SplitAt (long pos, wxRichTextObject **previousObject=NULL)

  Splits an object at this position if necessary, and returns the previous object, or NULL if inserting at the beginning.
- virtual void MoveToList (wxRichTextObject *obj, wxList &list)

  Moves content to a list from this point.
- virtual void MoveFromList (wxList &list)

  Adds content back from a list.
- bool GetContiguousPlainText (wxString &text, const wxRichTextRange &range, bool fromStart=true)

  Returns the plain text searching from the start or end of the range.
- bool FindWrapPosition (const wxRichTextRange &range, wxDC &dc, wxRichTextDrawingContext &context, int availableSpace, long &wrapPosition, wxArrayInt *partialExtents)

  Finds a suitable wrap position.
- wxRichTextObject * FindObjectAtPosition (long position)

  Finds the object at the given position.
- wxString GetBulletText ()

  Returns the bullet text for this paragraph.
- wxRichTextLine * AllocateLine (int pos)

  Allocates or reuses a line object.
- bool ClearUnusedLines (int lineCount)

  Clears remaining unused line objects, if any.
- wxRichTextAttr GetCombinedAttributes (const wxRichTextAttr &contentStyle, bool includingBoxAttr=false) const

  Returns combined attributes of the base style, paragraph style and character style.
- wxRichTextAttr GetCombinedAttributes (bool includingBoxAttr=false) const

  Returns the combined attributes of the base style and paragraph style.

- long GetFirstLineBreakPosition (long pos)

  Returns the first position from pos that has a line break character.

- void LayoutFloat (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style, wxRichTextFloatCollector *floatCollector)

  Lays out the floating objects.

- int GetImpactedByFloatingObjects () const

  Whether the paragraph is impacted by floating objects from above.

- void SetImpactedByFloatingObjects (int i)

  Sets whether the paragraph is impacted by floating objects from above.

Static Public Member Functions

- static void InitDefaultTabs ()

  Creates a default tabstop array.

- static void ClearDefaultTabs ()

  Clears the default tabstop array.

- static const wxArrayInt & GetDefaultTabs ()

  Returns the default tabstop array.

Protected Attributes

- wxRichTextLineList m_cachedLines
- int m_impactedByFloatingObjects

Static Protected Attributes

- static wxArrayInt sm_defaultTabs

Friends

- class wxRichTextFloatCollector

Additional Inherited Members

21.636.2 Constructor & Destructor Documentation

wxRichTextParagraph::wxRichTextParagraph (wxRichTextObject * *parent* = NULL, wxRichTextAttr * *style* = NULL)

Constructor taking a parent and style.

wxRichTextParagraph::wxRichTextParagraph (const wxString & *text*, wxRichTextObject * *parent* = NULL, wxRichTextAttr * *paraStyle* = NULL, wxRichTextAttr * *charStyle* = NULL)

Constructor taking a text string, a parent and paragraph and character attributes.

virtual wxRichTextParagraph::~wxRichTextParagraph ( ) [virtual]

wxRichTextParagraph::wxRichTextParagraph (const wxRichTextParagraph & *obj*) [inline]

21.636.3 Member Function Documentation

wxRichTextLine* wxRichTextParagraph::AllocateLine (int *pos*)

Allocates or reuses a line object.

virtual void wxRichTextParagraph::ApplyParagraphStyle (wxRichTextLine * *line,* const wxRichTextAttr & *attr,* const wxRect & *rect,* wxDC & *dc*) [virtual]

Applies paragraph styles such as centering to the wrapped lines.

virtual void wxRichTextParagraph::CalculateRange (long *start,* long & *end*) [virtual]

Calculates the range of the object.

By default, guess that the object is 1 unit long.

Reimplemented from wxRichTextCompositeObject.

static void wxRichTextParagraph::ClearDefaultTabs ( ) [static]

Clears the default tabstop array.

void wxRichTextParagraph::ClearLines ( )

Clears the cached lines.

bool wxRichTextParagraph::ClearUnusedLines (int *lineCount*)

Clears remaining unused line objects, if any.

virtual wxRichTextObject* wxRichTextParagraph::Clone ( ) const [inline], [virtual]

Clones the object.

Reimplemented from wxRichTextObject.

void wxRichTextParagraph::Copy (const wxRichTextParagraph & *obj*)

Copies the object.

virtual bool wxRichTextParagraph::Draw (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) [virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implements wxRichTextObject.

wxRichTextObject* **wxRichTextParagraph::FindObjectAtPosition** (**long** *position*)

Finds the object at the given position.

virtual bool wxRichTextParagraph::FindPosition (**wxDC** & *dc,* **wxRichTextDrawingContext** & *context,* **long** *index,* **wxPoint** & *pt,* **int** * *height,* **bool** *forceLineStart*) `[virtual]`

Finds the absolute position and row height for the given character position.

Reimplemented from wxRichTextCompositeObject.

bool wxRichTextParagraph::FindWrapPosition (**const wxRichTextRange** & *range,* **wxDC** & *dc,* **wxRichTextDrawingContext** & *context,* **int** *availableSpace,* **long** & *wrapPosition,* **wxArrayInt** * *partialExtents*)

Finds a suitable wrap position.

wrapPosition is the last position in the line to the left of the split.

wxString wxRichTextParagraph::GetBulletText ( )

Returns the bullet text for this paragraph.

wxRichTextAttr wxRichTextParagraph::GetCombinedAttributes (**const wxRichTextAttr** & *contentStyle,* **bool** *includingBoxAttr* = `false`) **const**

Returns combined attributes of the base style, paragraph style and character style.

We use this to dynamically retrieve the actual style.

wxRichTextAttr wxRichTextParagraph::GetCombinedAttributes (**bool** *includingBoxAttr* = `false`) **const**

Returns the combined attributes of the base style and paragraph style.

bool wxRichTextParagraph::GetContiguousPlainText (**wxString** & *text,* **const wxRichTextRange** & *range,* **bool** *fromStart* = `true`)

Returns the plain text searching from the start or end of the range.

The resulting string may be shorter than the range given.

static const wxArrayInt& wxRichTextParagraph::GetDefaultTabs ( ) `[inline],[static]`

Returns the default tabstop array.

long wxRichTextParagraph::GetFirstLineBreakPosition (**long** *pos*)

Returns the first position from pos that has a line break character.

int wxRichTextParagraph::GetImpactedByFloatingObjects ( ) **const** `[inline]`

Whether the paragraph is impacted by floating objects from above.

wxRichTextLineList& wxRichTextParagraph::GetLines ( ) [inline]

Returns the cached lines.

virtual bool wxRichTextParagraph::GetRangeSize (const wxRichTextRange & *range,* wxSize & *size,* int & *descent,* wxDC & *dc,* wxRichTextDrawingContext & *context,* int *flags,* const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Reimplemented from wxRichTextCompositeObject.

virtual wxString wxRichTextParagraph::GetXMLNodeName ( ) const [inline], [virtual]

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextObject.

virtual int wxRichTextParagraph::HitTest (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxPoint & *pt,* long & *textPosition,* wxRichTextObject ** *obj,* wxRichTextObject ** *contextObj,* int *flags* = 0) [virtual]

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented from wxRichTextCompositeObject.

void wxRichTextParagraph::Init ( )

static void wxRichTextParagraph::InitDefaultTabs ( ) [static]

Creates a default tabstop array.

virtual bool wxRichTextParagraph::InsertText (long *pos,* const wxString & *text*) [virtual]

Inserts text at the given position.

virtual bool wxRichTextParagraph::Layout (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style*) [virtual]

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implements wxRichTextObject.

void wxRichTextParagraph::LayoutFloat (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style,* wxRichTextFloatCollector * *floatCollector*)

Lays out the floating objects.

virtual void wxRichTextParagraph::MoveFromList (wxList & *list*) `[virtual]`

Adds content back from a list.

virtual void wxRichTextParagraph::MoveToList (wxRichTextObject * *obj,* wxList & *list*) `[virtual]`

Moves content to a list from this point.

void wxRichTextParagraph::SetImpactedByFloatingObjects (int *i*) `[inline]`

Sets whether the paragraph is impacted by floating objects from above.

virtual wxRichTextObject* wxRichTextParagraph::SplitAt (long *pos,* wxRichTextObject ** *previousObject* = NULL) `[virtual]`

Splits an object at this position if necessary, and returns the previous object, or NULL if inserting at the beginning.

21.636.4 Friends And Related Function Documentation

friend class wxRichTextFloatCollector `[friend]`

21.636.5 Member Data Documentation

wxRichTextLineList wxRichTextParagraph::m_cachedLines `[protected]`

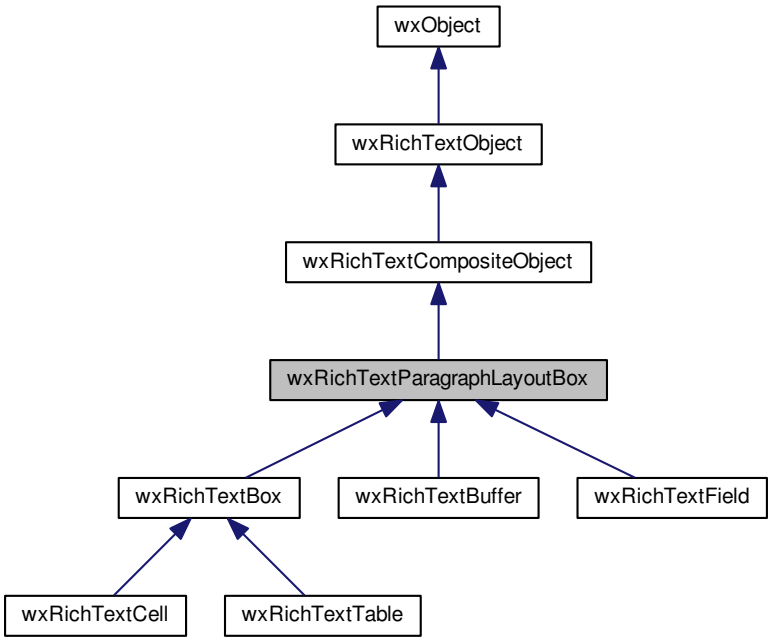
int wxRichTextParagraph::m_impactedByFloatingObjects `[protected]`

wxArrayInt wxRichTextParagraph::sm_defaultTabs `[static]`, `[protected]`

21.637 wxRichTextParagraphLayoutBox Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextParagraphLayoutBox:



21.637.1 Detailed Description

This class knows how to lay out paragraphs.

Library: wxRichText

Category: Rich Text

See also

wxRichTextCompositeObject, wxRichTextObject, wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextParagraphLayoutBox (wxRichTextObject *parent=NULL)
- wxRichTextParagraphLayoutBox (const wxRichTextParagraphLayoutBox &obj)
- ~wxRichTextParagraphLayoutBox ()
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

  Hit-testing: returns a flag indicating hit test details, plus information about position.
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

Lay the item out at the specified position with the given size constraint.

- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.

- virtual bool DeleteRange (const wxRichTextRange &range)

  Deletes the given range.

- virtual wxString GetTextForRange (const wxRichTextRange &range) const

  Returns any text in this object for the given range.

- virtual bool ImportFromXML (wxRichTextBuffer *buffer, wxXmlNode *node, wxRichTextXMLHandler *handler, bool *recurse)

  Imports this object from XML.

- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.

- virtual bool AcceptsFocus () const

  Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.

- void SetRichTextCtrl (wxRichTextCtrl *ctrl)

  Associates a control with the buffer, for operations that for example require refreshing the window.

- wxRichTextCtrl * GetRichTextCtrl () const

  Returns the associated control.

- void SetPartialParagraph (bool partialPara)

  Sets a flag indicating whether the last paragraph is partial or complete.

- bool GetPartialParagraph () const

  Returns a flag indicating whether the last paragraph is partial or complete.

- virtual wxRichTextStyleSheet * GetStyleSheet () const

  Returns the style sheet associated with the overall buffer.

- virtual bool IsTopLevel () const

  Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.

- bool InsertParagraphsWithUndo (wxRichTextBuffer *buffer, long pos, const wxRichTextParagraphLayoutBox &paragraphs, wxRichTextCtrl *ctrl, int flags=0)

  Submits a command to insert paragraphs.

- bool InsertTextWithUndo (wxRichTextBuffer *buffer, long pos, const wxString &text, wxRichTextCtrl *ctrl, int flags=0)

  Submits a command to insert the given text.

- bool InsertNewlineWithUndo (wxRichTextBuffer *buffer, long pos, wxRichTextCtrl *ctrl, int flags=0)

  Submits a command to insert the given text.

- bool InsertImageWithUndo (wxRichTextBuffer *buffer, long pos, const wxRichTextImageBlock &imageBlock, wxRichTextCtrl *ctrl, int flags, const wxRichTextAttr &textAttr)

  Submits a command to insert the given image.

- wxRichTextField * InsertFieldWithUndo (wxRichTextBuffer *buffer, long pos, const wxString &fieldType, const wxRichTextProperties &properties, wxRichTextCtrl *ctrl, int flags, const wxRichTextAttr &textAttr)

  Submits a command to insert the given field.

- wxRichTextAttr GetStyleForNewParagraph (wxRichTextBuffer *buffer, long pos, bool caretPosition=false, bool lookUpNewParaStyle=false) const

  Returns the style that is appropriate for a new paragraph at this position.

- wxRichTextObject * InsertObjectWithUndo (wxRichTextBuffer *buffer, long pos, wxRichTextObject *object, wxRichTextCtrl *ctrl, int flags=0)

  Inserts an object.

- bool DeleteRangeWithUndo (const wxRichTextRange &range, wxRichTextCtrl *ctrl, wxRichTextBuffer *buffer)

  Submits a command to delete this range.

- void DrawFloats (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draws the floating objects in this buffer.
- void MoveAnchoredObjectToParagraph (wxRichTextParagraph *from, wxRichTextParagraph *to, wxRichTextObject *obj)

  Moves an anchored object to another paragraph.
- void Init ()

  Initializes the object.
- virtual void Clear ()

  Clears all the children.
- virtual void Reset ()

  Clears and initializes with one blank paragraph.
- virtual wxRichTextRange AddParagraph (const wxString &text, wxRichTextAttr *paraStyle=NULL)

  Convenience function to add a paragraph of text.
- virtual wxRichTextRange AddImage (const wxImage &image, wxRichTextAttr *paraStyle=NULL)

  Convenience function to add an image.
- virtual wxRichTextRange AddParagraphs (const wxString &text, wxRichTextAttr *paraStyle=NULL)

  Adds multiple paragraphs, based on newlines.
- virtual wxRichTextLine * GetLineAtPosition (long pos, bool caretPosition=false) const

  Returns the line at the given position.
- virtual wxRichTextLine * GetLineAtYPosition (int y) const

  Returns the line at the given y pixel position, or the last line.
- virtual wxRichTextParagraph * GetParagraphAtPosition (long pos, bool caretPosition=false) const

  Returns the paragraph at the given character or caret position.
- virtual wxSize GetLineSizeAtPosition (long pos, bool caretPosition=false) const

  Returns the line size at the given position.
- virtual long GetVisibleLineNumber (long pos, bool caretPosition=false, bool startOfLine=false) const

  Given a position, returns the number of the visible line (potentially many to a paragraph), starting from zero at the start of the buffer.
- virtual wxRichTextLine * GetLineForVisibleLineNumber (long lineNumber) const

  Given a line number, returns the corresponding wxRichTextLine object.
- virtual wxRichTextObject * GetLeafObjectAtPosition (long position) const

  Returns the leaf object in a paragraph at this position.
- virtual wxRichTextParagraph * GetParagraphAtLine (long paragraphNumber) const

  Returns the paragraph by number.
- virtual wxRichTextParagraph * GetParagraphForLine (wxRichTextLine *line) const

  Returns the paragraph for a given line.
- virtual int GetParagraphLength (long paragraphNumber) const

  Returns the length of the paragraph.
- virtual int GetParagraphCount () const

  Returns the number of paragraphs.
- virtual int GetLineCount () const

  Returns the number of visible lines.
- virtual wxString GetParagraphText (long paragraphNumber) const

  Returns the text of the paragraph.
- virtual long XYToPosition (long x, long y) const

  Converts zero-based line column and paragraph number to a position.
- virtual bool PositionToXY (long pos, long *x, long *y) const

  Converts a zero-based position to line column and paragraph number.
- virtual bool SetStyle (const wxRichTextRange &range, const wxRichTextAttr &style, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO)

Sets the attributes for the given range.

- virtual void SetStyle (wxRichTextObject *obj, const wxRichTextAttr &textAttr, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO)

  Sets the attributes for the given object only, for example the box attributes for a text box.
- virtual bool GetStyle (long position, wxRichTextAttr &style)

  Returns the combined text attributes for this position.
- virtual bool GetUncombinedStyle (long position, wxRichTextAttr &style)

  Returns the content (uncombined) attributes for this position.
- virtual bool DoGetStyle (long position, wxRichTextAttr &style, bool combineStyles=true)

  Implementation helper for GetStyle.
- virtual bool GetStyleForRange (const wxRichTextRange &range, wxRichTextAttr &style)

  This function gets a style representing the common, combined attributes in the given range.
- bool CollectStyle (wxRichTextAttr &currentStyle, const wxRichTextAttr &style, wxRichTextAttr &clashingAttr, wxRichTextAttr &absentAttr)

  Combines style with currentStyle for the purpose of summarising the attributes of a range of content.
- virtual bool ClearListStyle (const wxRichTextRange &range, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO)

  Clears the list style from the given range, clearing list-related attributes and applying any named paragraph style associated with each paragraph.
- virtual bool DoNumberList (const wxRichTextRange &range, const wxRichTextRange &promotionRange, int promoteBy, wxRichTextListStyleDefinition *def, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Helper for NumberList and PromoteList, that does renumbering and promotion simultaneously def can be NULL/empty to indicate that the existing list style should be used.
- virtual bool FindNextParagraphNumber (wxRichTextParagraph *previousParagraph, wxRichTextAttr &attr) const

  Fills in the attributes for numbering a paragraph after previousParagraph.
- virtual bool SetProperties (const wxRichTextRange &range, const wxRichTextProperties &properties, int flags=wxRICHTEXT_SETPROPERTIES_WITH_UNDO)

  Sets the properties for the given range, passing flags to determine how the attributes are set.
- virtual bool SetObjectPropertiesWithUndo (wxRichTextObject &obj, const wxRichTextProperties &properties, wxRichTextObject *objToSet=NULL)

  Sets with undo the properties for the given object.
- virtual bool HasCharacterAttributes (const wxRichTextRange &range, const wxRichTextAttr &style) const

  Test if this whole range has character attributes of the specified kind.
- virtual bool HasParagraphAttributes (const wxRichTextRange &range, const wxRichTextAttr &style) const

  Test if this whole range has paragraph attributes of the specified kind.
- virtual wxRichTextObject * Clone () const

  Clones the object.
- virtual void PrepareContent (wxRichTextParagraphLayoutBox &container)

  Prepares the content just before insertion (or after buffer reset).
- virtual bool InsertFragment (long position, wxRichTextParagraphLayoutBox &fragment)

  Insert fragment into this box at the given position.
- virtual bool CopyFragment (const wxRichTextRange &range, wxRichTextParagraphLayoutBox &fragment)

  Make a copy of the fragment corresponding to the given range, putting it in fragment.
- virtual bool ApplyStyleSheet (wxRichTextStyleSheet *styleSheet)

  Apply the style sheet to the buffer, for example if the styles have changed.
- void Copy (const wxRichTextParagraphLayoutBox &obj)
- void operator= (const wxRichTextParagraphLayoutBox &obj)
- virtual void UpdateRanges ()

  Calculate ranges.
- virtual wxString GetText () const

Get all the text.

- virtual bool SetDefaultStyle (const wxRichTextAttr &style)

  Sets the default style, affecting the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).
- virtual const wxRichTextAttr & GetDefaultStyle () const

  Returns the current default style, affecting the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).
- virtual void SetBasicStyle (const wxRichTextAttr &style)

  Sets the basic (overall) style.
- virtual const wxRichTextAttr & GetBasicStyle () const

  Returns the basic (overall) style.
- virtual void Invalidate (const wxRichTextRange &invalidRange=wxRICHTEXT_ALL)

  Invalidates the buffer.
- virtual void DoInvalidate (const wxRichTextRange &invalidRange)

  Do the (in)validation for this object only.
- virtual void InvalidateHierarchy (const wxRichTextRange &invalidRange=wxRICHTEXT_ALL)

  Do the (in)validation both up and down the hierarchy.
- virtual bool UpdateFloatingObjects (const wxRect &availableRect, wxRichTextObject *untilObj=NULL)

  Gather information about floating objects.
- wxRichTextRange GetInvalidRange (bool wholeParagraphs=false) const

  Get invalid range, rounding to entire paragraphs if argument is true.
- bool IsDirty () const

  Returns true if this object needs layout.
- wxRichTextFloatCollector * GetFloatCollector ()

  Returns the wxRichTextFloatCollector of this object.
- int GetFloatingObjectCount () const

  Returns the number of floating objects at this level.
- bool GetFloatingObjects (wxRichTextObjectList &objects) const

  Returns a list of floating objects.

- virtual bool SetListStyle (const wxRichTextRange &range, wxRichTextListStyleDefinition *def, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Sets the list attributes for the given range, passing flags to determine how the attributes are set.
- virtual bool SetListStyle (const wxRichTextRange &range, const wxString &defName, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Sets the list attributes for the given range, passing flags to determine how the attributes are set.

- virtual bool NumberList (const wxRichTextRange &range, wxRichTextListStyleDefinition *def=NULL, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Numbers the paragraphs in the given range.
- virtual bool NumberList (const wxRichTextRange &range, const wxString &defName, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int startFrom=1, int specifiedLevel=-1)

  Numbers the paragraphs in the given range.

- virtual bool PromoteList (int promoteBy, const wxRichTextRange &range, wxRichTextListStyleDefinition *def=NULL, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int specifiedLevel=-1)

  Promotes the list items within the given range.
- virtual bool PromoteList (int promoteBy, const wxRichTextRange &range, const wxString &defName, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO, int specifiedLevel=-1)

  Promotes the list items within the given range.

Protected Attributes

- wxRichTextCtrl * m_ctrl
- wxRichTextAttr m_defaultAttributes
- wxRichTextRange m_invalidRange
- bool m_partialParagraph
- wxRichTextFloatCollector * m_floatCollector

Additional Inherited Members

21.637.2 Constructor & Destructor Documentation

wxRichTextParagraphLayoutBox::wxRichTextParagraphLayoutBox (wxRichTextObject * *parent* = NULL)

wxRichTextParagraphLayoutBox::wxRichTextParagraphLayoutBox (const wxRichTextParagraphLayoutBox & *obj*) `[inline]`

wxRichTextParagraphLayoutBox::~wxRichTextParagraphLayoutBox ( )

21.637.3 Member Function Documentation

virtual bool wxRichTextParagraphLayoutBox::AcceptsFocus ( ) const `[inline], [virtual]`

Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.

For example, containers supporting text, such as a text box object, can accept the focus, but a table can't (set the focus to individual cells instead).

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextField.

virtual wxRichTextRange wxRichTextParagraphLayoutBox::AddImage (const wxImage & *image*, wxRichTextAttr * *paraStyle* = NULL) `[virtual]`

Convenience function to add an image.

virtual wxRichTextRange wxRichTextParagraphLayoutBox::AddParagraph (const wxString & *text*, wxRichTextAttr * *paraStyle* = NULL) `[virtual]`

Convenience function to add a paragraph of text.

Reimplemented in wxRichTextBuffer.

virtual wxRichTextRange wxRichTextParagraphLayoutBox::AddParagraphs (const wxString & *text*, wxRichTextAttr * *paraStyle* = NULL) `[virtual]`

Adds multiple paragraphs, based on newlines.

virtual bool wxRichTextParagraphLayoutBox::ApplyStyleSheet (wxRichTextStyleSheet * *styleSheet*) `[virtual]`

Apply the style sheet to the buffer, for example if the styles have changed.

virtual void wxRichTextParagraphLayoutBox::Clear ( ) [virtual]

Clears all the children.

virtual bool wxRichTextParagraphLayoutBox::ClearListStyle (const wxRichTextRange & *range,* int *flags =* wxRICHTEXT_SETSTYLE_WITH_UNDO) [virtual]

Clears the list style from the given range, clearing list-related attributes and applying any named paragraph style associated with each paragraph.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.

See also

SetListStyle(), PromoteList(), NumberList()

virtual wxRichTextObject* wxRichTextParagraphLayoutBox::Clone ( ) const [inline], [virtual]

Clones the object.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextBuffer, wxRichTextField, and wxRichTextBox.

bool wxRichTextParagraphLayoutBox::CollectStyle (wxRichTextAttr & *currentStyle,* const wxRichTextAttr & *style,* wxRichTextAttr & *clashingAttr,* wxRichTextAttr & *absentAttr*)

Combines *style* with *currentStyle* for the purpose of summarising the attributes of a range of content.

void wxRichTextParagraphLayoutBox::Copy (const wxRichTextParagraphLayoutBox & *obj*)

virtual bool wxRichTextParagraphLayoutBox::CopyFragment (const wxRichTextRange & *range,* wxRichTextParagraphLayoutBox & *fragment*) [virtual]

Make a copy of the fragment corresponding to the given range, putting it in *fragment*.

virtual bool wxRichTextParagraphLayoutBox::DeleteRange (const wxRichTextRange & *range*) [virtual]

Deletes the given range.

Reimplemented from wxRichTextCompositeObject.

Reimplemented in wxRichTextTable.

bool wxRichTextParagraphLayoutBox::DeleteRangeWithUndo (const wxRichTextRange & *range,* wxRichTextCtrl * *ctrl,* wxRichTextBuffer * *buffer*)

Submits a command to delete this range.

virtual bool wxRichTextParagraphLayoutBox::DoGetStyle (long *position*, wxRichTextAttr & *style*, bool *combineStyles* = true) [virtual]

Implementation helper for GetStyle.

If combineStyles is true, combine base, paragraph and context attributes.

virtual void wxRichTextParagraphLayoutBox::DoInvalidate (const wxRichTextRange & *invalidRange*) [virtual]

Do the (in)validation for this object only.

virtual bool wxRichTextParagraphLayoutBox::DoNumberList (const wxRichTextRange & *range*, const wxRichTextRange & *promotionRange*, int *promoteBy*, wxRichTextListStyleDefinition * *def*, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Helper for NumberList and PromoteList, that does renumbering and promotion simultaneously *def* can be NUL↩ L/empty to indicate that the existing list style should be used.

virtual bool wxRichTextParagraphLayoutBox::Draw (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRichTextRange & *range*, const wxRichTextSelection & *selection*, const wxRect & *rect*, int *descent*, int *style*) [virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implements wxRichTextObject.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextField, and wxRichTextBox.

void wxRichTextParagraphLayoutBox::DrawFloats (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRichTextRange & *range*, const wxRichTextSelection & *selection*, const wxRect & *rect*, int *descent*, int *style*)

Draws the floating objects in this buffer.

virtual bool wxRichTextParagraphLayoutBox::FindNextParagraphNumber (wxRichTextParagraph * *previousParagraph*, wxRichTextAttr & *attr*) const [virtual]

Fills in the attributes for numbering a paragraph after previousParagraph.

virtual const wxRichTextAttr& wxRichTextParagraphLayoutBox::GetBasicStyle ( ) const [inline], [virtual]

Returns the basic (overall) style.

This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

virtual const wxRichTextAttr& wxRichTextParagraphLayoutBox::GetDefaultStyle ( ) const [inline], [virtual]

Returns the current default style, affecting the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

wxRichTextFloatCollector* wxRichTextParagraphLayoutBox::GetFloatCollector ( ) [inline]

Returns the wxRichTextFloatCollector of this object.

int wxRichTextParagraphLayoutBox::GetFloatingObjectCount ( ) const

Returns the number of floating objects at this level.

bool wxRichTextParagraphLayoutBox::GetFloatingObjects (wxRichTextObjectList & *objects*) const

Returns a list of floating objects.

wxRichTextRange wxRichTextParagraphLayoutBox::GetInvalidRange (bool *wholeParagraphs* = false) const

Get invalid range, rounding to entire paragraphs if argument is true.

virtual wxRichTextObject* wxRichTextParagraphLayoutBox::GetLeafObjectAtPosition (long *position*) const [virtual]

Returns the leaf object in a paragraph at this position.

virtual wxRichTextLine* wxRichTextParagraphLayoutBox::GetLineAtPosition (long *pos,* bool *caretPosition* = false) const [virtual]

Returns the line at the given position.

If *caretPosition* is true, the position is a caret position, which is normally a smaller number.

virtual wxRichTextLine* wxRichTextParagraphLayoutBox::GetLineAtYPosition (int *y*) const [virtual]

Returns the line at the given y pixel position, or the last line.

virtual int wxRichTextParagraphLayoutBox::GetLineCount ( ) const [virtual]

Returns the number of visible lines.

virtual wxRichTextLine* wxRichTextParagraphLayoutBox::GetLineForVisibleLineNumber (long *lineNumber*) const [virtual]

Given a line number, returns the corresponding wxRichTextLine object.

virtual wxSize wxRichTextParagraphLayoutBox::GetLineSizeAtPosition (long *pos,* bool *caretPosition* = false) const [virtual]

Returns the line size at the given position.

virtual wxRichTextParagraph* wxRichTextParagraphLayoutBox::GetParagraphAtLine (long *paragraphNumber*) const [virtual]

Returns the paragraph by number.

virtual wxRichTextParagraph* wxRichTextParagraphLayoutBox::GetParagraphAtPosition (long *pos,* bool *caretPosition* = false) const [virtual]

Returns the paragraph at the given character or caret position.

virtual int wxRichTextParagraphLayoutBox::GetParagraphCount ( ) const [inline], [virtual]

Returns the number of paragraphs.

virtual wxRichTextParagraph* wxRichTextParagraphLayoutBox::GetParagraphForLine (wxRichTextLine * *line*) const [virtual]

Returns the paragraph for a given line.

virtual int wxRichTextParagraphLayoutBox::GetParagraphLength (long *paragraphNumber*) const [virtual]

Returns the length of the paragraph.

virtual wxString wxRichTextParagraphLayoutBox::GetParagraphText (long *paragraphNumber*) const [virtual]

Returns the text of the paragraph.

bool wxRichTextParagraphLayoutBox::GetPartialParagraph ( ) const [inline]

Returns a flag indicating whether the last paragraph is partial or complete.

virtual bool wxRichTextParagraphLayoutBox::GetRangeSize (const wxRichTextRange & *range,* wxSize & *size,* int & *descent,* wxDC & *dc,* wxRichTextDrawingContext & *context,* int *flags,* const wxPoint & *position* = wxPoint (0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Reimplemented from wxRichTextCompositeObject.

Reimplemented in wxRichTextTable, and wxRichTextField.

wxRichTextCtrl* wxRichTextParagraphLayoutBox::GetRichTextCtrl ( ) const [inline]

Returns the associated control.

virtual bool wxRichTextParagraphLayoutBox::GetStyle (long *position,* wxRichTextAttr & *style*) [virtual]

Returns the combined text attributes for this position.

This function gets the *uncombined* style - that is, the attributes associated with the paragraph or character content, and not necessarily the combined attributes you see on the screen. To get the combined attributes, use GetStyle(). If you specify (any) paragraph attribute in *style's* flags, this function will fetch the paragraph attributes. Otherwise, it will return the character attributes.

wxRichTextAttr wxRichTextParagraphLayoutBox::GetStyleForNewParagraph (wxRichTextBuffer * *buffer,* long *pos,* bool *caretPosition* = false, bool *lookUpNewParaStyle* = false) const

Returns the style that is appropriate for a new paragraph at this position.

If the previous paragraph has a paragraph style name, looks up the next-paragraph style.

virtual bool wxRichTextParagraphLayoutBox::GetStyleForRange (const wxRichTextRange & *range,* wxRichTextAttr & *style*) [virtual]

This function gets a style representing the common, combined attributes in the given range.

Attributes which have different values within the specified range will not be included the style flags.

The function is used to get the attributes to display in the formatting dialog: the user can edit the attributes common to the selection, and optionally specify the values of further attributes to be applied uniformly.

To apply the edited attributes, you can use SetStyle() specifying the wxRICHTEXT_SETSTYLE_OPTIMIZE flag, which will only apply attributes that are different from the *combined* attributes within the range. So, the user edits the effective, displayed attributes for the range, but his choice won't be applied unnecessarily to content. As an example, say the style for a paragraph specifies bold, but the paragraph text doesn't specify a weight. The combined style is bold, and this is what the user will see on-screen and in the formatting dialog. The user now specifies red text, in addition to bold. When applying with SetStyle(), the content font weight attributes won't be changed to bold because this is already specified by the paragraph. However the text colour attributes *will* be changed to show red.

virtual wxRichTextStyleSheet* wxRichTextParagraphLayoutBox::GetStyleSheet ( ) const [virtual]

Returns the style sheet associated with the overall buffer.

Reimplemented in wxRichTextBuffer.

virtual wxString wxRichTextParagraphLayoutBox::GetText ( ) const [virtual]

Get all the text.

virtual wxString wxRichTextParagraphLayoutBox::GetTextForRange (const wxRichTextRange & *range*) const [virtual]

Returns any text in this object for the given range.

Reimplemented from wxRichTextCompositeObject.

Reimplemented in wxRichTextTable.

virtual bool wxRichTextParagraphLayoutBox::GetUncombinedStyle (long *position,* wxRichTextAttr & *style*) [virtual]

Returns the content (uncombined) attributes for this position.

virtual long wxRichTextParagraphLayoutBox::GetVisibleLineNumber (long *pos,* bool *caretPosition* = false, bool *startOfLine* = false) const [virtual]

Given a position, returns the number of the visible line (potentially many to a paragraph), starting from zero at the start of the buffer.

We also have to pass a bool (*startOfLine*) that indicates whether the caret is being shown at the end of the previous line or at the start of the next, since the caret can be shown at two visible positions for the same underlying position.

virtual wxString wxRichTextParagraphLayoutBox::GetXMLNodeName ( ) const `[inline], [virtual]`

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable, wxRichTextCell, wxRichTextField, and wxRichTextBox.

virtual bool wxRichTextParagraphLayoutBox::HasCharacterAttributes (const wxRichTextRange & *range,* const wxRichTextAttr & *style*) const `[virtual]`

Test if this whole range has character attributes of the specified kind.

If any of the attributes are different within the range, the test fails. You can use this to implement, for example, bold button updating. style must have flags indicating which attributes are of interest.

virtual bool wxRichTextParagraphLayoutBox::HasParagraphAttributes (const wxRichTextRange & *range,* const wxRichTextAttr & *style*) const `[virtual]`

Test if this whole range has paragraph attributes of the specified kind.

If any of the attributes are different within the range, the test fails. You can use this to implement, for example, centering button updating. style must have flags indicating which attributes are of interest.

virtual int wxRichTextParagraphLayoutBox::HitTest (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxPoint & *pt,* long & *textPosition,* wxRichTextObject ** *obj,* wxRichTextObject ** *contextObj,* int *flags* = 0) `[virtual]`

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented from wxRichTextCompositeObject.

Reimplemented in wxRichTextTable, wxRichTextCell, and wxRichTextBuffer.

virtual bool wxRichTextParagraphLayoutBox::ImportFromXML (wxRichTextBuffer * *buffer,* wxXmlNode * *node,* wxRichTextXMLHandler * *handler,* bool * *recurse*) `[virtual]`

Imports this object from XML.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextTable.

void wxRichTextParagraphLayoutBox::Init ( )

Initializes the object.

wxRichTextField* wxRichTextParagraphLayoutBox::InsertFieldWithUndo (**wxRichTextBuffer** * *buffer,* long *pos,* const **wxString** & *fieldType,* const **wxRichTextProperties** & *properties,* **wxRichTextCtrl** * *ctrl,* int *flags,* const **wxRichTextAttr** & *textAttr*)

Submits a command to insert the given field.

Field data can be included in properties.

See also

wxRichTextField, wxRichTextFieldType, wxRichTextFieldTypeStandard

virtual bool wxRichTextParagraphLayoutBox::InsertFragment (long *position,* **wxRichTextParagraphLayoutBox** & *fragment*) [virtual]

Insert fragment into this box at the given position.

If partialParagraph is true, it is assumed that the last (or only) paragraph is just a piece of data with no paragraph marker.

bool wxRichTextParagraphLayoutBox::InsertImageWithUndo (**wxRichTextBuffer** * *buffer,* long *pos,* const **wxRichTextImageBlock** & *imageBlock,* **wxRichTextCtrl** * *ctrl,* int *flags,* const **wxRichTextAttr** & *textAttr*)

Submits a command to insert the given image.

bool wxRichTextParagraphLayoutBox::InsertNewlineWithUndo (**wxRichTextBuffer** * *buffer,* long *pos,* **wxRichTextCtrl** * *ctrl,* int *flags* = 0)

Submits a command to insert the given text.

wxRichTextObject* wxRichTextParagraphLayoutBox::InsertObjectWithUndo (**wxRichTextBuffer** * *buffer,* long *pos,* **wxRichTextObject** * *object,* **wxRichTextCtrl** * *ctrl,* int *flags* = 0)

Inserts an object.

bool wxRichTextParagraphLayoutBox::InsertParagraphsWithUndo (**wxRichTextBuffer** * *buffer,* long *pos,* const **wxRichTextParagraphLayoutBox** & *paragraphs,* **wxRichTextCtrl** * *ctrl,* int *flags* = 0)

Submits a command to insert paragraphs.

bool wxRichTextParagraphLayoutBox::InsertTextWithUndo (**wxRichTextBuffer** * *buffer,* long *pos,* const **wxString** & *text,* **wxRichTextCtrl** * *ctrl,* int *flags* = 0)

Submits a command to insert the given text.

virtual void wxRichTextParagraphLayoutBox::Invalidate (const **wxRichTextRange** & *invalidRange* = **wxRICHTEXT_ALL**) [virtual]

Invalidates the buffer.

With no argument, invalidates whole buffer.

Reimplemented from wxRichTextCompositeObject.

virtual void wxRichTextParagraphLayoutBox::InvalidateHierarchy (const wxRichTextRange & *invalidRange* = wxRICHTEXT_ALL) [virtual]

Do the (in)validation both up and down the hierarchy.

bool wxRichTextParagraphLayoutBox::IsDirty ( ) const [inline]

Returns true if this object needs layout.

virtual bool wxRichTextParagraphLayoutBox::IsTopLevel ( ) const [inline], [virtual]

Returns true if this object is top-level, i.e. contains its own paragraphs, such as a text box.

Reimplemented from wxRichTextObject.

Reimplemented in wxRichTextField.

virtual bool wxRichTextParagraphLayoutBox::Layout (wxDC & *dc*, wxRichTextDrawingContext & *context*, const wxRect & *rect*, const wxRect & *parentRect*, int *style*) [virtual]

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implements wxRichTextObject.

Reimplemented in wxRichTextTable, and wxRichTextField.

void wxRichTextParagraphLayoutBox::MoveAnchoredObjectToParagraph (wxRichTextParagraph * *from*, wxRichTextParagraph * *to*, wxRichTextObject * *obj*)

Moves an anchored object to another paragraph.

virtual bool wxRichTextParagraphLayoutBox::NumberList (const wxRichTextRange & *range*, wxRichTextListStyleDefinition * *def* = NULL, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Numbers the paragraphs in the given range.

Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

def can be NULL to indicate that the existing list style should be used.

See also

SetListStyle(), PromoteList(), ClearListStyle()

virtual bool wxRichTextParagraphLayoutBox::NumberList (const wxRichTextRange & *range*, const wxString & *defName*, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Numbers the paragraphs in the given range.

Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

def can be NULL to indicate that the existing list style should be used.

See also

SetListStyle(), PromoteList(), ClearListStyle()

void wxRichTextParagraphLayoutBox::operator= (const wxRichTextParagraphLayoutBox & *obj*) [inline]

virtual bool wxRichTextParagraphLayoutBox::PositionToXY (long *pos*, long * *x*, long * *y*) const [virtual]

Converts a zero-based position to line column and paragraph number.

virtual void wxRichTextParagraphLayoutBox::PrepareContent (wxRichTextParagraphLayoutBox & *container*) [virtual]

Prepares the content just before insertion (or after buffer reset).

Currently is only called if undo mode is on.

virtual bool wxRichTextParagraphLayoutBox::PromoteList (int *promoteBy*, const wxRichTextRange & *range*, wxRichTextListStyleDefinition * *def* = NULL, int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *specifiedLevel* = -1) [virtual]

Promotes the list items within the given range.

A positive *promoteBy* produces a smaller indent, and a negative number produces a larger indent. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

SetListStyle(), SetListStyle(), ClearListStyle()

virtual bool wxRichTextParagraphLayoutBox::PromoteList (int *promoteBy,* const wxRichTextRange & *range,* const wxString & *defName,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *specifiedLevel* = -1) [virtual]

Promotes the list items within the given range.

A positive *promoteBy* produces a smaller indent, and a negative number produces a larger indent. Pass flags to determine how the attributes are set. Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

SetListStyle(), SetListStyle(), ClearListStyle()

virtual void wxRichTextParagraphLayoutBox::Reset ( ) [virtual]

Clears and initializes with one blank paragraph.

virtual void wxRichTextParagraphLayoutBox::SetBasicStyle (const wxRichTextAttr & *style*) [inline], [virtual]

Sets the basic (overall) style.

This is the style of the whole buffer before further styles are applied, unlike the default style, which only affects the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

virtual bool wxRichTextParagraphLayoutBox::SetDefaultStyle (const wxRichTextAttr & *style*) [virtual]

Sets the default style, affecting the style currently being applied (for example, setting the default style to bold will cause subsequently inserted text to be bold).

This is not cumulative - setting the default style will replace the previous default style.

Setting it to a default attribute object makes new content take on the 'basic' style.

virtual bool wxRichTextParagraphLayoutBox::SetListStyle (const wxRichTextRange & *range,* wxRichTextListStyle↩ Definition * *def,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Sets the list attributes for the given range, passing flags to determine how the attributes are set.

Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.

- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

NumberList(), PromoteList(), ClearListStyle().

virtual bool wxRichTextParagraphLayoutBox::SetListStyle (const wxRichTextRange & *range,* const wxString & *defName,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO, int *startFrom* = 1, int *specifiedLevel* = -1) [virtual]

Sets the list attributes for the given range, passing flags to determine how the attributes are set.

Either the style definition or the name of the style definition (in the current sheet) can be passed.

flags is a bit list of the following:

- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this command will be undoable.
- wxRICHTEXT_SETSTYLE_RENUMBER: specifies that numbering should start from *startFrom*, otherwise existing attributes are used.
- wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL: specifies that *listLevel* should be used as the level for all paragraphs, otherwise the current indentation will be used.

See also

NumberList(), PromoteList(), ClearListStyle().

virtual bool wxRichTextParagraphLayoutBox::SetObjectPropertiesWithUndo (wxRichTextObject & *obj,* const wxRichTextProperties & *properties,* wxRichTextObject * *objToSet* = NULL) [virtual]

Sets with undo the properties for the given object.

void wxRichTextParagraphLayoutBox::SetPartialParagraph (bool *partialPara*) [inline]

Sets a flag indicating whether the last paragraph is partial or complete.

virtual bool wxRichTextParagraphLayoutBox::SetProperties (const wxRichTextRange & *range,* const wxRichTextProperties & *properties,* int *flags* = wxRICHTEXT_SETPROPERTIES_WITH_UNDO) [virtual]

Sets the properties for the given range, passing flags to determine how the attributes are set.

You can merge properties or replace them.

The end point of range is specified as the last character position of the span of text, plus one. So, for example, to set the properties for a character at position 5, use the range (5,6).

flags may contain a bit list of the following values:

- wxRICHTEXT_SETPROPERTIES_NONE: no flag.
- wxRICHTEXT_SETPROPERTIES_WITH_UNDO: specifies that this operation should be undoable.
- wxRICHTEXT_SETPROPERTIES_PARAGRAPHS_ONLY: specifies that the properties should only be applied to paragraphs, and not the content.
- wxRICHTEXT_SETPROPERTIES_CHARACTERS_ONLY: specifies that the properties should only be applied to characters, and not the paragraph.

- wxRICHTEXT_SETPROPERTIES_RESET: resets (clears) the existing properties before applying the new properties.
- wxRICHTEXT_SETPROPERTIES_REMOVE: removes the specified properties.

void wxRichTextParagraphLayoutBox::SetRichTextCtrl (wxRichTextCtrl * *ctrl*) [inline]

Associates a control with the buffer, for operations that for example require refreshing the window.

virtual bool wxRichTextParagraphLayoutBox::SetStyle (const wxRichTextRange & *range,* const wxRichTextAttr & *style,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO) [virtual]

Sets the attributes for the given range.

Pass flags to determine how the attributes are set.

The end point of range is specified as the last character position of the span of text. So, for example, to set the style for a character at position 5, use the range (5,5). This differs from the wxRichTextCtrl API, where you would specify (5,6).

flags may contain a bit list of the following values:

- wxRICHTEXT_SETSTYLE_NONE: no style flag.
- wxRICHTEXT_SETSTYLE_WITH_UNDO: specifies that this operation should be undoable.
- wxRICHTEXT_SETSTYLE_OPTIMIZE: specifies that the style should not be applied if the combined style at this point is already the style in question.
- wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY: specifies that the style should only be applied to paragraphs, and not the content. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY: specifies that the style should only be applied to characters, and not the paragraph. This allows content styling to be preserved independently from that of e.g. a named paragraph style.
- wxRICHTEXT_SETSTYLE_RESET: resets (clears) the existing style before applying the new style.
- wxRICHTEXT_SETSTYLE_REMOVE: removes the specified style. Only the style flags are used in this operation.

virtual void wxRichTextParagraphLayoutBox::SetStyle (wxRichTextObject * *obj,* const wxRichTextAttr & *textAttr,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO) [virtual]

Sets the attributes for the given object only, for example the box attributes for a text box.

virtual bool wxRichTextParagraphLayoutBox::UpdateFloatingObjects (const wxRect & *availableRect,* wxRichTextObject * *untilObj* = NULL) [virtual]

Gather information about floating objects.

If untilObj is non-NULL, will stop getting information if the current object is this, since we will collect the rest later.

virtual void wxRichTextParagraphLayoutBox::UpdateRanges ( ) [virtual]

Calculate ranges.

virtual long wxRichTextParagraphLayoutBox::XYToPosition (long *x*, long *y*) const `[virtual]`

Converts zero-based line column and paragraph number to a position.

21.637.4 Member Data Documentation

wxRichTextCtrl* wxRichTextParagraphLayoutBox::m_ctrl `[protected]`

wxRichTextAttr wxRichTextParagraphLayoutBox::m_defaultAttributes `[protected]`

wxRichTextFloatCollector* wxRichTextParagraphLayoutBox::m_floatCollector `[protected]`

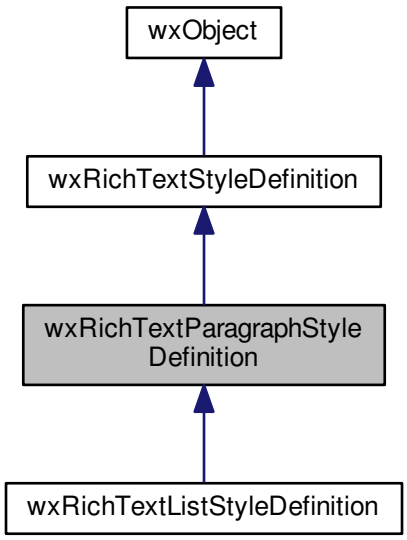
wxRichTextRange wxRichTextParagraphLayoutBox::m_invalidRange `[protected]`

bool wxRichTextParagraphLayoutBox::m_partialParagraph `[protected]`

21.638 wxRichTextParagraphStyleDefinition Class Reference

`#include <wx/richtext/richtextstyles.h>`

Inheritance diagram for wxRichTextParagraphStyleDefinition:



21.638.1 Detailed Description

This class represents a paragraph style definition, usually added to a wxRichTextStyleSheet.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextParagraphStyleDefinition (const wxString &name=wxEmptyString)

  Constructor.

- virtual ~wxRichTextParagraphStyleDefinition ()

  Destructor.

- const wxString & GetNextStyle () const

  Returns the style that should normally follow this style.

- void SetNextStyle (const wxString &name)

  Sets the style that should normally follow this style.

Additional Inherited Members

21.638.2 Constructor & Destructor Documentation

wxRichTextParagraphStyleDefinition::wxRichTextParagraphStyleDefinition (const wxString & *name* = wxEmptyString)

Constructor.

virtual wxRichTextParagraphStyleDefinition::~wxRichTextParagraphStyleDefinition ( ) [virtual]

Destructor.

21.638.3 Member Function Documentation

const wxString& wxRichTextParagraphStyleDefinition::GetNextStyle ( ) const

Returns the style that should normally follow this style.

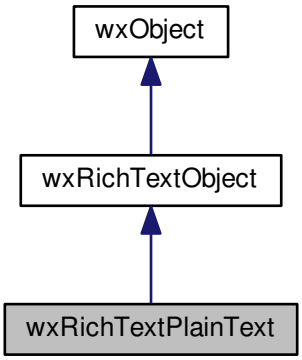
void wxRichTextParagraphStyleDefinition::SetNextStyle (const wxString & *name*)

Sets the style that should normally follow this style.

21.639 wxRichTextPlainText Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextPlainText:



21.639.1 Detailed Description

This object represents a single piece of text.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextPlainText (const wxString &text=wxEmptyString, wxRichTextObject *parent=NULL, wxRichTextAttr *style=NULL)

  Constructor.
- wxRichTextPlainText (const wxRichTextPlainText &obj)

  Copy constructor.
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

  Lay the item out at the specified position with the given size constraint.
- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.
- virtual wxString GetTextForRange (const wxRichTextRange &range) const

  Returns any text in this object for the given range.

- virtual wxRichTextObject * DoSplit (long pos)

  Do a split from pos, returning an object containing the second part, and setting the first part in 'this'.
- virtual void CalculateRange (long start, long &end)

  Calculates the range of the object.
- virtual bool DeleteRange (const wxRichTextRange &range)

  Deletes the given range.
- virtual bool IsEmpty () const

  Returns true if the object is empty.
- virtual bool CanMerge (wxRichTextObject *object, wxRichTextDrawingContext &context) const

  Returns true if this object can merge itself with the given one.
- virtual bool Merge (wxRichTextObject *object, wxRichTextDrawingContext &context)

  Returns true if this object merged itself with the given one.
- virtual bool CanSplit (wxRichTextDrawingContext &context) const

  Returns true if this object can potentially be split, by virtue of having different virtual attributes for individual sub-objects.
- virtual wxRichTextObject * Split (wxRichTextDrawingContext &context)

  Returns the final object in the split objects if this object was split due to differences between sub-object virtual attributes.
- virtual void Dump (wxTextOutputStream &stream)

  Dump object data to the given output stream for debugging.
- long GetFirstLineBreakPosition (long pos)

  Get the first position from pos that has a line break character.
- virtual bool UsesParagraphAttributes () const

  Does this object take note of paragraph attributes? Text and image objects don't.
- virtual bool ImportFromXML (wxRichTextBuffer *buffer, wxXmlNode *node, wxRichTextXMLHandler *handler, bool *recurse)

  Imports this object from XML.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- const wxString & GetText () const

  Returns the text.
- void SetText (const wxString &text)

  Sets the text.
- void Copy (const wxRichTextPlainText &obj)
- virtual wxRichTextObject * Clone () const

  Clones the object.

Protected Attributes

- wxString m_text

Private Member Functions

- bool DrawTabbedString (wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect, wxString &str, wxCoord &x, wxCoord &y, bool selected)

Additional Inherited Members

21.639.2 Constructor & Destructor Documentation

wxRichTextPlainText::wxRichTextPlainText (const wxString & *text* = wxEmptyString, wxRichTextObject * *parent* = NULL, wxRichTextAttr * *style* = NULL)

Constructor.

wxRichTextPlainText::wxRichTextPlainText (const wxRichTextPlainText & *obj*) [inline]

Copy constructor.

21.639.3 Member Function Documentation

virtual void wxRichTextPlainText::CalculateRange (long *start*, long & *end*) [virtual]

Calculates the range of the object.

By default, guess that the object is 1 unit long.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::CanMerge (wxRichTextObject * *object*, wxRichTextDrawingContext & *context*) const [virtual]

Returns true if this object can merge itself with the given one.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::CanSplit (wxRichTextDrawingContext & *context*) const [virtual]

Returns true if this object can potentially be split, by virtue of having different virtual attributes for individual sub-objects.

Reimplemented from wxRichTextObject.

virtual wxRichTextObject* wxRichTextPlainText::Clone ( ) const [inline], [virtual]

Clones the object.

Reimplemented from wxRichTextObject.

void wxRichTextPlainText::Copy (const wxRichTextPlainText & *obj*)

virtual bool wxRichTextPlainText::DeleteRange (const wxRichTextRange & *range*) [virtual]

Deletes the given range.

Reimplemented from wxRichTextObject.

virtual wxRichTextObject* wxRichTextPlainText::DoSplit (long *pos*) [virtual]

Do a split from *pos*, returning an object containing the second part, and setting the first part in 'this'.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::Draw (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) [virtual]

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Implements wxRichTextObject.

bool wxRichTextPlainText::DrawTabbedString (wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect,* wxString & *str,* wxCoord & *x,* wxCoord & *y,* bool *selected*) [private]

virtual void wxRichTextPlainText::Dump (wxTextOutputStream & *stream*) [virtual]

Dump object data to the given output stream for debugging.

Reimplemented from wxRichTextObject.

long wxRichTextPlainText::GetFirstLineBreakPosition (long *pos*)

Get the first position from pos that has a line break character.

virtual bool wxRichTextPlainText::GetRangeSize (const wxRichTextRange & *range,* wxSize & *size,* int & *descent,* wxDC & *dc,* wxRichTextDrawingContext & *context,* int *flags,* const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const [virtual]

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Implements wxRichTextObject.

const wxString& wxRichTextPlainText::GetText ( ) const [inline]

Returns the text.

virtual wxString wxRichTextPlainText::GetTextForRange (const wxRichTextRange & *range*) const [virtual]

Returns any text in this object for the given range.

Reimplemented from wxRichTextObject.

virtual wxString wxRichTextPlainText::GetXMLNodeName ( ) const [inline], [virtual]

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::ImportFromXML (wxRichTextBuffer * *buffer,* wxXmlNode * *node,* wxRichTextXMLHandler * *handler,* bool * *recurse*) [virtual]

Imports this object from XML.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::IsEmpty ( ) const `[inline], [virtual]`

Returns true if the object is empty.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::Layout (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style*) `[virtual]`

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Implements wxRichTextObject.

virtual bool wxRichTextPlainText::Merge (wxRichTextObject * *object,* wxRichTextDrawingContext & *context*) `[virtual]`

Returns true if this object merged itself with the given one.

The calling code will then delete the given object.

Reimplemented from wxRichTextObject.

void wxRichTextPlainText::SetText (const wxString & *text*) `[inline]`

Sets the text.

virtual wxRichTextObject* wxRichTextPlainText::Split (wxRichTextDrawingContext & *context*) `[virtual]`

Returns the final object in the split objects if this object was split due to differences between sub-object virtual attributes.

Returns itself if it was not split.

Reimplemented from wxRichTextObject.

virtual bool wxRichTextPlainText::UsesParagraphAttributes ( ) const `[inline], [virtual]`

Does this object take note of paragraph attributes? Text and image objects don't.

Reimplemented from wxRichTextObject.

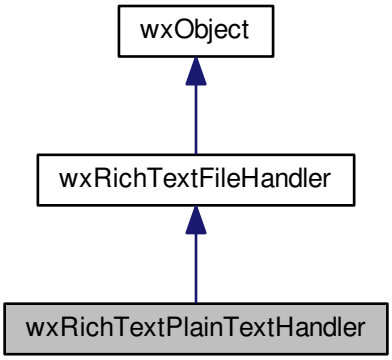
21.639.4 Member Data Documentation

wxString wxRichTextPlainText::m_text `[protected]`

21.640 wxRichTextPlainTextHandler Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextPlainTextHandler:



21.640.1 Detailed Description

Implements saving a buffer to plain text.

Library: wxRichText

Category: Rich Text

See also

wxRichTextFileHandler, wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextPlainTextHandler (const wxString &name="Text", const wxString &ext="txt", wxRichTextFileType type=wxRICHTEXT_TYPE_TEXT)
- virtual bool CanSave () const

  Returns true if we can save using this handler.
- virtual bool CanLoad () const

  Returns true if we can load using this handler.

Protected Member Functions

- virtual bool DoLoadFile (wxRichTextBuffer *buffer, wxInputStream &stream)

  Override to load content from stream into buffer.
- virtual bool DoSaveFile (wxRichTextBuffer *buffer, wxOutputStream &stream)

  Override to save content to stream from buffer.

Additional Inherited Members

21.640.2 Constructor & Destructor Documentation

wxRichTextPlainTextHandler::wxRichTextPlainTextHandler (const wxString & *name* = "Text", const wxString & *ext* = "txt", wxRichTextFileType *type* = wxRICHTEXT_TYPE_TEXT) `[inline]`

21.640.3 Member Function Documentation

virtual bool wxRichTextPlainTextHandler::CanLoad ( ) const `[inline], [virtual]`

Returns true if we can load using this handler.

Reimplemented from wxRichTextFileHandler.

virtual bool wxRichTextPlainTextHandler::CanSave ( ) const `[inline], [virtual]`

Returns true if we can save using this handler.

Reimplemented from wxRichTextFileHandler.

virtual bool wxRichTextPlainTextHandler::DoLoadFile (wxRichTextBuffer * *buffer*, wxInputStream & *stream*) `[protected], [virtual]`

Override to load content from *stream* into *buffer*.

Implements wxRichTextFileHandler.

virtual bool wxRichTextPlainTextHandler::DoSaveFile (wxRichTextBuffer * *buffer*, wxOutputStream & *stream*) `[protected], [virtual]`

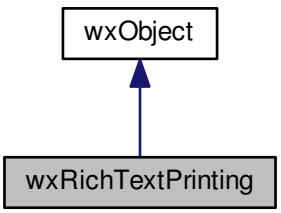
Override to save content to *stream* from *buffer*.

Implements wxRichTextFileHandler.

21.641 wxRichTextPrinting Class Reference

`#include <wx/richtext/richtextprint.h>`

Inheritance diagram for wxRichTextPrinting:



21.641.1 Detailed Description

This class provides a simple interface for performing wxRichTextBuffer printing and previewing.

It uses wxRichTextPrintout for layout and rendering.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextPrinting (const wxString &name="Printing", wxWindow *parentWindow=NULL)

  Constructor.
- wxString GetFooterText (wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE) const

  A convenience function to get the footer text.
- const wxRichTextHeaderFooterData & GetHeaderFooterData () const

  Returns the internal wxRichTextHeaderFooterData object.
- wxString GetHeaderText (wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE) const

  A convenience function to get the header text.
- wxPageSetupDialogData * GetPageSetupData ()

  Returns a pointer to the internal page setup data.
- wxWindow * GetParentWindow () const

  Returns the parent window to be used for the preview window and printing wait dialog.
- const wxRect & GetPreviewRect () const

  Returns the dimensions to be used for the preview window.
- wxPrintData * GetPrintData ()

  Returns a pointer to the internal print data.
- const wxString & GetTitle () const

  Returns the title of the preview window or printing wait caption.
- void PageSetup ()

  Shows the page setup dialog.
- bool PreviewBuffer (const wxRichTextBuffer &buffer)

  Shows a preview window for the given buffer.
- bool PreviewFile (const wxString &richTextFile)

  Shows a preview window for the given file.
- bool PrintBuffer (const wxRichTextBuffer &buffer, bool showPrintDialog=true)

  Prints the given buffer.
- bool PrintFile (const wxString &richTextFile, bool showPrintDialog=true)

  Prints the given file.
- void SetFooterText (const wxString &text, wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE)

  A convenience function to set the footer text.
- void SetHeaderFooterData (const wxRichTextHeaderFooterData &data)

  Sets the internal wxRichTextHeaderFooterData object.
- void SetHeaderFooterFont (const wxFont &font)

  Sets the wxRichTextHeaderFooterData font.
- void SetHeaderFooterTextColour (const wxColour &colour)

Sets the wxRichTextHeaderFooterData text colour.

- void SetHeaderText (const wxString &text, wxRichTextOddEvenPage page=wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation location=wxRICHTEXT_PAGE_CENTRE)

  A convenience function to set the header text.

- void SetPageSetupData (const wxPageSetupDialogData &pageSetupData)

  Sets the page setup data.

- void SetParentWindow (wxWindow *parent)

  Sets the parent window to be used for the preview window and printing wait dialog.

- void SetPreviewRect (const wxRect &rect)

  Sets the dimensions to be used for the preview window.

- void SetPrintData (const wxPrintData &printData)

  Sets the print data.

- void SetShowOnFirstPage (bool show)

  Pass true to show the header and footer on the first page.

- void SetTitle (const wxString &title)

  Pass the title of the preview window or printing wait caption.

Additional Inherited Members

21.641.2 Constructor & Destructor Documentation

wxRichTextPrinting::wxRichTextPrinting (const wxString & *name* = "Printing", wxWindow * *parentWindow* = NULL)

Constructor.

Optionally pass a title to be used in the preview frame and printing wait dialog, and also a parent window for these windows.

21.641.3 Member Function Documentation

wxString wxRichTextPrinting::GetFooterText (wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE) const

A convenience function to get the footer text.

See wxRichTextHeaderFooterData for details.

const wxRichTextHeaderFooterData& wxRichTextPrinting::GetHeaderFooterData ( ) const

Returns the internal wxRichTextHeaderFooterData object.

wxString wxRichTextPrinting::GetHeaderText (wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_EVEN, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE) const

A convenience function to get the header text.

See wxRichTextHeaderFooterData for details.

wxPageSetupDialogData* wxRichTextPrinting::GetPageSetupData ( )

Returns a pointer to the internal page setup data.

wxWindow* wxRichTextPrinting::GetParentWindow ( ) const

Returns the parent window to be used for the preview window and printing wait dialog.

const wxRect& wxRichTextPrinting::GetPreviewRect ( ) const

Returns the dimensions to be used for the preview window.

wxPrintData* wxRichTextPrinting::GetPrintData ( )

Returns a pointer to the internal print data.

const wxString& wxRichTextPrinting::GetTitle ( ) const

Returns the title of the preview window or printing wait caption.

void wxRichTextPrinting::PageSetup ( )

Shows the page setup dialog.

bool wxRichTextPrinting::PreviewBuffer (const wxRichTextBuffer & *buffer*)

Shows a preview window for the given buffer.

The function takes its own copy of *buffer*.

bool wxRichTextPrinting::PreviewFile (const wxString & *richTextFile*)

Shows a preview window for the given file.

richTextFile can be a text file or XML file, or other file depending on the available file handlers.

bool wxRichTextPrinting::PrintBuffer (const wxRichTextBuffer & *buffer*, bool *showPrintDialog* = `true`)

Prints the given buffer.

The function takes its own copy of *buffer*. *showPrintDialog* can be true to show the print dialog, or false to print quietly.

bool wxRichTextPrinting::PrintFile (const wxString & *richTextFile*, bool *showPrintDialog* = `true`)

Prints the given file.

richTextFile can be a text file or XML file, or other file depending on the available file handlers. *showPrintDialog* can be true to show the print dialog, or false to print quietly.

void wxRichTextPrinting::SetFooterText (const wxString & *text*, wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE)

A convenience function to set the footer text.

See wxRichTextHeaderFooterData for details.

void wxRichTextPrinting::SetHeaderFooterData (const wxRichTextHeaderFooterData & *data*)

Sets the internal wxRichTextHeaderFooterData object.

void wxRichTextPrinting::SetHeaderFooterFont (const wxFont & *font*)

Sets the wxRichTextHeaderFooterData font.

void wxRichTextPrinting::SetHeaderFooterTextColour (const wxColour & *colour*)

Sets the wxRichTextHeaderFooterData text colour.

void wxRichTextPrinting::SetHeaderText (const wxString & *text*, wxRichTextOddEvenPage *page* = wxRICHTEXT_PAGE_ALL, wxRichTextPageLocation *location* = wxRICHTEXT_PAGE_CENTRE)

A convenience function to set the header text.

See wxRichTextHeaderFooterData for details.

void wxRichTextPrinting::SetPageSetupData (const wxPageSetupDialogData & *pageSetupData*)

Sets the page setup data.

void wxRichTextPrinting::SetParentWindow (wxWindow * *parent*)

Sets the parent window to be used for the preview window and printing wait dialog.

void wxRichTextPrinting::SetPreviewRect (const wxRect & *rect*)

Sets the dimensions to be used for the preview window.

void wxRichTextPrinting::SetPrintData (const wxPrintData & *printData*)

Sets the print data.

void wxRichTextPrinting::SetShowOnFirstPage (bool *show*)

Pass true to show the header and footer on the first page.

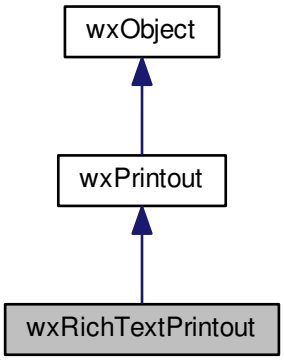
void wxRichTextPrinting::SetTitle (const wxString & *title*)

Pass the title of the preview window or printing wait caption.

21.642 wxRichTextPrintout Class Reference

```
#include <wx/richtext/richtextprint.h>
```

Inheritance diagram for wxRichTextPrintout:



21.642.1 Detailed Description

This class implements print layout for wxRichTextBuffer.

Instead of using it directly, you should normally use the wxRichTextPrinting class.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextPrintout (const wxString &title="Printout")

  Constructor.
- void CalculateScaling (wxDC *dc, wxRect &textRect, wxRect &headerRect, wxRect &footerRect)

  Calculates scaling and text, header and footer rectangles.
- const wxRichTextHeaderFooterData & GetHeaderFooterData () const

  Returns the header and footer data associated with the printout.
- virtual void GetPageInfo (int *minPage, int *maxPage, int *selPageFrom, int *selPageTo)

  Gets the page information.
- wxRichTextBuffer * GetRichTextBuffer () const

  Returns a pointer to the buffer being rendered.
- virtual bool HasPage (int page)

  Returns true if the given page exists in the printout.
- virtual void OnPreparePrinting ()

  Prepares for printing, laying out the buffer and calculating pagination.
- virtual bool OnPrintPage (int page)

  Does the actual printing for this page.
- void SetHeaderFooterData (const wxRichTextHeaderFooterData &data)

  Sets the header and footer data associated with the printout.

- void SetMargins (int top=254, int bottom=254, int left=254, int right=254)

  Sets margins in 10ths of millimetre.
- void SetRichTextBuffer (wxRichTextBuffer *buffer)

  Sets the buffer to print.

Additional Inherited Members

21.642.2 Constructor & Destructor Documentation

wxRichTextPrintout::wxRichTextPrintout (const wxString & *title* = "Printout")

Constructor.

21.642.3 Member Function Documentation

void wxRichTextPrintout::CalculateScaling (wxDC * *dc,* wxRect & *textRect,* wxRect & *headerRect,* wxRect & *footerRect*)

Calculates scaling and text, header and footer rectangles.

const wxRichTextHeaderFooterData& wxRichTextPrintout::GetHeaderFooterData ( ) const

Returns the header and footer data associated with the printout.

virtual void wxRichTextPrintout::GetPageInfo (int * *minPage,* int * *maxPage,* int * *selPageFrom,* int * *selPageTo*) [virtual]

Gets the page information.

Reimplemented from wxPrintout.

wxRichTextBuffer* wxRichTextPrintout::GetRichTextBuffer ( ) const

Returns a pointer to the buffer being rendered.

virtual bool wxRichTextPrintout::HasPage (int *page*) [virtual]

Returns true if the given page exists in the printout.

Reimplemented from wxPrintout.

virtual void wxRichTextPrintout::OnPreparePrinting ( ) [virtual]

Prepares for printing, laying out the buffer and calculating pagination.

Reimplemented from wxPrintout.

virtual bool wxRichTextPrintout::OnPrintPage (int *page*) [virtual]

Does the actual printing for this page.

Implements wxPrintout.

void wxRichTextPrintout::SetHeaderFooterData (const wxRichTextHeaderFooterData & *data*)

Sets the header and footer data associated with the printout.

void wxRichTextPrintout::SetMargins (int *top* = 254, int *bottom* = 254, int *left* = 254, int *right* = 254)

Sets margins in 10ths of millimetre.

Defaults to 1 inch for margins.

void wxRichTextPrintout::SetRichTextBuffer (wxRichTextBuffer * *buffer*)

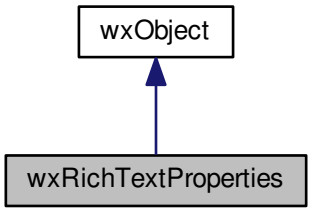
Sets the buffer to print.

wxRichTextPrintout does not manage this pointer; it should be managed by the calling code, such as wxRichText↩Printing.

21.643 wxRichTextProperties Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextProperties:



21.643.1 Detailed Description

A simple property class using wxVariants.

This is used to give each rich text object the ability to store custom properties that can be used by the application.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextObject, wxRichTextCtrl

Public Member Functions

- wxRichTextProperties ()

  Default constructor.
- wxRichTextProperties (const wxRichTextProperties &props)

  Copy constructor.
- void operator= (const wxRichTextProperties &props)

  Assignment operator.
- bool operator== (const wxRichTextProperties &props) const

  Equality operator.
- void Copy (const wxRichTextProperties &props)

  Copies from props.
- const wxVariant & operator[] (size_t idx) const

  Returns the variant at the given index.
- wxVariant & operator[] (size_t idx)

  Returns the variant at the given index.
- void Clear ()

  Clears the properties.
- const wxRichTextVariantArray & GetProperties () const

  Returns the array of variants implementing the properties.
- wxRichTextVariantArray & GetProperties ()

  Returns the array of variants implementing the properties.
- void SetProperties (const wxRichTextVariantArray &props)

  Sets the array of variants.
- wxArrayString GetPropertyNames () const

  Returns all the property names.
- size_t GetCount () const

  Returns a count of the properties.
- bool HasProperty (const wxString &name) const

  Returns true if the given property is found.
- int Find (const wxString &name) const

  Finds the given property.
- bool Remove (const wxString &name)

  Removes the given property.
- const wxVariant & GetProperty (const wxString &name) const

  Gets the property variant by name.
- wxVariant * FindOrCreateProperty (const wxString &name)

  Finds or creates a property with the given name, returning a pointer to the variant.
- wxString GetPropertyString (const wxString &name) const

  Gets the value of the named property as a string.
- long GetPropertyLong (const wxString &name) const

  Gets the value of the named property as a long integer.
- bool GetPropertyBool (const wxString &name) const

  Gets the value of the named property as a boolean.
- double GetPropertyDouble (const wxString &name) const

  Gets the value of the named property as a double.
- void SetProperty (const wxVariant &variant)

  Sets the property by passing a variant which contains a name and value.
- void SetProperty (const wxString &name, const wxVariant &variant)

  Sets a property by name and variant.
- void SetProperty (const wxString &name, const wxString &value)

Sets a property by name and string value.

- void SetProperty (const wxString &name, const wxChar *value)

  Sets a property by name and wxChar value.*

- void SetProperty (const wxString &name, long value)

  Sets property by name and long integer value.

- void SetProperty (const wxString &name, double value)

  Sets property by name and double value.

- void SetProperty (const wxString &name, bool value)

  Sets property by name and boolean value.

- void RemoveProperties (const wxRichTextProperties &properties)

  Removes the given properties from these properties.

- void MergeProperties (const wxRichTextProperties &properties)

  Merges the given properties with these properties.

Protected Attributes

- wxRichTextVariantArray m_properties

Additional Inherited Members

21.643.2 Constructor & Destructor Documentation

wxRichTextProperties::wxRichTextProperties ( ) [inline]

Default constructor.

wxRichTextProperties::wxRichTextProperties (const wxRichTextProperties & *props*) [inline]

Copy constructor.

21.643.3 Member Function Documentation

void wxRichTextProperties::Clear ( ) [inline]

Clears the properties.

void wxRichTextProperties::Copy (const wxRichTextProperties & *props*) [inline]

Copies from *props*.

int wxRichTextProperties::Find (const wxString & *name*) const

Finds the given property.

wxVariant* wxRichTextProperties::FindOrCreateProperty (const wxString & *name*)

Finds or creates a property with the given name, returning a pointer to the variant.

size_t wxRichTextProperties::GetCount ( ) const `[inline]`

Returns a count of the properties.

const wxRichTextVariantArray& wxRichTextProperties::GetProperties ( ) const `[inline]`

Returns the array of variants implementing the properties.

wxRichTextVariantArray& wxRichTextProperties::GetProperties ( ) `[inline]`

Returns the array of variants implementing the properties.

const wxVariant& wxRichTextProperties::GetProperty (const wxString & *name*) const

Gets the property variant by name.

bool wxRichTextProperties::GetPropertyBool (const wxString & *name*) const

Gets the value of the named property as a boolean.

double wxRichTextProperties::GetPropertyDouble (const wxString & *name*) const

Gets the value of the named property as a double.

long wxRichTextProperties::GetPropertyLong (const wxString & *name*) const

Gets the value of the named property as a long integer.

wxArrayString wxRichTextProperties::GetPropertyNames ( ) const

Returns all the property names.

wxString wxRichTextProperties::GetPropertyString (const wxString & *name*) const

Gets the value of the named property as a string.

bool wxRichTextProperties::HasProperty (const wxString & *name*) const `[inline]`

Returns true if the given property is found.

void wxRichTextProperties::MergeProperties (const wxRichTextProperties & *properties*)

Merges the given properties with these properties.

void wxRichTextProperties::operator= (const wxRichTextProperties & *props*) `[inline]`

Assignment operator.

bool wxRichTextProperties::operator== (const wxRichTextProperties & *props*) const

Equality operator.

const wxVariant& wxRichTextProperties::operator[] (size_t *idx*) const `[inline]`

Returns the variant at the given index.

wxVariant& wxRichTextProperties::operator[] (size_t *idx*) `[inline]`

Returns the variant at the given index.

bool wxRichTextProperties::Remove (const wxString & *name*)

Removes the given property.

void wxRichTextProperties::RemoveProperties (const wxRichTextProperties & *properties*)

Removes the given properties from these properties.

void wxRichTextProperties::SetProperties (const wxRichTextVariantArray & *props*) `[inline]`

Sets the array of variants.

void wxRichTextProperties::SetProperty (const wxVariant & *variant*)

Sets the property by passing a variant which contains a name and value.

void wxRichTextProperties::SetProperty (const wxString & *name,* const wxVariant & *variant*)

Sets a property by name and variant.

void wxRichTextProperties::SetProperty (const wxString & *name,* const wxString & *value*)

Sets a property by name and string value.

void wxRichTextProperties::SetProperty (const wxString & *name,* const wxChar * *value*)

Sets a property by name and wxChar* value.

void wxRichTextProperties::SetProperty (const wxString & *name,* long *value*)

Sets property by name and long integer value.

void wxRichTextProperties::SetProperty (const wxString & *name,* double *value*)

Sets property by name and double value.

void wxRichTextProperties::SetProperty (const wxString & *name,* bool *value*)

Sets property by name and boolean value.

21.643.4 Member Data Documentation

wxRichTextVariantArray wxRichTextProperties::m_properties `[protected]`

21.644 wxRichTextRange Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.644.1 Detailed Description

This stores beginning and end positions for a range of data.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextRange ()
  Default constructor.
- wxRichTextRange (long start, long end)
  Constructor taking start and end positions.
- wxRichTextRange (const wxRichTextRange &range)
  Copy constructor.
- ~wxRichTextRange ()
- void operator= (const wxRichTextRange &range)
  Assigns range to this range.
- bool operator== (const wxRichTextRange &range) const
  Equality operator.
- bool operator!= (const wxRichTextRange &range) const
  Inequality operator.
- wxRichTextRange operator- (const wxRichTextRange &range) const
  Subtracts a range from this range.
- wxRichTextRange operator+ (const wxRichTextRange &range) const
  Adds a range to this range.
- void SetRange (long start, long end)
  Sets the range start and end positions.
- void SetStart (long start)
  Sets the start position.
- long GetStart () const
  Returns the start position.

- void SetEnd (long end)

  Sets the end position.
- long GetEnd () const

  Gets the end position.
- bool IsOutside (const wxRichTextRange &range) const

  Returns true if this range is completely outside range.
- bool IsWithin (const wxRichTextRange &range) const

  Returns true if this range is completely within range.
- bool Contains (long pos) const

  Returns true if pos was within the range.
- bool LimitTo (const wxRichTextRange &range)

  Limit this range to be within range.
- long GetLength () const

  Gets the length of the range.
- void Swap ()

  Swaps the start and end.
- wxRichTextRange ToInternal () const

  Converts the API-standard range, whose end is one past the last character in the range, to the internal form, which uses the first and last character positions of the range.
- wxRichTextRange FromInternal () const

  Converts the internal range, which uses the first and last character positions of the range, to the API-standard range, whose end is one past the last character in the range.

Protected Attributes

- long m_start
- long m_end

21.644.2 Constructor & Destructor Documentation

wxRichTextRange::wxRichTextRange ( ) [inline]

Default constructor.

wxRichTextRange::wxRichTextRange (long *start,* long *end*) [inline]

Constructor taking start and end positions.

wxRichTextRange::wxRichTextRange (const wxRichTextRange & *range*) [inline]

Copy constructor.

wxRichTextRange::~wxRichTextRange ( ) [inline]

21.644.3 Member Function Documentation

bool wxRichTextRange::Contains (long *pos*) const [inline]

Returns true if *pos* was within the range.

Does not match if the range is empty.

wxRichTextRange wxRichTextRange::FromInternal ( ) const `[inline]`

Converts the internal range, which uses the first and last character positions of the range, to the API-standard range, whose end is one past the last character in the range.

In other words, one is added to the end position. (n, n+1) is the range of a single character.

long wxRichTextRange::GetEnd ( ) const `[inline]`

Gets the end position.

long wxRichTextRange::GetLength ( ) const `[inline]`

Gets the length of the range.

long wxRichTextRange::GetStart ( ) const `[inline]`

Returns the start position.

bool wxRichTextRange::IsOutside (const wxRichTextRange & *range*) const `[inline]`

Returns true if this range is completely outside *range*.

bool wxRichTextRange::IsWithin (const wxRichTextRange & *range*) const `[inline]`

Returns true if this range is completely within *range*.

bool wxRichTextRange::LimitTo (const wxRichTextRange & *range*)

Limit this range to be within *range*.

bool wxRichTextRange::operator!= (const wxRichTextRange & *range*) const `[inline]`

Inequality operator.

wxRichTextRange wxRichTextRange::operator+ (const wxRichTextRange & *range*) const `[inline]`

Adds a range to this range.

wxRichTextRange wxRichTextRange::operator- (const wxRichTextRange & *range*) const `[inline]`

Subtracts a range from this range.

void wxRichTextRange::operator= (const wxRichTextRange & *range*) `[inline]`

Assigns *range* to this range.

bool wxRichTextRange::operator== (const wxRichTextRange & *range*) const [inline]

Equality operator.

Returns true if *range* is the same as this range.

void wxRichTextRange::SetEnd (long *end*) [inline]

Sets the end position.

void wxRichTextRange::SetRange (long *start,* long *end*) [inline]

Sets the range start and end positions.

void wxRichTextRange::SetStart (long *start*) [inline]

Sets the start position.

void wxRichTextRange::Swap ( ) [inline]

Swaps the start and end.

wxRichTextRange wxRichTextRange::ToInternal ( ) const [inline]

Converts the API-standard range, whose end is one past the last character in the range, to the internal form, which uses the first and last character positions of the range.

In other words, one is subtracted from the end position. (n, n) is the range of a single character.

21.644.4 Member Data Documentation

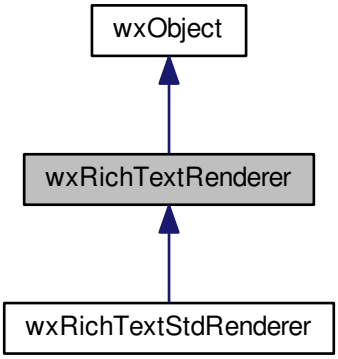
long wxRichTextRange::m_end [protected]

long wxRichTextRange::m_start [protected]

21.645 wxRichTextRenderer Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextRenderer:



21.645.1 Detailed Description

This class isolates some common drawing functionality.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextRenderer ()

  Constructor.
- virtual ~wxRichTextRenderer ()
- virtual bool DrawStandardBullet (wxRichTextParagraph *paragraph, wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect)=0

  Draws a standard bullet, as specified by the value of GetBulletName.
- virtual bool DrawTextBullet (wxRichTextParagraph *paragraph, wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect, const wxString &text)=0

  Draws a bullet that can be described by text, such as numbered or symbol bullets.
- virtual bool DrawBitmapBullet (wxRichTextParagraph *paragraph, wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect)=0

  Draws a bitmap bullet, where the bullet bitmap is specified by the value of GetBulletName.
- virtual bool EnumerateStandardBulletNames (wxArrayString &bulletNames)=0

  Enumerate the standard bullet names currently supported.

Additional Inherited Members

21.645.2 Constructor & Destructor Documentation

wxRichTextRenderer::wxRichTextRenderer ( ) [inline]

Constructor.

virtual wxRichTextRenderer::~wxRichTextRenderer ( ) [inline], [virtual]

21.645.3 Member Function Documentation

virtual bool wxRichTextRenderer::DrawBitmapBullet (wxRichTextParagraph * *paragraph,* wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect*) [pure virtual]

Draws a bitmap bullet, where the bullet bitmap is specified by the value of GetBulletName.

This function should be overridden.

Implemented in wxRichTextStdRenderer.

virtual bool wxRichTextRenderer::DrawStandardBullet (wxRichTextParagraph * *paragraph,* wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect*) [pure virtual]

Draws a standard bullet, as specified by the value of GetBulletName.

This function should be overridden.

Implemented in wxRichTextStdRenderer.

virtual bool wxRichTextRenderer::DrawTextBullet (wxRichTextParagraph * *paragraph,* wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect,* const wxString & *text*) [pure virtual]

Draws a bullet that can be described by text, such as numbered or symbol bullets.

This function should be overridden.

Implemented in wxRichTextStdRenderer.

virtual bool wxRichTextRenderer::EnumerateStandardBulletNames (wxArrayString & *bulletNames*) [pure virtual]

Enumerate the standard bullet names currently supported.

This function should be overridden.

Implemented in wxRichTextStdRenderer.

21.646 wxRichTextSelection Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.646.1 Detailed Description

Stores selection information.

The selection does not have to be contiguous, though currently non-contiguous selections are only supported for a range of table cells (a geometric block of cells can consist of a set of non-contiguous positions).

The selection consists of an array of ranges, and the container that is the context for the selection. It follows that a single selection object can only represent ranges with the same parent container.

Library: wxRichText

Category: Rich Text

See also

wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextSelection (const wxRichTextSelection &sel)

  Copy constructor.
- wxRichTextSelection (const wxRichTextRange &range, wxRichTextParagraphLayoutBox *container)

  Creates a selection from a range and a container.
- wxRichTextSelection ()

  Default constructor.
- void Reset ()

  Resets the selection.
- void Set (const wxRichTextRange &range, wxRichTextParagraphLayoutBox *container)

  Sets the selection.
- void Add (const wxRichTextRange &range)

  Adds a range to the selection.
- void Set (const wxRichTextRangeArray &ranges, wxRichTextParagraphLayoutBox *container)

  Sets the selections from an array of ranges and a container object.
- void Copy (const wxRichTextSelection &sel)

  Copies from sel.
- void operator= (const wxRichTextSelection &sel)

  Assignment operator.
- bool operator== (const wxRichTextSelection &sel) const

  Equality operator.
- wxRichTextRange operator[] (size_t i) const

  Index operator.
- wxRichTextRangeArray & GetRanges ()

  Returns the selection ranges.
- const wxRichTextRangeArray & GetRanges () const

  Returns the selection ranges.
- void SetRanges (const wxRichTextRangeArray &ranges)

  Sets the selection ranges.
- size_t GetCount () const

  Returns the number of ranges in the selection.
- wxRichTextRange GetRange (size_t i) const

  Returns the range at the given index.
- wxRichTextRange GetRange () const

  Returns the first range if there is one, otherwise wxRICHTEXT_NO_SELECTION.
- void SetRange (const wxRichTextRange &range)

  Sets a single range.
- wxRichTextParagraphLayoutBox * GetContainer () const

Returns the container for which the selection is valid.

- void SetContainer (wxRichTextParagraphLayoutBox ∗container)

  Sets the container for which the selection is valid.

- bool IsValid () const

  Returns true if the selection is valid.

- wxRichTextRangeArray GetSelectionForObject (wxRichTextObject ∗obj) const

  Returns the selection appropriate to the specified object, if any; returns an empty array if none at the level of the object's container.

- bool WithinSelection (long pos, wxRichTextObject ∗obj) const

  Returns true if the given position is within the selection.

- bool WithinSelection (long pos) const

  Returns true if the given position is within the selection.

Static Public Member Functions

- static bool WithinSelection (long pos, const wxRichTextRangeArray &ranges)

  Returns true if the given position is within the selection range.

- static bool WithinSelection (const wxRichTextRange &range, const wxRichTextRangeArray &ranges)

  Returns true if the given range is within the selection range.

Public Attributes

- wxRichTextRangeArray m_ranges
- wxRichTextParagraphLayoutBox ∗ m_container

21.646.2 Constructor & Destructor Documentation

wxRichTextSelection::wxRichTextSelection (const wxRichTextSelection & *sel*) `[inline]`

Copy constructor.

wxRichTextSelection::wxRichTextSelection (const wxRichTextRange & *range,* wxRichTextParagraphLayoutBox ∗ *container*) `[inline]`

Creates a selection from a range and a container.

wxRichTextSelection::wxRichTextSelection ( ) `[inline]`

Default constructor.

21.646.3 Member Function Documentation

void wxRichTextSelection::Add (const wxRichTextRange & *range*) `[inline]`

Adds a range to the selection.

void wxRichTextSelection::Copy (const wxRichTextSelection & *sel*) `[inline]`

Copies from *sel*.

wxRichTextParagraphLayoutBox* wxRichTextSelection::GetContainer ( ) const [inline]

Returns the container for which the selection is valid.

size_t wxRichTextSelection::GetCount ( ) const [inline]

Returns the number of ranges in the selection.

wxRichTextRange wxRichTextSelection::GetRange (size_t *i*) const [inline]

Returns the range at the given index.

wxRichTextRange wxRichTextSelection::GetRange ( ) const [inline]

Returns the first range if there is one, otherwise wxRICHTEXT_NO_SELECTION.

wxRichTextRangeArray& wxRichTextSelection::GetRanges ( ) [inline]

Returns the selection ranges.

const wxRichTextRangeArray& wxRichTextSelection::GetRanges ( ) const [inline]

Returns the selection ranges.

wxRichTextRangeArray wxRichTextSelection::GetSelectionForObject (wxRichTextObject * *obj*) const

Returns the selection appropriate to the specified object, if any; returns an empty array if none at the level of the object's container.

bool wxRichTextSelection::IsValid ( ) const [inline]

Returns true if the selection is valid.

void wxRichTextSelection::operator= (const wxRichTextSelection & *sel*) [inline]

Assignment operator.

bool wxRichTextSelection::operator== (const wxRichTextSelection & *sel*) const

Equality operator.

wxRichTextRange wxRichTextSelection::operator[] (size_t *i*) const [inline]

Index operator.

void wxRichTextSelection::Reset ( ) [inline]

Resets the selection.

void wxRichTextSelection::Set (const wxRichTextRange & *range,* wxRichTextParagraphLayoutBox * *container*) [inline]

Sets the selection.

void wxRichTextSelection::Set (const wxRichTextRangeArray & *ranges,* wxRichTextParagraphLayoutBox * *container*) [inline]

Sets the selections from an array of ranges and a container object.

void wxRichTextSelection::SetContainer (wxRichTextParagraphLayoutBox * *container*) [inline]

Sets the container for which the selection is valid.

void wxRichTextSelection::SetRange (const wxRichTextRange & *range*) [inline]

Sets a single range.

void wxRichTextSelection::SetRanges (const wxRichTextRangeArray & *ranges*) [inline]

Sets the selection ranges.

bool wxRichTextSelection::WithinSelection (long *pos,* wxRichTextObject * *obj*) const

Returns true if the given position is within the selection.

bool wxRichTextSelection::WithinSelection (long *pos*) const [inline]

Returns true if the given position is within the selection.

static bool wxRichTextSelection::WithinSelection (long *pos,* const wxRichTextRangeArray & *ranges*) [static]

Returns true if the given position is within the selection range.

static bool wxRichTextSelection::WithinSelection (const wxRichTextRange & *range,* const wxRichTextRangeArray & *ranges*) [static]

Returns true if the given range is within the selection range.

21.646.4 Member Data Documentation

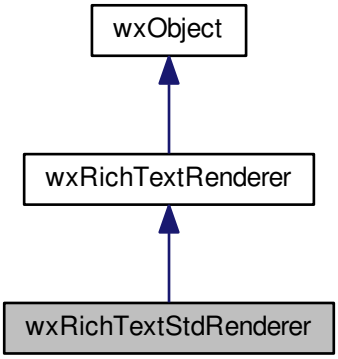
wxRichTextParagraphLayoutBox* wxRichTextSelection::m_container

wxRichTextRangeArray wxRichTextSelection::m_ranges

21.647 wxRichTextStdRenderer Class Reference

```
#include <wx/richtext/richtextbuffer.h>
```

Inheritance diagram for wxRichTextStdRenderer:



21.647.1 Detailed Description

The standard renderer for drawing bullets.

Library: wxRichText

Category: Rich Text

See also

wxRichTextRenderer, wxRichTextBuffer, wxRichTextCtrl

Public Member Functions

- wxRichTextStdRenderer ()

  Constructor.

- virtual bool DrawStandardBullet (wxRichTextParagraph *paragraph, wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect)

  Draws a standard bullet, as specified by the value of GetBulletName.

- virtual bool DrawTextBullet (wxRichTextParagraph *paragraph, wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect, const wxString &text)

  Draws a bullet that can be described by text, such as numbered or symbol bullets.

- virtual bool DrawBitmapBullet (wxRichTextParagraph *paragraph, wxDC &dc, const wxRichTextAttr &attr, const wxRect &rect)

  Draws a bitmap bullet, where the bullet bitmap is specified by the value of GetBulletName.

- virtual bool EnumerateStandardBulletNames (wxArrayString &bulletNames)

  Enumerate the standard bullet names currently supported.

Additional Inherited Members

21.647.2 Constructor & Destructor Documentation

wxRichTextStdRenderer::wxRichTextStdRenderer ( ) [inline]

Constructor.

21.647.3 Member Function Documentation

virtual bool wxRichTextStdRenderer::DrawBitmapBullet (wxRichTextParagraph * *paragraph,* wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect*) [virtual]

Draws a bitmap bullet, where the bullet bitmap is specified by the value of GetBulletName.

This function should be overridden.

Implements wxRichTextRenderer.

virtual bool wxRichTextStdRenderer::DrawStandardBullet (wxRichTextParagraph * *paragraph,* wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect*) [virtual]

Draws a standard bullet, as specified by the value of GetBulletName.

This function should be overridden.

Implements wxRichTextRenderer.

virtual bool wxRichTextStdRenderer::DrawTextBullet (wxRichTextParagraph * *paragraph,* wxDC & *dc,* const wxRichTextAttr & *attr,* const wxRect & *rect,* const wxString & *text*) [virtual]

Draws a bullet that can be described by text, such as numbered or symbol bullets.

This function should be overridden.

Implements wxRichTextRenderer.

virtual bool wxRichTextStdRenderer::EnumerateStandardBulletNames (wxArrayString & *bulletNames*) [virtual]

Enumerate the standard bullet names currently supported.

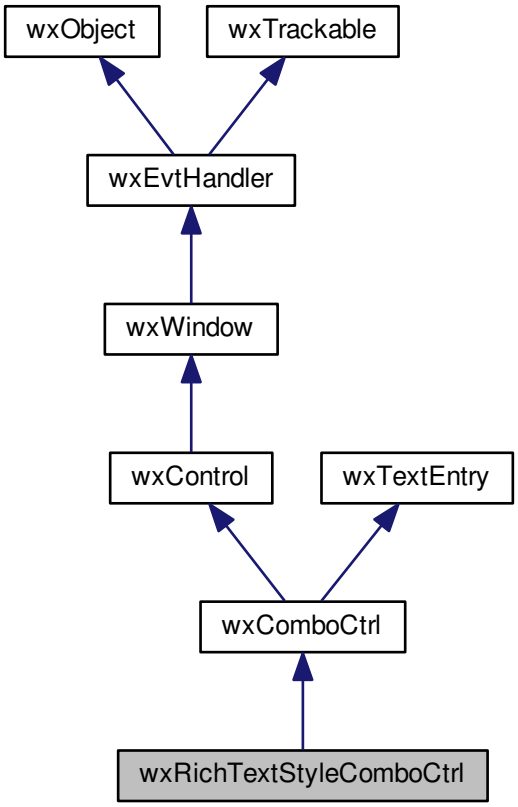
This function should be overridden.

Implements wxRichTextRenderer.

21.648 wxRichTextStyleComboCtrl Class Reference

`#include <wx/richtext/richtextstyles.h>`

Inheritance diagram for wxRichTextStyleComboCtrl:



21.648.1 Detailed Description

This is a combo control that can display the styles in a wxRichTextStyleSheet, and apply the selection to an associated wxRichTextCtrl.

See `samples/richtext` for an example of how to use it.

Library: wxRichText

Category: Rich Text

See also

wxRichTextStyleListBox, wxRichTextCtrl Overview

Public Member Functions

- wxRichTextStyleComboCtrl (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

Constructor.

- virtual ~wxRichTextStyleComboCtrl ()

  Destructor.

- wxRichTextCtrl * GetRichTextCtrl () const

  Returns the wxRichTextCtrl associated with this control.

- wxRichTextStyleSheet * GetStyleSheet () const

  Returns the style sheet associated with this control.

- void SetRichTextCtrl (wxRichTextCtrl *ctrl)

  Associates the control with a wxRichTextCtrl.

- void SetStyleSheet (wxRichTextStyleSheet *styleSheet)

  Associates the control with a style sheet.

- void UpdateStyles ()

  Updates the combo control from the associated style sheet.

Additional Inherited Members

21.648.2 Constructor & Destructor Documentation

wxRichTextStyleComboCtrl::wxRichTextStyleComboCtrl (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Constructor.

virtual wxRichTextStyleComboCtrl::~wxRichTextStyleComboCtrl ( ) [virtual]

Destructor.

21.648.3 Member Function Documentation

wxRichTextCtrl* wxRichTextStyleComboCtrl::GetRichTextCtrl ( ) const

Returns the wxRichTextCtrl associated with this control.

wxRichTextStyleSheet* wxRichTextStyleComboCtrl::GetStyleSheet ( ) const

Returns the style sheet associated with this control.

void wxRichTextStyleComboCtrl::SetRichTextCtrl (wxRichTextCtrl * *ctrl*)

Associates the control with a wxRichTextCtrl.

void wxRichTextStyleComboCtrl::SetStyleSheet (wxRichTextStyleSheet * *styleSheet*)

Associates the control with a style sheet.

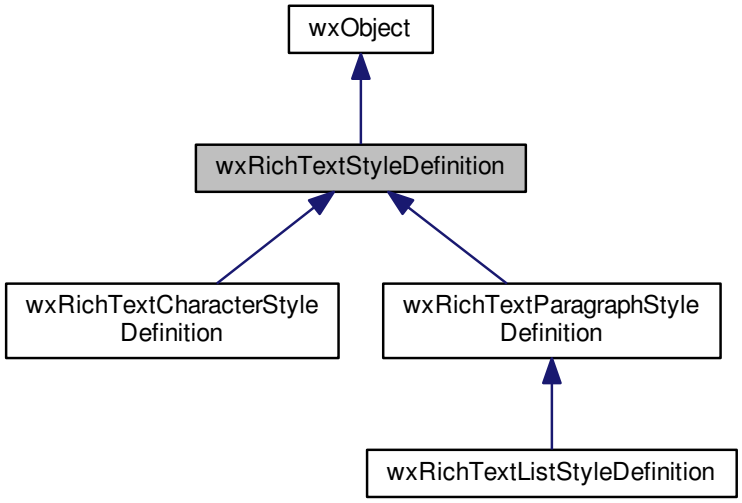
void wxRichTextStyleComboCtrl::UpdateStyles ( )

Updates the combo control from the associated style sheet.

21.649 wxRichTextStyleDefinition Class Reference

`#include <wx/richtext/richtextstyles.h>`

Inheritance diagram for wxRichTextStyleDefinition:



21.649.1 Detailed Description

This is a base class for paragraph and character styles.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextStyleDefinition (const wxString &name=wxEmptyString)
  Constructor.
- virtual ~wxRichTextStyleDefinition ()
  Destructor.
- const wxString & GetBaseStyle () const
  Returns the style on which this style is based.
- const wxString & GetDescription () const
  Returns the style's description.
- const wxString & GetName () const
  Returns the style name.
- virtual wxRichTextAttr GetStyleMergedWithBase (const wxRichTextStyleSheet *sheet) const
  Returns the style attributes combined with the attributes of the specified base style, if any.
- void SetBaseStyle (const wxString &name)

Sets the name of the style that this style is based on.

- void SetDescription (const wxString &descr)

  Sets the style description.
- void SetName (const wxString &name)

  Sets the name of the style.
- void SetStyle (const wxRichTextAttr &style)

  Sets the attributes for this style.
- wxRichTextProperties & GetProperties ()

  Returns the definition's properties.
- const wxRichTextProperties & GetProperties () const

  Returns the definition's properties.
- void SetProperties (const wxRichTextProperties &props)

  Sets the definition's properties.

- wxRichTextAttr GetStyle () const

  Returns the attributes associated with this style.
- const wxRichTextAttr GetStyle () const

  Returns the attributes associated with this style.

Additional Inherited Members

21.649.2 Constructor & Destructor Documentation

wxRichTextStyleDefinition::wxRichTextStyleDefinition (const wxString & *name* = wxEmptyString)

Constructor.

virtual wxRichTextStyleDefinition::~wxRichTextStyleDefinition ( ) [virtual]

Destructor.

21.649.3 Member Function Documentation

const wxString& wxRichTextStyleDefinition::GetBaseStyle ( ) const

Returns the style on which this style is based.

const wxString& wxRichTextStyleDefinition::GetDescription ( ) const

Returns the style's description.

const wxString& wxRichTextStyleDefinition::GetName ( ) const

Returns the style name.

wxRichTextProperties& wxRichTextStyleDefinition::GetProperties ( )

Returns the definition's properties.

const wxRichTextProperties& wxRichTextStyleDefinition::GetProperties ( ) const

Returns the definition's properties.

wxRichTextAttr wxRichTextStyleDefinition::GetStyle ( ) const

Returns the attributes associated with this style.

const wxRichTextAttr wxRichTextStyleDefinition::GetStyle ( ) const

Returns the attributes associated with this style.

virtual wxRichTextAttr wxRichTextStyleDefinition::GetStyleMergedWithBase (const wxRichTextStyleSheet * *sheet*) const [virtual]

Returns the style attributes combined with the attributes of the specified base style, if any.

This function works recursively.

void wxRichTextStyleDefinition::SetBaseStyle (const wxString & *name*)

Sets the name of the style that this style is based on.

void wxRichTextStyleDefinition::SetDescription (const wxString & *descr*)

Sets the style description.

void wxRichTextStyleDefinition::SetName (const wxString & *name*)

Sets the name of the style.

void wxRichTextStyleDefinition::SetProperties (const wxRichTextProperties & *props*)

Sets the definition's properties.

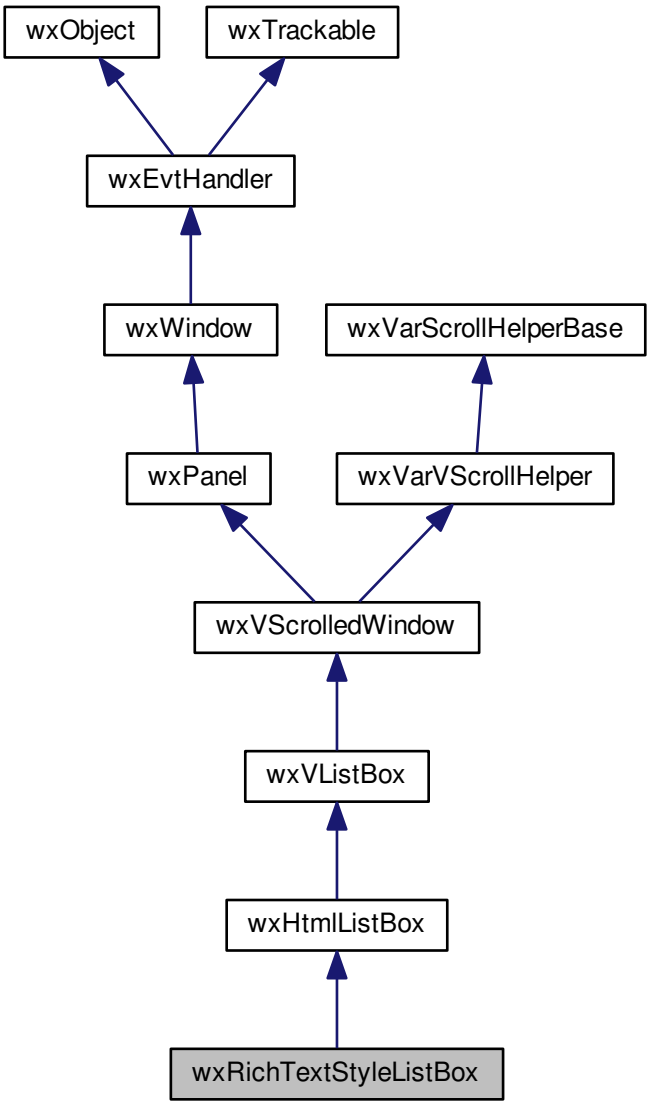
void wxRichTextStyleDefinition::SetStyle (const wxRichTextAttr & *style*)

Sets the attributes for this style.

21.650 wxRichTextStyleListBox Class Reference

#include <wx/richtext/richtextstyles.h>

Inheritance diagram for wxRichTextStyleListBox:



21.650.1 Detailed Description

This is a listbox that can display the styles in a wxRichTextStyleSheet, and apply the selection to an associated wxRichTextCtrl.

See `samples/richtext` for an example of how to use it.

Library: wxRichText

Category: Rich Text

See also

wxRichTextStyleComboCtrl, wxRichTextCtrl Overview

Public Types

- enum wxRichTextStyleType {
  wxRICHTEXT_STYLE_ALL,
  wxRICHTEXT_STYLE_PARAGRAPH,
  wxRICHTEXT_STYLE_CHARACTER,
  wxRICHTEXT_STYLE_LIST,
  wxRICHTEXT_STYLE_BOX }

  Which type of style definition is currently showing?

Public Member Functions

- wxRichTextStyleListBox (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Constructor.
- virtual ~wxRichTextStyleListBox ()

  Destructor.
- void ApplyStyle (int i)

  Applies the ith style to the associated rich text control.
- int ConvertTenthsMMToPixels (wxDC &dc, int units) const

  Converts units in tenths of a millimetre to device units.
- wxString CreateHTML (wxRichTextStyleDefinition *def) const

  Creates a suitable HTML fragment for a definition.
- bool GetApplyOnSelection () const

  If the return value is true, clicking on a style name in the list will immediately apply the style to the associated rich text control.
- wxRichTextCtrl * GetRichTextCtrl () const

  Returns the wxRichTextCtrl associated with this listbox.
- wxRichTextStyleDefinition * GetStyle (size_t i) const

  Gets a style for a listbox index.
- wxRichTextStyleSheet * GetStyleSheet () const

  Returns the style sheet associated with this listbox.
- wxRichTextStyleListBox::wxRichTextStyleType GetStyleType () const

  Returns the type of style to show in the list box.
- void OnLeftDown (wxMouseEvent &event)

  Implements left click behaviour, applying the clicked style to the wxRichTextCtrl.
- void SetApplyOnSelection (bool applyOnSelection)

  If applyOnSelection is true, clicking on a style name in the list will immediately apply the style to the associated rich text control.
- void SetRichTextCtrl (wxRichTextCtrl *ctrl)

  Associates the listbox with a wxRichTextCtrl.
- void SetStyleSheet (wxRichTextStyleSheet *styleSheet)

  Associates the control with a style sheet.
- void SetStyleType (wxRichTextStyleListBox::wxRichTextStyleType styleType)

  Sets the style type to display.
- void UpdateStyles ()

  Updates the list from the associated style sheet.

Protected Member Functions

- virtual wxString OnGetItem (size_t n) const

  Returns the HTML for this item.

Additional Inherited Members

21.650.2 Member Enumeration Documentation

enum wxRichTextStyleListBox::wxRichTextStyleType

Which type of style definition is currently showing?

Enumerator

wxRICHTEXT_STYLE_ALL

wxRICHTEXT_STYLE_PARAGRAPH

wxRICHTEXT_STYLE_CHARACTER

wxRICHTEXT_STYLE_LIST

wxRICHTEXT_STYLE_BOX

21.650.3 Constructor & Destructor Documentation

wxRichTextStyleListBox::wxRichTextStyleListBox (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Constructor.

virtual wxRichTextStyleListBox::~wxRichTextStyleListBox ( ) [virtual]

Destructor.

21.650.4 Member Function Documentation

void wxRichTextStyleListBox::ApplyStyle (int *i*)

Applies the *ith* style to the associated rich text control.

int wxRichTextStyleListBox::ConvertTenthsMMToPixels (wxDC & *dc*, int *units*) const

Converts units in tenths of a millimetre to device units.

wxString wxRichTextStyleListBox::CreateHTML (wxRichTextStyleDefinition * *def*) const

Creates a suitable HTML fragment for a definition.

bool wxRichTextStyleListBox::GetApplyOnSelection ( ) const

If the return value is true, clicking on a style name in the list will immediately apply the style to the associated rich text control.

wxRichTextCtrl* **wxRichTextStyleListBox::GetRichTextCtrl** ( ) **const**

Returns the wxRichTextCtrl associated with this listbox.

wxRichTextStyleDefinition* **wxRichTextStyleListBox::GetStyle** (**size_t** *i*) **const**

Gets a style for a listbox index.

wxRichTextStyleSheet* **wxRichTextStyleListBox::GetStyleSheet** ( ) **const**

Returns the style sheet associated with this listbox.

wxRichTextStyleListBox::wxRichTextStyleType **wxRichTextStyleListBox::GetStyleType** ( ) **const**

Returns the type of style to show in the list box.

virtual wxString **wxRichTextStyleListBox::OnGetItem** (**size_t** *n*) **const** `[protected], [virtual]`

Returns the HTML for this item.

Implements wxHtmlListBox.

void wxRichTextStyleListBox::OnLeftDown (**wxMouseEvent** & *event*)

Implements left click behaviour, applying the clicked style to the wxRichTextCtrl.

void wxRichTextStyleListBox::SetApplyOnSelection (**bool** *applyOnSelection*)

If *applyOnSelection* is true, clicking on a style name in the list will immediately apply the style to the associated rich text control.

void wxRichTextStyleListBox::SetRichTextCtrl (**wxRichTextCtrl** * *ctrl*)

Associates the listbox with a wxRichTextCtrl.

void wxRichTextStyleListBox::SetStyleSheet (**wxRichTextStyleSheet** * *styleSheet*)

Associates the control with a style sheet.

void wxRichTextStyleListBox::SetStyleType (**wxRichTextStyleListBox::wxRichTextStyleType** *styleType*)

Sets the style type to display.

One of

- wxRichTextStyleListBox::wxRICHTEXT_STYLE_ALL,
- wxRichTextStyleListBox::wxRICHTEXT_STYLE_PARAGRAPH,
- wxRichTextStyleListBox::wxRICHTEXT_STYLE_CHARACTER
- wxRichTextStyleListBox::wxRICHTEXT_STYLE_LIST.

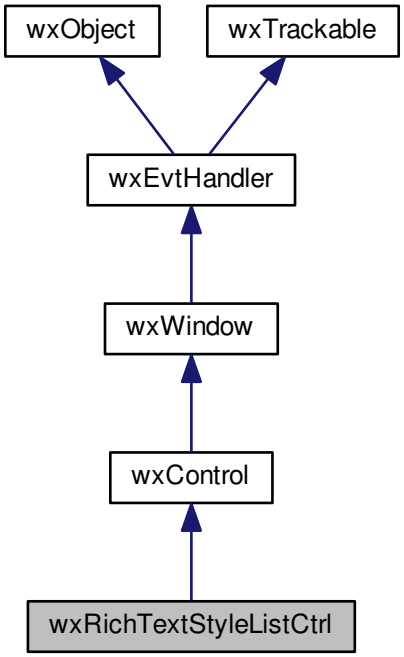
void wxRichTextStyleListBox::UpdateStyles ( )

Updates the list from the associated style sheet.

21.651 wxRichTextStyleListCtrl Class Reference

`#include <wx/richtext/richtextstyles.h>`

Inheritance diagram for wxRichTextStyleListCtrl:



21.651.1 Detailed Description

This class incorporates a wxRichTextStyleListBox and a choice control that allows the user to select the category of style to view.

It is demonstrated in the wxRichTextCtrl sample in `samples/richtext`.

To use wxRichTextStyleListCtrl, add the control to your window hierarchy and call wxRichTextStyleListCtrl::SetStyleType with one of wxRichTextStyleListBox::wxRICHTEXT_STYLE_ALL, wxRichTextStyleListBox::wxRICHTEXT_STYLE_PARAGRAPH, wxRichTextStyleListBox::wxRICHTEXT_STYLE_CHARACTER and wxRichTextStyleListBox::wxRICHTEXT_STYLE_LIST to set the current view.

Associate the control with a style sheet and rich text control with SetStyleSheet and SetRichTextCtrl, so that when a style is double-clicked, it is applied to the selection.

Styles

This class supports the following styles:

- wxRICHTEXTSTYLELIST_HIDE_TYPE_SELECTOR: This style hides the category selection control.

Library: wxRichText

Category: Rich Text

Public Member Functions

- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Creates the windows.
- wxRichTextCtrl * GetRichTextCtrl () const

  Returns the associated rich text control, if any.
- wxChoice * GetStyleChoice () const

  Returns the wxChoice control used for selecting the style category.
- wxRichTextStyleListBox * GetStyleListBox () const

  Returns the wxListBox control used to view the style list.
- wxRichTextStyleSheet * GetStyleSheet () const

  Returns the associated style sheet, if any.
- wxRichTextStyleListBox::wxRichTextStyleType GetStyleType () const

  Returns the type of style to show in the list box.
- void SetRichTextCtrl (wxRichTextCtrl *ctrl)

  Associates the control with a wxRichTextCtrl.
- void SetStyleSheet (wxRichTextStyleSheet *styleSheet)

  Associates the control with a style sheet.
- void SetStyleType (wxRichTextStyleListBox::wxRichTextStyleType styleType)

  Sets the style type to display.
- void UpdateStyles ()

  Updates the style list box.

- wxRichTextStyleListCtrl (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0)

  Constructors.
- wxRichTextStyleListCtrl ()

  Constructors.

Additional Inherited Members

21.651.2 Constructor & Destructor Documentation

wxRichTextStyleListCtrl::wxRichTextStyleListCtrl (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Constructors.

wxRichTextStyleListCtrl::wxRichTextStyleListCtrl ( )

Constructors.

21.651.3 Member Function Documentation

bool wxRichTextStyleListCtrl::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0)

Creates the windows.

wxRichTextCtrl* wxRichTextStyleListCtrl::GetRichTextCtrl ( ) const

Returns the associated rich text control, if any.

wxChoice* wxRichTextStyleListCtrl::GetStyleChoice ( ) const

Returns the wxChoice control used for selecting the style category.

wxRichTextStyleListBox* wxRichTextStyleListCtrl::GetStyleListBox ( ) const

Returns the wxListBox control used to view the style list.

wxRichTextStyleSheet* wxRichTextStyleListCtrl::GetStyleSheet ( ) const

Returns the associated style sheet, if any.

wxRichTextStyleListBox::wxRichTextStyleType wxRichTextStyleListCtrl::GetStyleType ( ) const

Returns the type of style to show in the list box.

void wxRichTextStyleListCtrl::SetRichTextCtrl (wxRichTextCtrl * *ctrl*)

Associates the control with a wxRichTextCtrl.

void wxRichTextStyleListCtrl::SetStyleSheet (wxRichTextStyleSheet * *styleSheet*)

Associates the control with a style sheet.

void wxRichTextStyleListCtrl::SetStyleType (wxRichTextStyleListBox::wxRichTextStyleType *styleType*)

Sets the style type to display.

One of

- wxRichTextStyleListBox::wxRICHTEXT_STYLE_ALL,
- wxRichTextStyleListBox::wxRICHTEXT_STYLE_PARAGRAPH,
- wxRichTextStyleListBox::wxRICHTEXT_STYLE_CHARACTER
- wxRichTextStyleListBox::wxRICHTEXT_STYLE_LIST.

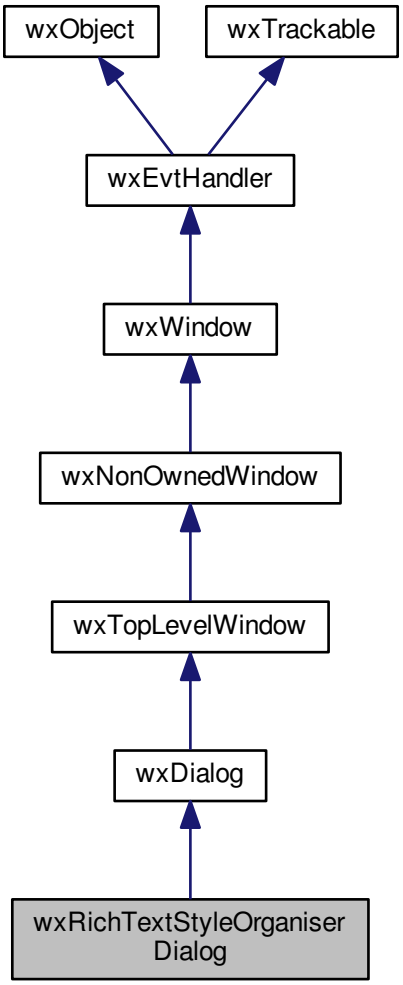
void wxRichTextStyleListCtrl::UpdateStyles ( )

Updates the style list box.

21.652 wxRichTextStyleOrganiserDialog Class Reference

`#include <wx/richtext/richtextstyledlg.h>`

Inheritance diagram for wxRichTextStyleOrganiserDialog:



21.652.1 Detailed Description

This class shows a style sheet and allows the user to edit, add and remove styles.

It can also be used as a style browser, for example if the application is not using a permanent wxRichTextStyleComboCtrl or wxRichTextStyleListCtrl to present styles.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextStyleOrganiserDialog ()

  Default ctor.
- wxRichTextStyleOrganiserDialog (int flags, wxRichTextStyleSheet *sheet, wxRichTextCtrl *ctrl, wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &caption=_("Style Organiser"), const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxSYSTEM_MENU|wxCLOSE_BOX)

  Constructor.
- bool ApplyStyle (wxRichTextCtrl *ctrl=NULL)

  Applies the selected style to selection in the given control or the control passed to the constructor.
- bool Create (int flags, wxRichTextStyleSheet *sheet, wxRichTextCtrl *ctrl, wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &caption=wxGetTranslation("Style Organiser"), const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxSize(400, 300), long style=wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxSYSTEM_MENU|wxCLOSE_BOX)

  Creates the dialog.
- bool GetRestartNumbering () const

  Returns true if the user has opted to restart numbering.
- wxRichTextCtrl * GetRichTextCtrl () const

  Returns the associated rich text control (if any).
- wxString GetSelectedStyle () const

  Returns selected style name.
- wxRichTextStyleDefinition * GetSelectedStyleDefinition () const

  Returns selected style definition.
- wxRichTextStyleSheet * GetStyleSheet () const

  Returns the associated style sheet.
- void SetFlags (int flags)

  Sets the flags used to control the interface presented to the user.
- void SetRestartNumbering (bool restartNumbering)

  Checks or unchecks the restart numbering checkbox.
- void SetRichTextCtrl (wxRichTextCtrl *ctrl)

  Sets the control to be associated with the dialog, for the purposes of applying a style to the selection.
- void SetStyleSheet (wxRichTextStyleSheet *sheet)

  Sets the associated style sheet.
- int GetFlags () const

  Returns the flags used to control the interface presented to the user.

Static Public Member Functions

- static void SetShowToolTips (bool show)

  Determines whether tooltips will be shown.

Additional Inherited Members

21.652.2 Constructor & Destructor Documentation

wxRichTextStyleOrganiserDialog::wxRichTextStyleOrganiserDialog ( )

Default ctor.

wxRichTextStyleOrganiserDialog::wxRichTextStyleOrganiserDialog (int *flags,* wxRichTextStyleSheet * *sheet,* wxRichTextCtrl * *ctrl,* wxWindow * *parent,* wxWindowID *id* = wxID_ANY, const wxString & *caption* = _("Style Organiser"), const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxSYSTEM_MENU|wxCLOSE_BOX)

Constructor.

To create a dialog, pass a bitlist of *flags* (see below), a style sheet, a text control to apply a selected style to (or NULL), followed by the usual window parameters.

To specify the operations available to the user, pass a combination of these values to *flags:*

- **wxRICHTEXT_ORGANISER_DELETE_STYLES:** Provides a button for deleting styles.
- **wxRICHTEXT_ORGANISER_CREATE_STYLES:** Provides buttons for creating styles.
- **wxRICHTEXT_ORGANISER_APPLY_STYLES:** Provides a button for applying the currently selected style to the selection.
- **wxRICHTEXT_ORGANISER_EDIT_STYLES:** Provides a button for editing styles.
- **wxRICHTEXT_ORGANISER_RENAME_STYLES:** Provides a button for renaming styles.
- **wxRICHTEXT_ORGANISER_OK_CANCEL:** Provides OK and Cancel buttons.
- **wxRICHTEXT_ORGANISER_RENUMBER:** Provides a checkbox for specifying that the selection should be renumbered.

The following flags determine what will be displayed in the style list:

- **wxRICHTEXT_ORGANISER_SHOW_CHARACTER:** Displays character styles only.
- **wxRICHTEXT_ORGANISER_SHOW_PARAGRAPH:** Displays paragraph styles only.
- **wxRICHTEXT_ORGANISER_SHOW_LIST:** Displays list styles only.
- **wxRICHTEXT_ORGANISER_SHOW_ALL:** Displays all styles.

The following symbols define commonly-used combinations of flags:

- **wxRICHTEXT_ORGANISER_ORGANISE:** Enable all style editing operations so the dialog behaves as a style organiser.
- **wxRICHTEXT_ORGANISER_BROWSE:** Show a list of all styles and their previews, but only allow application of a style or cancellation of the dialog. This makes the dialog behave as a style browser.
- **wxRICHTEXT_ORGANISER_BROWSE_NUMBERING:** Enables only list style browsing, plus a control to specify renumbering. This allows the dialog to be used for applying list styles to the selection.

21.652.3 Member Function Documentation

bool wxRichTextStyleOrganiserDialog::ApplyStyle (wxRichTextCtrl * *ctrl* = NULL)

Applies the selected style to selection in the given control or the control passed to the constructor.

bool wxRichTextStyleOrganiserDialog::Create (int *flags,* wxRichTextStyleSheet * *sheet,* wxRichTextCtrl * *ctrl,* wxWindow * *parent,* wxWindowID *id* = wxID_ANY, const wxString & *caption* = wxGetTranslation("Style Organiser"), const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxSize(400, 300), long *style* = wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxSYSTEM_MEN↩ U|wxCLOSE_BOX)

Creates the dialog.

See the ctor.

int wxRichTextStyleOrganiserDialog::GetFlags ( ) const

Returns the flags used to control the interface presented to the user.

bool wxRichTextStyleOrganiserDialog::GetRestartNumbering ( ) const

Returns true if the user has opted to restart numbering.

wxRichTextCtrl* wxRichTextStyleOrganiserDialog::GetRichTextCtrl ( ) const

Returns the associated rich text control (if any).

wxString wxRichTextStyleOrganiserDialog::GetSelectedStyle ( ) const

Returns selected style name.

wxRichTextStyleDefinition* wxRichTextStyleOrganiserDialog::GetSelectedStyleDefinition ( ) const

Returns selected style definition.

wxRichTextStyleSheet* wxRichTextStyleOrganiserDialog::GetStyleSheet ( ) const

Returns the associated style sheet.

void wxRichTextStyleOrganiserDialog::SetFlags (int *flags*)

Sets the flags used to control the interface presented to the user.

void wxRichTextStyleOrganiserDialog::SetRestartNumbering (bool *restartNumbering*)

Checks or unchecks the restart numbering checkbox.

void wxRichTextStyleOrganiserDialog::SetRichTextCtrl (wxRichTextCtrl * *ctrl*)

Sets the control to be associated with the dialog, for the purposes of applying a style to the selection.

static void wxRichTextStyleOrganiserDialog::SetShowToolTips (bool *show*) `[static]`

Determines whether tooltips will be shown.

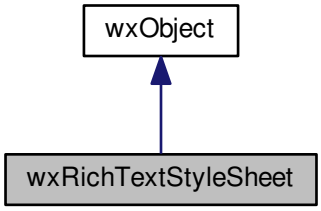
void wxRichTextStyleOrganiserDialog::SetStyleSheet (wxRichTextStyleSheet * *sheet*)

Sets the associated style sheet.

21.653 wxRichTextStyleSheet Class Reference

`#include <wx/richtext/richtextstyles.h>`

Inheritance diagram for wxRichTextStyleSheet:



21.653.1 Detailed Description

A style sheet contains named paragraph and character styles that make it easy for a user to apply combinations of attributes to a wxRichTextCtrl.

You can use a wxRichTextStyleListBox in your user interface to show available styles to the user, and allow application of styles to the control.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextStyleSheet ()

  Constructor.
- virtual ~wxRichTextStyleSheet ()

  Destructor.
- bool AddCharacterStyle (wxRichTextCharacterStyleDefinition *def)

  Adds a definition to the character style list.
- bool AddListStyle (wxRichTextListStyleDefinition *def)

  Adds a definition to the list style list.
- bool AddParagraphStyle (wxRichTextParagraphStyleDefinition *def)

  Adds a definition to the paragraph style list.
- bool AddStyle (wxRichTextStyleDefinition *def)

  Adds a definition to the appropriate style list.
- void DeleteStyles ()

  Deletes all styles.
- wxRichTextCharacterStyleDefinition * FindCharacterStyle (const wxString &name, bool recurse=true) const

  Finds a character definition by name.
- wxRichTextListStyleDefinition * FindListStyle (const wxString &name, bool recurse=true) const

  Finds a list definition by name.
- wxRichTextParagraphStyleDefinition * FindParagraphStyle (const wxString &name, bool recurse=true) const

  Finds a paragraph definition by name.
- wxRichTextStyleDefinition * FindStyle (const wxString &name) const

Finds a style definition by name.

- wxRichTextCharacterStyleDefinition * GetCharacterStyle (size_t n) const

  Returns the nth character style.

- size_t GetCharacterStyleCount () const

  Returns the number of character styles.

- const wxString & GetDescription () const

  Returns the style sheet's description.

- wxRichTextListStyleDefinition * GetListStyle (size_t n) const

  Returns the nth list style.

- size_t GetListStyleCount () const

  Returns the number of list styles.

- const wxString & GetName () const

  Returns the style sheet's name.

- wxRichTextParagraphStyleDefinition * GetParagraphStyle (size_t n) const

  Returns the nth paragraph style.

- size_t GetParagraphStyleCount () const

  Returns the number of paragraph styles.

- bool RemoveCharacterStyle (wxRichTextStyleDefinition *def, bool deleteStyle=false)

  Removes a character style.

- bool RemoveListStyle (wxRichTextStyleDefinition *def, bool deleteStyle=false)

  Removes a list style.

- bool RemoveParagraphStyle (wxRichTextStyleDefinition *def, bool deleteStyle=false)

  Removes a paragraph style.

- bool RemoveStyle (wxRichTextStyleDefinition *def, bool deleteStyle=false)

  Removes a style.

- void SetDescription (const wxString &descr)

  Sets the style sheet's description.

- void SetName (const wxString &name)

  Sets the style sheet's name.

- wxRichTextProperties & GetProperties ()

  Returns the sheet's properties.

- const wxRichTextProperties & GetProperties () const

  Returns the sheet's properties.

- void SetProperties (const wxRichTextProperties &props)

  Sets the sheet's properties.

Additional Inherited Members

21.653.2 Constructor & Destructor Documentation

wxRichTextStyleSheet::wxRichTextStyleSheet ( )

Constructor.

virtual wxRichTextStyleSheet::~wxRichTextStyleSheet ( ) [virtual]

Destructor.

21.653.3 Member Function Documentation

bool wxRichTextStyleSheet::AddCharacterStyle (wxRichTextCharacterStyleDefinition * *def*)

Adds a definition to the character style list.

bool wxRichTextStyleSheet::AddListStyle (wxRichTextListStyleDefinition * *def*)

Adds a definition to the list style list.

bool wxRichTextStyleSheet::AddParagraphStyle (wxRichTextParagraphStyleDefinition * *def*)

Adds a definition to the paragraph style list.

bool wxRichTextStyleSheet::AddStyle (wxRichTextStyleDefinition * *def*)

Adds a definition to the appropriate style list.

void wxRichTextStyleSheet::DeleteStyles ( )

Deletes all styles.

wxRichTextCharacterStyleDefinition* wxRichTextStyleSheet::FindCharacterStyle (const wxString & *name,* bool *recurse* = `true`) const

Finds a character definition by name.

wxRichTextListStyleDefinition* wxRichTextStyleSheet::FindListStyle (const wxString & *name,* bool *recurse* = `true`) const

Finds a list definition by name.

wxRichTextParagraphStyleDefinition* wxRichTextStyleSheet::FindParagraphStyle (const wxString & *name,* bool *recurse* = `true`) const

Finds a paragraph definition by name.

wxRichTextStyleDefinition* wxRichTextStyleSheet::FindStyle (const wxString & *name*) const

Finds a style definition by name.

wxRichTextCharacterStyleDefinition* wxRichTextStyleSheet::GetCharacterStyle (size_t *n*) const

Returns the *nth* character style.

size_t wxRichTextStyleSheet::GetCharacterStyleCount ( ) const

Returns the number of character styles.

const wxString& wxRichTextStyleSheet::GetDescription ( ) const

Returns the style sheet's description.

wxRichTextListStyleDefinition* wxRichTextStyleSheet::GetListStyle (size_t *n*) const

Returns the *nth* list style.

size_t wxRichTextStyleSheet::GetListStyleCount ( ) const

Returns the number of list styles.

const wxString& wxRichTextStyleSheet::GetName ( ) const

Returns the style sheet's name.

wxRichTextParagraphStyleDefinition* wxRichTextStyleSheet::GetParagraphStyle (size_t *n*) const

Returns the *nth* paragraph style.

size_t wxRichTextStyleSheet::GetParagraphStyleCount ( ) const

Returns the number of paragraph styles.

wxRichTextProperties& wxRichTextStyleSheet::GetProperties ( )

Returns the sheet's properties.

const wxRichTextProperties& wxRichTextStyleSheet::GetProperties ( ) const

Returns the sheet's properties.

bool wxRichTextStyleSheet::RemoveCharacterStyle (wxRichTextStyleDefinition * *def,* bool *deleteStyle* = `false`)

Removes a character style.

bool wxRichTextStyleSheet::RemoveListStyle (wxRichTextStyleDefinition * *def,* bool *deleteStyle* = `false`)

Removes a list style.

bool wxRichTextStyleSheet::RemoveParagraphStyle (wxRichTextStyleDefinition * *def,* bool *deleteStyle* = `false`)

Removes a paragraph style.

bool wxRichTextStyleSheet::RemoveStyle (wxRichTextStyleDefinition * *def,* bool *deleteStyle* = `false`)

Removes a style.

void wxRichTextStyleSheet::SetDescription (const wxString & *descr*)

Sets the style sheet's description.

void wxRichTextStyleSheet::SetName (const wxString & *name*)

Sets the style sheet's name.

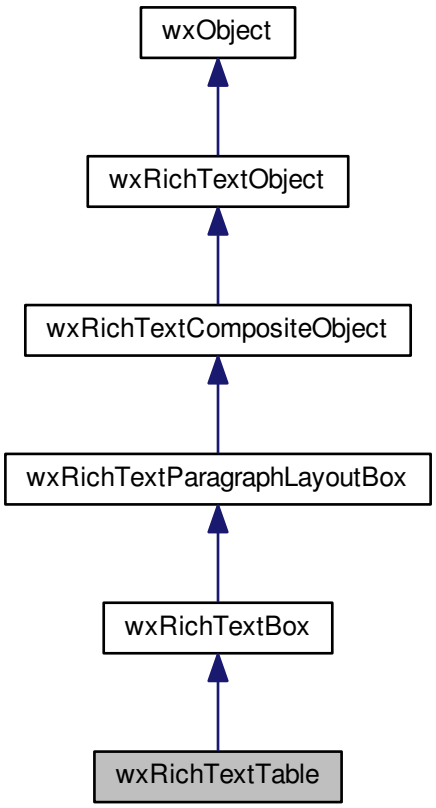
void wxRichTextStyleSheet::SetProperties (const wxRichTextProperties & *props*)

Sets the sheet's properties.

21.654 wxRichTextTable Class Reference

`#include <wx/richtext/richtextbuffer.h>`

Inheritance diagram for wxRichTextTable:



21.654.1 Detailed Description

wxRichTextTable represents a table with arbitrary columns and rows.

Public Member Functions

- wxRichTextTable (wxRichTextObject *parent=NULL)

  Default constructor; optionally pass the parent object.
- wxRichTextTable (const wxRichTextTable &obj)

  Copy constructor.
- virtual bool Draw (wxDC &dc, wxRichTextDrawingContext &context, const wxRichTextRange &range, const wxRichTextSelection &selection, const wxRect &rect, int descent, int style)

  Draw the item, within the given range.
- virtual int HitTest (wxDC &dc, wxRichTextDrawingContext &context, const wxPoint &pt, long &textPosition, wxRichTextObject **obj, wxRichTextObject **contextObj, int flags=0)

  Hit-testing: returns a flag indicating hit test details, plus information about position.
- virtual wxString GetXMLNodeName () const

  Returns the XML node name of this object.
- virtual bool Layout (wxDC &dc, wxRichTextDrawingContext &context, const wxRect &rect, const wxRect &parentRect, int style)

  Lay the item out at the specified position with the given size constraint.
- virtual bool GetRangeSize (const wxRichTextRange &range, wxSize &size, int &descent, wxDC &dc, wxRichTextDrawingContext &context, int flags, const wxPoint &position=wxPoint(0, 0), const wxSize &parentSize=wxDefaultSize, wxArrayInt *partialExtents=NULL) const

  Returns the object size for the given range.
- virtual bool DeleteRange (const wxRichTextRange &range)

  Deletes the given range.
- virtual wxString GetTextForRange (const wxRichTextRange &range) const

  Returns any text in this object for the given range.
- virtual bool ImportFromXML (wxRichTextBuffer *buffer, wxXmlNode *node, wxRichTextXMLHandler *handler, bool *recurse)

  Imports this object from XML.
- virtual bool FindPosition (wxDC &dc, wxRichTextDrawingContext &context, long index, wxPoint &pt, int *height, bool forceLineStart)

  Finds the absolute position and row height for the given character position.
- virtual void CalculateRange (long start, long &end)

  Calculates the range of the object.
- virtual bool HandlesChildSelections () const

  Returns true if this object can handle the selections of its children, fOr example a table.
- virtual wxRichTextSelection GetSelection (long start, long end) const

  Returns a selection object specifying the selections between start and end character positions.
- virtual bool CanEditProperties () const

  Returns true if we can edit the object's properties via a GUI.
- virtual bool EditProperties (wxWindow *parent, wxRichTextBuffer *buffer)

  Edits the object's properties via a GUI.
- virtual wxString GetPropertiesMenuLabel () const

  Returns the label to be used for the properties context menu item.
- virtual bool AcceptsFocus () const

  Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.
- const wxRichTextObjectPtrArrayArray & GetCells () const

  Returns the cells array.
- wxRichTextObjectPtrArrayArray & GetCells ()

  Returns the cells array.
- int GetRowCount () const

  Returns the row count.

- int GetColumnCount () const

  Returns the column count.
- virtual wxRichTextCell * GetCell (int row, int col) const

  Returns the cell at the given row/column position.
- virtual wxRichTextCell * GetCell (long pos) const

  Returns the cell at the given character position (in the range of the table).
- virtual bool GetCellRowColumnPosition (long pos, int &row, int &col) const

  Returns the row/column for a given character position.
- virtual wxPosition GetFocusedCell () const

  Returns the coordinates of the cell with keyboard focus, or (-1,-1) if none.
- virtual void ClearTable ()

  Clears the table.
- virtual bool CreateTable (int rows, int cols)

  Creates a table of the given dimensions.
- virtual bool SetCellStyle (const wxRichTextSelection &selection, const wxRichTextAttr &style, int flags=wxRICHTEXT_SETSTYLE_WITH_UNDO)

  Sets the attributes for the cells specified by the selection.
- virtual bool DeleteRows (int startRow, int noRows=1)

  Deletes rows from the given row position.
- virtual bool DeleteColumns (int startCol, int noCols=1)

  Deletes columns from the given column position.
- virtual bool AddRows (int startRow, int noRows=1, const wxRichTextAttr &attr=wxRichTextAttr())

  Adds rows from the given row position.
- virtual bool AddColumns (int startCol, int noCols=1, const wxRichTextAttr &attr=wxRichTextAttr())

  Adds columns from the given column position.
- virtual wxRichTextObject * Clone () const

  Clones the object.
- void Copy (const wxRichTextTable &obj)

Protected Attributes

- int m_rowCount
- int m_colCount
- wxRichTextObjectPtrArrayArray m_cells

Additional Inherited Members

21.654.2 Constructor & Destructor Documentation

wxRichTextTable::wxRichTextTable (wxRichTextObject * *parent* = NULL)

Default constructor; optionally pass the parent object.

wxRichTextTable::wxRichTextTable (const wxRichTextTable & *obj*) [inline]

Copy constructor.

21.654.3 Member Function Documentation

virtual bool wxRichTextTable::AcceptsFocus ( ) const `[inline], [virtual]`

Returns true if objects of this class can accept the focus, i.e. a call to SetFocusObject is possible.

For example, containers supporting text, such as a text box object, can accept the focus, but a table can't (set the focus to individual cells instead).

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextTable::AddColumns (int *startCol,* int *noCols* = `1`, const wxRichTextAttr & *attr* = wxRichTextAttr `()`) `[virtual]`

Adds columns from the given column position.

virtual bool wxRichTextTable::AddRows (int *startRow,* int *noRows* = `1`, const wxRichTextAttr & *attr* = wxRichTextAttr `()`) `[virtual]`

Adds rows from the given row position.

virtual void wxRichTextTable::CalculateRange (long *start,* long & *end*) `[virtual]`

Calculates the range of the object.

By default, guess that the object is 1 unit long.

Reimplemented from wxRichTextCompositeObject.

virtual bool wxRichTextTable::CanEditProperties ( ) const `[inline], [virtual]`

Returns true if we can edit the object's properties via a GUI.

Reimplemented from wxRichTextBox.

virtual void wxRichTextTable::ClearTable ( ) `[virtual]`

Clears the table.

virtual wxRichTextObject* wxRichTextTable::Clone ( ) const `[inline], [virtual]`

Clones the object.

Reimplemented from wxRichTextBox.

void wxRichTextTable::Copy (const wxRichTextTable & *obj*)

virtual bool wxRichTextTable::CreateTable (int *rows,* int *cols*) `[virtual]`

Creates a table of the given dimensions.

virtual bool wxRichTextTable::DeleteColumns (int *startCol,* int *noCols* = `1`) `[virtual]`

Deletes columns from the given column position.

virtual bool wxRichTextTable::DeleteRange (const wxRichTextRange & *range*) `[virtual]`

Deletes the given range.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextTable::DeleteRows (int *startRow,* int *noRows* = 1) `[virtual]`

Deletes rows from the given row position.

virtual bool wxRichTextTable::Draw (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRichTextRange & *range,* const wxRichTextSelection & *selection,* const wxRect & *rect,* int *descent,* int *style*) `[virtual]`

Draw the item, within the given range.

Some objects may ignore the range (for example paragraphs) while others must obey it (lines, to implement wrapping)

Reimplemented from wxRichTextBox.

virtual bool wxRichTextTable::EditProperties (wxWindow * *parent,* wxRichTextBuffer * *buffer*) `[virtual]`

Edits the object's properties via a GUI.

Reimplemented from wxRichTextBox.

virtual bool wxRichTextTable::FindPosition (wxDC & *dc,* wxRichTextDrawingContext & *context,* long *index,* wxPoint & *pt,* int * *height,* bool *forceLineStart*) `[virtual]`

Finds the absolute position and row height for the given character position.

Reimplemented from wxRichTextCompositeObject.

virtual wxRichTextCell* wxRichTextTable::GetCell (int *row,* int *col*) const `[virtual]`

Returns the cell at the given row/column position.

virtual wxRichTextCell* wxRichTextTable::GetCell (long *pos*) const `[virtual]`

Returns the cell at the given character position (in the range of the table).

virtual bool wxRichTextTable::GetCellRowColumnPosition (long *pos,* int & *row,* int & *col*) const `[virtual]`

Returns the row/column for a given character position.

const wxRichTextObjectPtrArrayArray& wxRichTextTable::GetCells ( ) const `[inline]`

Returns the cells array.

wxRichTextObjectPtrArrayArray& wxRichTextTable::GetCells ( ) `[inline]`

Returns the cells array.

int wxRichTextTable::GetColumnCount ( ) const `[inline]`

Returns the column count.

virtual wxPosition wxRichTextTable::GetFocusedCell ( ) const `[virtual]`

Returns the coordinates of the cell with keyboard focus, or (-1,-1) if none.

virtual wxString wxRichTextTable::GetPropertiesMenuLabel ( ) const `[inline], [virtual]`

Returns the label to be used for the properties context menu item.

Reimplemented from wxRichTextBox.

virtual bool wxRichTextTable::GetRangeSize (const wxRichTextRange & *range*, wxSize & *size*, int & *descent*, wxDC & *dc*, wxRichTextDrawingContext & *context*, int *flags*, const wxPoint & *position* = wxPoint(0, 0), const wxSize & *parentSize* = wxDefaultSize, wxArrayInt * *partialExtents* = NULL) const `[virtual]`

Returns the object size for the given range.

Returns false if the range is invalid for this object.

Reimplemented from wxRichTextParagraphLayoutBox.

int wxRichTextTable::GetRowCount ( ) const `[inline]`

Returns the row count.

virtual wxRichTextSelection wxRichTextTable::GetSelection (long *start*, long *end*) const `[virtual]`

Returns a selection object specifying the selections between start and end character positions.

For example, a table would deduce what cells (of range length 1) are selected when dragging across the table.

Reimplemented from wxRichTextObject.

virtual wxString wxRichTextTable::GetTextForRange (const wxRichTextRange & *range*) const `[virtual]`

Returns any text in this object for the given range.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual wxString wxRichTextTable::GetXMLNodeName ( ) const `[inline], [virtual]`

Returns the XML node name of this object.

This must be overridden for wxXmlNode-base XML export to work.

Reimplemented from wxRichTextBox.

virtual bool wxRichTextTable::HandlesChildSelections ( ) const `[inline], [virtual]`

Returns true if this object can handle the selections of its children, fOr example a table.

Required for composite selection handling to work.

Reimplemented from wxRichTextObject.

virtual int wxRichTextTable::HitTest (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxPoint & *pt,* long & *textPosition,* wxRichTextObject ** *obj,* wxRichTextObject ** *contextObj,* int *flags* = 0) [virtual]

Hit-testing: returns a flag indicating hit test details, plus information about position.

contextObj is returned to specify what object position is relevant to, since otherwise there's an ambiguity. @ obj might not be a child of *contextObj*, since we may be referring to the container itself if we have no hit on a child - for example if we click outside an object.

The function puts the position in *textPosition* if one is found. *pt* is in logical units (a zero y position is at the beginning of the buffer).

Returns

One of the wxRichTextHitTestFlags values.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextTable::ImportFromXML (wxRichTextBuffer * *buffer,* wxXmlNode * *node,* wxRichTextXMLHandler * *handler,* bool * *recurse*) [virtual]

Imports this object from XML.

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextTable::Layout (wxDC & *dc,* wxRichTextDrawingContext & *context,* const wxRect & *rect,* const wxRect & *parentRect,* int *style*) [virtual]

Lay the item out at the specified position with the given size constraint.

Layout must set the cached size. *rect* is the available space for the object, and *parentRect* is the container that is used to determine a relative size or position (for example if a text box must be 50% of the parent text box).

Reimplemented from wxRichTextParagraphLayoutBox.

virtual bool wxRichTextTable::SetCellStyle (const wxRichTextSelection & *selection,* const wxRichTextAttr & *style,* int *flags* = wxRICHTEXT_SETSTYLE_WITH_UNDO) [virtual]

Sets the attributes for the cells specified by the selection.

21.654.4 Member Data Documentation

wxRichTextObjectPtrArrayArray wxRichTextTable::m_cells [protected]

int wxRichTextTable::m_colCount [protected]

int wxRichTextTable::m_rowCount [protected]

21.655 wxRichTextTableBlock Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.655.1 Detailed Description

Stores the coordinates for a block of cells.

Public Member Functions

- wxRichTextTableBlock ()
- wxRichTextTableBlock (int colStart, int colEnd, int rowStart, int rowEnd)
- wxRichTextTableBlock (const wxRichTextTableBlock &block)
- void Init ()
- void Copy (const wxRichTextTableBlock &block)
- void operator= (const wxRichTextTableBlock &block)
- bool operator== (const wxRichTextTableBlock &block)
- bool ComputeBlockForSelection (wxRichTextTable *table, wxRichTextCtrl *ctrl, bool requireCell↩ Selection=true)

  Computes the block given a table (perhaps about to be edited) and a rich text control that may have a selection.
- bool IsWholeTable (wxRichTextTable *table) const

  Does this block represent the whole table?
- int & ColStart ()
- int ColStart () const
- int & ColEnd ()
- int ColEnd () const
- int & RowStart ()
- int RowStart () const
- int & RowEnd ()
- int RowEnd () const

Static Public Member Functions

- static wxRichTextCell * GetFocusedCell (wxRichTextCtrl *ctrl)

  Returns the cell focused in the table, if any.

Public Attributes

- int m_colStart
- int m_colEnd
- int m_rowStart
- int m_rowEnd

21.655.2 Constructor & Destructor Documentation

wxRichTextTableBlock::wxRichTextTableBlock ( ) [inline]

wxRichTextTableBlock::wxRichTextTableBlock (int *colStart*, int *colEnd*, int *rowStart*, int *rowEnd*) [inline]

wxRichTextTableBlock::wxRichTextTableBlock (const wxRichTextTableBlock & *block*) [inline]

21.655.3 Member Function Documentation

int& wxRichTextTableBlock::ColEnd ( ) [inline]

int wxRichTextTableBlock::ColEnd ( ) const [inline]

int& wxRichTextTableBlock::ColStart ( ) [inline]

int wxRichTextTableBlock::ColStart ( ) const [inline]

bool wxRichTextTableBlock::ComputeBlockForSelection (wxRichTextTable * *table,* wxRichTextCtrl * *ctrl,* bool *requireCellSelection* = `true`)

Computes the block given a table (perhaps about to be edited) and a rich text control that may have a selection.

If no selection, the whole table is used. If just the whole content of one cell is selected, this cell only is used. If the cell contents is not selected and requireCellSelection is false, the focused cell will count as a selected cell.

void wxRichTextTableBlock::Copy (const wxRichTextTableBlock & *block*) `[inline]`

static wxRichTextCell* wxRichTextTableBlock::GetFocusedCell (wxRichTextCtrl * *ctrl*) `[static]`

Returns the cell focused in the table, if any.

void wxRichTextTableBlock::Init ( ) `[inline]`

bool wxRichTextTableBlock::IsWholeTable (wxRichTextTable * *table*) const

Does this block represent the whole table?

void wxRichTextTableBlock::operator= (const wxRichTextTableBlock & *block*) `[inline]`

bool wxRichTextTableBlock::operator== (const wxRichTextTableBlock & *block*) `[inline]`

int& wxRichTextTableBlock::RowEnd ( ) `[inline]`

int wxRichTextTableBlock::RowEnd ( ) const `[inline]`

int& wxRichTextTableBlock::RowStart ( ) `[inline]`

int wxRichTextTableBlock::RowStart ( ) const `[inline]`

21.655.4 Member Data Documentation

int wxRichTextTableBlock::m_colEnd

int wxRichTextTableBlock::m_colStart

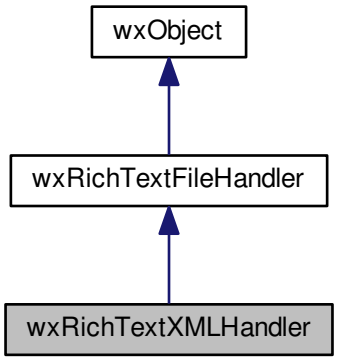
int wxRichTextTableBlock::m_rowEnd

int wxRichTextTableBlock::m_rowStart

21.656 wxRichTextXMLHandler Class Reference

`#include <wx/richtext/richtextxml.h>`

Inheritance diagram for wxRichTextXMLHandler:



21.656.1 Detailed Description

A handler for loading and saving content in an XML format specific to wxRichTextBuffer.

You can either add the handler to the buffer and load and save through the buffer or control API, or you can create an instance of the handler on the stack and call its functions directly.

21.656.2 Handler flags

The following flags can be used with this handler, via the handler's SetFlags() function or the buffer or control's SetHandlerFlags() function:

- wxRICHTEXT_HANDLER_INCLUDE_STYLESHEET Include the style sheet in loading and saving operations.

Library: wxRichText

Category: Rich Text

Public Member Functions

- wxRichTextXMLHandler (const wxString &name="XML", const wxString &ext="xml", int type=wxRICHTEXT_TYPE_XML)

  Constructor.
- virtual bool CanLoad () const

  Returns true.
- virtual bool CanSave () const

  Returns true.
- bool ExportXML (wxOutputStream &stream, wxRichTextObject &obj, int level)

  Recursively exports an object to the stream.
- bool ImportXML (wxRichTextBuffer *buffer, wxRichTextObject *obj, wxXmlNode *node)

  Recursively imports an object.

Static Public Member Functions

- static void RegisterNodeName (const wxString &nodeName, const wxString &className)

  Call with XML node name, C++ class name so that wxRTC can read in the node.
- static void ClearNodeToClassMap ()

  Cleans up the mapping between node name and C++ class.

Protected Member Functions

- virtual bool DoLoadFile (wxRichTextBuffer *buffer, wxInputStream &stream)

  Loads buffer context from the given stream.
- virtual bool DoSaveFile (wxRichTextBuffer *buffer, wxOutputStream &stream)

  Saves buffer context to the given stream.

Additional Inherited Members

21.656.3 Constructor & Destructor Documentation

wxRichTextXMLHandler::wxRichTextXMLHandler (const wxString & *name* = "XML", const wxString & *ext* = "xml", int *type* = wxRICHTEXT_TYPE_XML)

Constructor.

21.656.4 Member Function Documentation

virtual bool wxRichTextXMLHandler::CanLoad ( ) const `[virtual]`

Returns true.

Reimplemented from wxRichTextFileHandler.

virtual bool wxRichTextXMLHandler::CanSave ( ) const `[virtual]`

Returns true.

Reimplemented from wxRichTextFileHandler.

static void wxRichTextXMLHandler::ClearNodeToClassMap ( ) `[inline], [static]`

Cleans up the mapping between node name and C++ class.

virtual bool wxRichTextXMLHandler::DoLoadFile (wxRichTextBuffer * *buffer*, wxInputStream & *stream*) `[protected], [virtual]`

Loads buffer context from the given stream.

Implements wxRichTextFileHandler.

virtual bool wxRichTextXMLHandler::DoSaveFile (wxRichTextBuffer * *buffer*, wxOutputStream & *stream*) `[protected], [virtual]`

Saves buffer context to the given stream.

Implements wxRichTextFileHandler.

bool wxRichTextXMLHandler::ExportXML (wxOutputStream & *stream,* wxRichTextObject & *obj,* int *level*)

Recursively exports an object to the stream.

bool wxRichTextXMLHandler::ImportXML (wxRichTextBuffer * *buffer,* wxRichTextObject * *obj,* wxXmlNode * *node*)

Recursively imports an object.

static void wxRichTextXMLHandler::RegisterNodeName (const wxString & *nodeName,* const wxString & *className*)
`[inline], [static]`

Call with XML node name, C++ class name so that wxRTC can read in the node.

If you add a custom object, call this.

21.657 wxRichToolTip Class Reference

`#include <wx/richtooltip.h>`

21.657.1 Detailed Description

Allows to show a tool tip with more customizations than wxToolTip.

Using this class is very simple, to give a standard warning for a password text control if the password was entered correctly you could simply do:

```
wxTextCtrl* password = new wxTextCtrl(..., wxTE_PASSWORD);
...
wxRichToolTip tip("Caps Lock is on",
                  "You might have made an error in your password\n"
                  "entry because Caps Lock is turned on.\n"
                  "\n"
                  "Press Caps Lock key to turn it off.");
tip.SetIcon(wxICON_WARNING);
tip.ShowFor(password);
```

Currently this class has generic implementation that can be used with any window and implements all the functionality but doesn't exactly match the appearance of the native tooltips (even though it makes some efforts to use the style most appropriate for the current platform) and a native MSW version which can be only used with text controls and doesn't provide as much in the way of customization. Because of this, it's inadvisable to customize the tooltips unnecessarily as doing this turns off auto-detection of the native style in the generic version and may prevent the native MSW version from being used at all.

Notice that this class is not derived from wxWindow and hence doesn't represent a window, even if its ShowFor() method does create one internally to show the tooltip.

The images below show some examples of rich tooltips on different platforms, with various customizations applied.

Library: wxAdvanced

Category: Miscellaneous Windows

Since

2.9.3

Public Member Functions

- wxRichToolTip (const wxString &title, const wxString &message)

  Constructor must specify the tooltip title and main message.
- void SetBackgroundColour (const wxColour &col, const wxColour &colEnd=wxColour())

  Set the background colour.
- void SetTimeout (unsigned millisecondsTimeout, unsigned millisecondsDelay=0)

  Set timeout after which the tooltip should disappear and optionally set a delay before the tooltip is shown, in milliseconds.
- void SetTipKind (wxTipKind tipKind)

  Choose the tip kind, possibly none.
- void SetTitleFont (const wxFont &font)

  Set the title text font.
- void ShowFor (wxWindow *win, const wxRect *rect=NULL)

  Show the tooltip for the given window and optionally specify where to show the tooltip.
- ~wxRichToolTip ()

  Destructor.

- void SetIcon (int icon=wxICON_INFORMATION)

  Set the small icon to show.
- void SetIcon (const wxIcon &icon)

  Set the small icon to show.

21.657.2 Constructor & Destructor Documentation

wxRichToolTip::wxRichToolTip (const wxString & *title,* const wxString & *message*)

Constructor must specify the tooltip title and main message.

The main message can contain embedded new lines. Both the title and message must be non-empty.

Additional attributes can be set later.

wxRichToolTip::~wxRichToolTip ( )

Destructor.

Notice that destroying this object does not hide the tooltip if it's currently shown, it will be hidden and destroyed when the user dismisses it or the timeout expires.

The destructor is non-virtual as this class is not supposed to be derived from.

21.657.3 Member Function Documentation

void wxRichToolTip::SetBackgroundColour (const wxColour & *col,* const wxColour & *colEnd* = wxColour ( ))

Set the background colour.

If two colours are specified, the background is drawn using a gradient from top to bottom, otherwise a single solid colour is used.

By default the colour or colours most appropriate for the current platform are used. If a colour is explicitly set, native MSW version won't be used as it doesn't support setting the colour.

void wxRichToolTip::SetIcon (int *icon* = wxICON_INFORMATION)

Set the small icon to show.

The icon can be either one of the standard information/warning/error ones, i.e. wxICON_INFORMATION, wxICO↩N_WARNING or wxICON_ERROR respectively (the question icon doesn't make sense for a tooltip so wxICON_↩QUESTION can't be used here) or a custom icon. The latter is unsupported by the native MSW implementation of this class so the use of a standard icon is preferred.

void wxRichToolTip::SetIcon (const wxIcon & *icon*)

Set the small icon to show.

The icon can be either one of the standard information/warning/error ones, i.e. wxICON_INFORMATION, wxICO↩N_WARNING or wxICON_ERROR respectively (the question icon doesn't make sense for a tooltip so wxICON_↩QUESTION can't be used here) or a custom icon. The latter is unsupported by the native MSW implementation of this class so the use of a standard icon is preferred.

void wxRichToolTip::SetTimeout (unsigned *millisecondsTimeout,* unsigned *millisecondsDelay* = 0)

Set timeout after which the tooltip should disappear and optionally set a delay before the tooltip is shown, in milliseconds.

By default the tooltip is shown immediately and hidden after a system-dependent interval of time elapses. This method can be used to change this or also disable hiding the tooltip automatically entirely by passing 0 in this parameter (but doing this will prevent the native MSW version from being used).

Notice that the tooltip will always be hidden if the user presses a key or clicks a mouse button.

Parameter *millisecondsDelay* is new since wxWidgets 2.9.5.

void wxRichToolTip::SetTipKind (wxTipKind *tipKind*)

Choose the tip kind, possibly none.

See wxTipKind documentation for the possible choices here.

By default the tip is positioned automatically, as if wxTipKind_Auto was used. Native MSW implementation doesn't support setting the tip kind explicitly and won't be used if this method is called with any value other than wxTip↩Kind_Auto.

Notice that using non automatic tooltip kind may result in the tooltip being positioned partially off screen and it's the callers responsibility to ensure that this doesn't happen in this case.

void wxRichToolTip::SetTitleFont (const wxFont & *font*)

Set the title text font.

By default it's emphasized using the font style or colour appropriate for the current platform. Calling this method prevents the native MSW implementation from being used as it doesn't support changing the font.

void wxRichToolTip::ShowFor (wxWindow * *win,* const wxRect * *rect* = NULL)

Show the tooltip for the given window and optionally specify where to show the tooltip.

By default the tooltip tip points to the (middle of the) specified window which must be non-NULL or, if *rect* is non-NULL, the middle of the specified wxRect.

The coordinates of the *rect* parameter are relative to the given window.

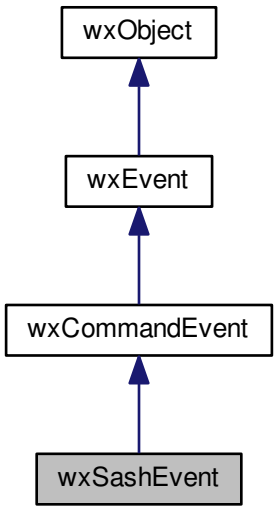
Currently the native MSW implementation is used only if *win* is a wxTextCtrl and *rect* is NULL. This limitation may be removed in the future.

Parameter *rect* is new since wxWidgets 2.9.5.

21.658 wxSashEvent Class Reference

`#include <wx/sashwin.h>`

Inheritance diagram for wxSashEvent:



21.658.1 Detailed Description

A sash event is sent when the sash of a wxSashWindow has been dragged by the user.

Remarks

When a sash belonging to a sash window is dragged by the user, and then released, this event is sent to the window, where it may be processed by an event table entry in a derived class, a plug-in event handler or an ancestor class. Note that the wxSashWindow doesn't change the window's size itself. It relies on the application's event handler to do that. This is because the application may have to handle other consequences of the resize, or it may wish to veto it altogether. The event handler should look at the drag rectangle: see wxSashEvent::GetDragRect to see what the new size of the window would be if the resize were to be applied. It should also call wxSashEvent::GetDragStatus to see whether the drag was OK or out of the current allowed range.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSashEvent& event)

Event macros:

- EVT_SASH_DRAGGED(id, func): Process a `wxEVT_SASH_DRAGGED` event, when the user has finished dragging a sash.
- EVT_SASH_DRAGGED_RANGE(id1, id2, func): Process a `wxEVT_SASH_DRAGGED_RANGE` event, when the user has finished dragging a sash. The event handler is called when windows with ids in the given range have their sashes dragged.

Library: wxAdvanced

Category: Events

See also

wxSashWindow, Events and Event Handling

Public Member Functions

- wxSashEvent (int id=0, wxSashEdgePosition edge=wxSASH_NONE)

  Constructor.
- wxRect GetDragRect () const

  Returns the rectangle representing the new size the window would be if the resize was applied.
- wxSashDragStatus GetDragStatus () const

  Returns the status of the sash: one of wxSASH_STATUS_OK, wxSASH_STATUS_OUT_OF_RANGE.
- wxSashEdgePosition GetEdge () const

  Returns the dragged edge.
- void SetEdge (wxSashEdgePosition edge)
- void SetDragRect (const wxRect &rect)
- void SetDragStatus (wxSashDragStatus status)

Additional Inherited Members

21.658.2 Constructor & Destructor Documentation

wxSashEvent::wxSashEvent (int *id* = 0, wxSashEdgePosition *edge* = wxSASH_NONE)

Constructor.

21.658.3 Member Function Documentation

wxRect wxSashEvent::GetDragRect ( ) const

Returns the rectangle representing the new size the window would be if the resize was applied.

It is up to the application to set the window size if required.

wxSashDragStatus wxSashEvent::GetDragStatus ( ) const

Returns the status of the sash: one of wxSASH_STATUS_OK, wxSASH_STATUS_OUT_OF_RANGE.

If the drag caused the notional bounding box of the window to flip over, for example, the drag will be out of rage.

wxSashEdgePosition wxSashEvent::GetEdge ( ) const

Returns the dragged edge.

The return value is one of wxSASH_TOP, wxSASH_RIGHT, wxSASH_BOTTOM, wxSASH_LEFT.

void wxSashEvent::SetDragRect (const wxRect & *rect*)

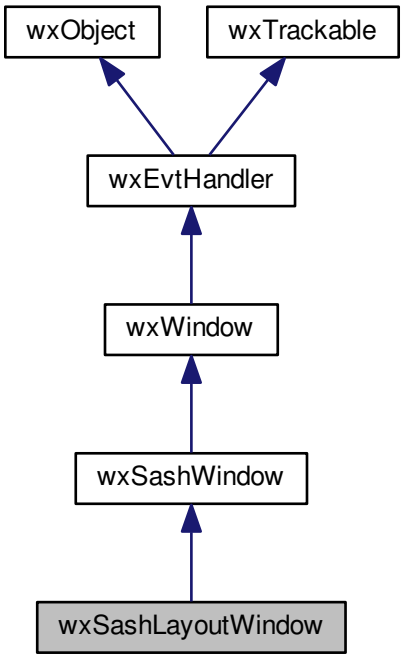
void wxSashEvent::SetDragStatus (wxSashDragStatus *status*)

void wxSashEvent::SetEdge (wxSashEdgePosition *edge*)

21.659 wxSashLayoutWindow Class Reference

`#include <wx/laywin.h>`

Inheritance diagram for wxSashLayoutWindow:



21.659.1 Detailed Description

wxSashLayoutWindow responds to OnCalculateLayout events generated by wxLayoutAlgorithm.

It allows the application to use simple accessors to specify how the window should be laid out, rather than having to respond to events.

The fact that the class derives from wxSashWindow allows sashes to be used if required, to allow the windows to be user-resizable.

The documentation for wxLayoutAlgorithm explains the purpose of this class in more detail.

For the window styles see wxSashWindow.

This class handles the EVT_QUERY_LAYOUT_INFO and EVT_CALCULATE_LAYOUT events for you. However, if you use sashes, see wxSashWindow for relevant event information. See also wxLayoutAlgorithm for information about the layout events.

Library: wxAdvanced

Category: Miscellaneous Windows

See also

wxLayoutAlgorithm, wxSashWindow, Events and Event Handling

Public Member Functions

- wxSashLayoutWindow ()

  Default ctor.
- wxSashLayoutWindow (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCLIP_CHILDREN|wxSW_3D, const wxString &name="layoutWindow")

  Constructs a sash layout window, which can be a child of a frame, dialog or any other non-control window.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCLIP_CHILDREN|wxSW_3D, const wxString &name="layoutWindow")

  Initializes a sash layout window, which can be a child of a frame, dialog or any other non-control window.
- wxLayoutAlignment GetAlignment () const

  Returns the alignment of the window: one of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.
- wxLayoutOrientation GetOrientation () const

  Returns the orientation of the window: one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.
- void OnCalculateLayout (wxCalculateLayoutEvent &event)

  The default handler for the event that is generated by wxLayoutAlgorithm.
- void OnQueryLayoutInfo (wxQueryLayoutInfoEvent &event)

  The default handler for the event that is generated by OnCalculateLayout to get size, alignment and orientation information for the window.
- void SetAlignment (wxLayoutAlignment alignment)

  Sets the alignment of the window (which edge of the available parent client area the window is attached to).
- void SetDefaultSize (const wxSize &size)

  Sets the default dimensions of the window.
- void SetOrientation (wxLayoutOrientation orientation)

  Sets the orientation of the window (the direction the window will stretch in, to fill the available parent client area).

Additional Inherited Members

21.659.2 Constructor & Destructor Documentation

wxSashLayoutWindow::wxSashLayoutWindow ( )

Default ctor.

wxSashLayoutWindow::wxSashLayoutWindow (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCLIP_CHILDREN|wxSW_3D, const wxString & *name* = "layoutWindow")

Constructs a sash layout window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent	Pointer to a parent window.
id	Window identifier. If -1, will automatically create an identifier.
pos	Window position. wxDefaultPosition is (-1, -1) which indicates that wxSashLayoutWindows should generate a default position for the window. If using the wxSashLayoutWindow class directly, supply an actual position.
size	Window size. wxDefaultSize is (-1, -1) which indicates that wxSashLayoutWindows should generate a default size for the window.
style	Window style. For window styles, please see wxSashLayoutWindow.
name	Window name.

21.659.3 Member Function Documentation

bool wxSashLayoutWindow::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCLIP_CHILDREN|wxSW_3D, const wxString & *name* = "layoutWindow")

Initializes a sash layout window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent	Pointer to a parent window.
id	Window identifier. If -1, will automatically create an identifier.
pos	Window position. wxDefaultPosition is (-1, -1) which indicates that wxSashLayoutWindows should generate a default position for the window. If using the wxSashLayoutWindow class directly, supply an actual position.
size	Window size. wxDefaultSize is (-1, -1) which indicates that wxSashLayoutWindows should generate a default size for the window.
style	Window style. For window styles, please see wxSashLayoutWindow.
name	Window name.

wxLayoutAlignment wxSashLayoutWindow::GetAlignment ( ) const

Returns the alignment of the window: one of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.

wxLayoutOrientation wxSashLayoutWindow::GetOrientation ( ) const

Returns the orientation of the window: one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.

void wxSashLayoutWindow::OnCalculateLayout (wxCalculateLayoutEvent & *event*)

The default handler for the event that is generated by wxLayoutAlgorithm.

The implementation of this function calls wxCalculateLayoutEvent::SetRect to shrink the provided size according to how much space this window takes up. For further details, see wxLayoutAlgorithm and wxCalculateLayoutEvent.

void wxSashLayoutWindow::OnQueryLayoutInfo (wxQueryLayoutInfoEvent & *event*)

The default handler for the event that is generated by OnCalculateLayout to get size, alignment and orientation information for the window.

The implementation of this function uses member variables as set by accessors called by the application.

For further details, see wxLayoutAlgorithm and wxQueryLayoutInfoEvent.

void wxSashLayoutWindow::SetAlignment (wxLayoutAlignment *alignment*)

Sets the alignment of the window (which edge of the available parent client area the window is attached to).

alignment is one of wxLAYOUT_TOP, wxLAYOUT_LEFT, wxLAYOUT_RIGHT, wxLAYOUT_BOTTOM.

void wxSashLayoutWindow::SetDefaultSize (const wxSize & *size*)

Sets the default dimensions of the window.

The dimension other than the orientation will be fixed to this value, and the orientation dimension will be ignored and the window stretched to fit the available space.

void wxSashLayoutWindow::SetOrientation (wxLayoutOrientation *orientation*)

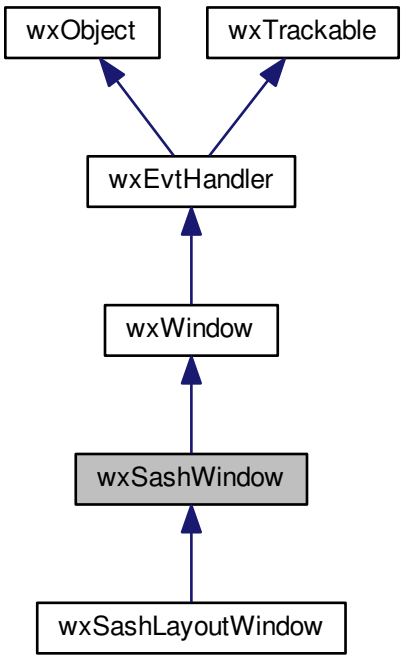
Sets the orientation of the window (the direction the window will stretch in, to fill the available parent client area).

orientation is one of wxLAYOUT_HORIZONTAL, wxLAYOUT_VERTICAL.

21.660 wxSashWindow Class Reference

`#include <wx/sashwin.h>`

Inheritance diagram for wxSashWindow:



21.660.1 Detailed Description

wxSashWindow allows any of its edges to have a sash which can be dragged to resize the window.

The actual content window will be created by the application as a child of wxSashWindow.

The window (or an ancestor) will be notified of a drag via a wxSashEvent notification.

Styles

This class supports the following styles:

- wxSW_3D: Draws a 3D effect sash and border.
- wxSW_3DSASH: Draws a 3D effect sash.
- wxSW_3DBORDER: Draws a 3D effect border.
- wxSW_BORDER: Draws a thin black border.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSashEvent& event)

Event macros for events emitted by this class:

- EVT_SASH_DRAGGED(id, func): Process a `wxEVT_SASH_DRAGGED` event, when the user has finished dragging a sash.
- EVT_SASH_DRAGGED_RANGE(id1, id2, func): Process a `wxEVT_SASH_DRAGGED_RANGE` event, when the user has finished dragging a sash. The event handler is called when windows with ids in the given range have their sashes dragged.

Library: wxAdvanced

Category: Miscellaneous Windows

See also

wxSashEvent, wxSashLayoutWindow, Events and Event Handling

Public Member Functions

- wxSashWindow ()

  Default ctor.
- wxSashWindow (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxCLIP_CHILDREN|wxSW_3D, const wxString &name="sashWindow")

  Constructs a sash window, which can be a child of a frame, dialog or any other non-control window.
- virtual ~wxSashWindow ()

  Destructor.
- virtual int GetMaximumSizeX () const

  Gets the maximum window size in the x direction.
- virtual int GetMaximumSizeY () const

  Gets the maximum window size in the y direction.

- virtual int GetMinimumSizeX () const

  Gets the minimum window size in the x direction.
- virtual int GetMinimumSizeY () const

  Gets the minimum window size in the y direction.
- bool GetSashVisible (wxSashEdgePosition edge) const

  Returns true if a sash is visible on the given edge, false otherwise.
- virtual void SetMaximumSizeX (int min)

  Sets the maximum window size in the x direction.
- virtual void SetMaximumSizeY (int min)

  Sets the maximum window size in the y direction.
- virtual void SetMinimumSizeX (int min)

  Sets the minimum window size in the x direction.
- virtual void SetMinimumSizeY (int min)

  Sets the minimum window size in the y direction.
- void SetSashVisible (wxSashEdgePosition edge, bool visible)

  Call this function to make a sash visible or invisible on a particular edge.
- int GetEdgeMargin (wxSashEdgePosition edge) const

  Get border size.
- void SetDefaultBorderSize (int width)

  Sets the default sash border size.
- int GetDefaultBorderSize () const

  Gets the default sash border size.
- void SetExtraBorderSize (int width)

  Sets the additional border size between child and sash window.
- int GetExtraBorderSize () const

  Gets the addition border size between child and sash window.
- wxSashEdgePosition SashHitTest (int x, int y, int tolerance=2)

  Tests for x, y over sash.
- void SizeWindows ()

  Resizes subwindows.

Additional Inherited Members

21.660.2 Constructor & Destructor Documentation

wxSashWindow::wxSashWindow ( )

Default ctor.

wxSashWindow::wxSashWindow (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxCLIP_CHILDREN|wxSW_3D, const wxString & *name* = "sashWindow")

Constructs a sash window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent	Pointer to a parent window.

id	Window identifier. If -1, will automatically create an identifier.
pos	Window position. wxDefaultPosition is (-1, -1) which indicates that wxSashWindows should generate a default position for the window. If using the wxSashWindow class directly, supply an actual position.
size	Window size. wxDefaultSize is (-1, -1) which indicates that wxSashWindows should generate a default size for the window.
style	Window style. For window styles, please see wxSashWindow.
name	Window name.

virtual wxSashWindow::~wxSashWindow ( ) [virtual]

Destructor.

21.660.3 Member Function Documentation

int wxSashWindow::GetDefaultBorderSize ( ) const

Gets the default sash border size.

int wxSashWindow::GetEdgeMargin (wxSashEdgePosition *edge*) const

Get border size.

int wxSashWindow::GetExtraBorderSize ( ) const

Gets the addition border size between child and sash window.

virtual int wxSashWindow::GetMaximumSizeX ( ) const [virtual]

Gets the maximum window size in the x direction.

virtual int wxSashWindow::GetMaximumSizeY ( ) const [virtual]

Gets the maximum window size in the y direction.

virtual int wxSashWindow::GetMinimumSizeX ( ) const [virtual]

Gets the minimum window size in the x direction.

virtual int wxSashWindow::GetMinimumSizeY ( ) const [virtual]

Gets the minimum window size in the y direction.

bool wxSashWindow::GetSashVisible (wxSashEdgePosition *edge*) const

Returns true if a sash is visible on the given edge, false otherwise.

Parameters

edge	Edge. One of wxSASH_TOP, wxSASH_RIGHT, wxSASH_BOTTOM, wxSASH_LEFT.

See also

SetSashVisible()

wxSashEdgePosition wxSashWindow::SashHitTest (int *x,* int *y,* int *tolerance* = 2)

Tests for x, y over sash.

void wxSashWindow::SetDefaultBorderSize (int *width*)

Sets the default sash border size.

void wxSashWindow::SetExtraBorderSize (int *width*)

Sets the additional border size between child and sash window.

virtual void wxSashWindow::SetMaximumSizeX (int *min*) [virtual]

Sets the maximum window size in the x direction.

virtual void wxSashWindow::SetMaximumSizeY (int *min*) [virtual]

Sets the maximum window size in the y direction.

virtual void wxSashWindow::SetMinimumSizeX (int *min*) [virtual]

Sets the minimum window size in the x direction.

virtual void wxSashWindow::SetMinimumSizeY (int *min*) [virtual]

Sets the minimum window size in the y direction.

void wxSashWindow::SetSashVisible (wxSashEdgePosition *edge,* bool *visible*)

Call this function to make a sash visible or invisible on a particular edge.

Parameters

edge	Edge to change. One of wxSASH_TOP, wxSASH_RIGHT, wxSASH_BOTTOM, wxSASH_↩ LEFT.
visible	true to make the sash visible, false to make it invisible.

See also

GetSashVisible()

void wxSashWindow::SizeWindows ( )

Resizes subwindows.

21.661 wxScopedArray< T > Class Template Reference

`#include <wx/scopedarray.h>`

21.661.1 Detailed Description

template<class T>class wxScopedArray< T >

A scoped array template class.

This is a simple scoped smart pointer array implementation that is similar to the Boost smart pointers (see http://www.boost.org/) but rewritten to use macros instead.

This class is similar to boost scoped_array class: http://www.boost.org/doc/libs/1_37_0/libs/smart_ptr/scoped_array.htm

Notice that objects of this class intentionally cannot be copied.

Library: wxBase

Category: Smart Pointers

Example:

Below is an example of using a wxWidgets scoped smart pointer and pointer array.

```
class MyClass { ... };

// declare a smart pointer to a MyClass called wxMyClassPtr
wxDECLARE_SCOPED_PTR(MyClass, wxMyClassPtr)
// declare a smart pointer to an array of chars
wxDECLARE_SCOPED_ARRAY(char, wxCharArray)

...

// define the first pointer class, must be complete
wxDEFINE_SCOPED_PTR(MyClass, wxMyClassPtr)
// define the second pointer class
wxDEFINE_SCOPED_ARRAY(char, wxCharArray)

// create an object with a new pointer to MyClass
wxMyClassPtr theObj(new MyClass());
// reset the pointer (deletes the previous one)
theObj.reset(new MyClass());

// access the pointer
theObj->MyFunc();

// create an object with a new array of chars
wxCharArray theCharObj(new char[100]);

// access the array
theCharObj[0] = "!";
```

Declaring new smart pointer types:

```
wxDECLAR_SCOPED_ARRAY( TYPE,        // type of the values
                       CLASSNAME ); // name of the class
```

A smart pointer holds a pointer to an object (which must be complete when wxDEFINE_SCOPED_ARRAY() is called).

The memory used by the object is deleted when the smart pointer goes out of scope. The first argument of the macro is the pointer type, the second is the name of the new smart pointer class being created. Below we will use wxScopedArray to represent the scoped pointer array class, but the user may create the class with any legal name.

Library: wxBase

Category: Smart Pointers

See also

wxScopedPtr

Public Types

- typedef T element_type

  The type of the array elements.

Public Member Functions

- wxScopedArray (type *T=NULL)

  Creates the smart pointer with the given pointer or none if NULL.
- const T * get ()

  This operator gets the pointer stored in the smart pointer or returns NULL if there is none.
- const T & operator[] (long int i)

  This operator acts like the standard [] indexing operator for C++ arrays.
- reset (T *p=NULL)

  Deletes the currently held pointer and sets it to 'p' or to NULL if no arguments are specified.
- swap (wxScopedArray &ot)

  Swap the pointer inside the smart pointer with ot.
- wxScopedArray (T *array=NULL)

  Constructor takes ownership of the given array.
- wxScopedArray (size_t count)

  Constructor allocating a new array of the specified size.
- ~wxScopedArray ()

  Destructor destroy the array.
- operator unspecified_bool_type () const

  Conversion to a boolean expression (in a variant which is not convertible to anything but a boolean expression).
- void reset (T *array=NULL)

  Change the array pointer stored.
- T & operator[] (size_t n) const

  Return the n-th element of the array.
- T * get () const

  Return the array pointer.
- void swap (wxScopedArray &other)

  Swaps the contents of this array with another one.

21.661.2 Member Typedef Documentation

template<class T > typedef T wxScopedArray< T >::element_type

The type of the array elements.

21.661.3 Constructor & Destructor Documentation

template<class T > wxScopedArray< T >::wxScopedArray (type * *T* = NULL)

Creates the smart pointer with the given pointer or none if NULL.

On compilers that support it, this uses the explicit keyword.

template<class T > wxScopedArray< T >::wxScopedArray (T * *array* = NULL) [explicit]

Constructor takes ownership of the given array.

If *array* is NULL, reset() must presumably be called later.

Parameters

array	An array allocated using `new`[] or NULL.

template<class T > wxScopedArray< T >::wxScopedArray (size_t *count*) [explicit]

Constructor allocating a new array of the specified size.

Parameters

count	The number of elements to allocate.

Since

3.1.0

template<class T > wxScopedArray< T >::~wxScopedArray ( )

Destructor destroy the array.

21.661.4 Member Function Documentation

template<class T > const T* wxScopedArray< T >::get ( )

This operator gets the pointer stored in the smart pointer or returns NULL if there is none.

template<class T > T* wxScopedArray< T >::get ( ) const

Return the array pointer.

The returned pointer may be NULL. It must not be deleted by the caller, call `reset(NULL)` instead.

template<class T > wxScopedArray< T >::operator unspecified_bool_type ( ) const

Conversion to a boolean expression (in a variant which is not convertible to anything but a boolean expression).

If this class contains a valid array it will return true, if it contains a NULL pointer it will return false.

template<class T > const T& wxScopedArray< T >::operator[] (long int *i*)

This operator acts like the standard [] indexing operator for C++ arrays.

The function does not do bounds checking.

template<class T > T& wxScopedArray< T >::operator[] (size_t *n*) const

Return the n-th element of the array.

Must not be called if the array has no valid pointer.

template<class T > wxScopedArray< T >::reset (T * *p* = NULL)

Deletes the currently held pointer and sets it to 'p' or to NULL if no arguments are specified.

This function does check to make sure that the pointer you are assigning is not the same pointer that is already stored.

template<class T > void wxScopedArray< T >::reset (T * *array* = NULL)

Change the array pointer stored.

The previously stored array is deleted.

Parameters

array	An array allocated using new[] or NULL.

template<class T > wxScopedArray< T >::swap (wxScopedArray< T > & *ot*)

Swap the pointer inside the smart pointer with *ot*.

The pointer being swapped must be of the same type (hence the same class name).

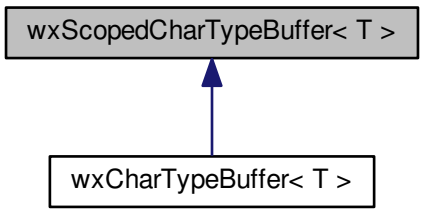
template<class T > void wxScopedArray< T >::swap (wxScopedArray< T > & *other*)

Swaps the contents of this array with another one.

21.662 wxScopedCharTypeBuffer< T > Class Template Reference

#include <wx/buffer.h>

Inheritance diagram for wxScopedCharTypeBuffer< T >:



21.662.1 Detailed Description

template<typename T>class wxScopedCharTypeBuffer< T >

wxScopedCharTypeBuffer<T> is a template class for storing characters.

Data are stored in reference-counted buffer. In other words, making a copy of wxScopedCharTypeBuffer<T> will *not* make another copy of the stored string data, it will still point to the same location in memory.

wxScopedCharTypeBuffer<T> supports two storage modes: owned and non-owned. "Owned" data buffer (created with CreateOwned() or wxCharTypeBuffer<T> derived class) owns the data and frees them when the last buffer pointing to them is destroyed.

"Non-owned" buffer (created with CreateNonOwned()), on the other hand, references data owned by somebody else – typical use is by wxString::mb_str() or wxString::wc_str(), which may return non-owned buffer pointing to wxString's internal store.

Because of this, the validity of data stored in wxScopedCharTypeBuffer<T> is limited by the lifetime of the "parent" object that created the buffer (e.g. the wxString on which mb_str() was called).

If you need to preserve the data for longer, assign it to wxCharTypeBuffer<T> instead of wxScopedCharTypeBuffer<T>. On the other hand, use wxScopedCharTypeBuffer<T> if the buffer is to be destroyed before the "parent" object – typical use would be creating it on the stack and destroying when it goes out of scope (hence the class' name).

Template Parameters

T	The type of the characters stored in this class.

Since

2.9.0

Library: None; this class implementation is entirely header-based.

Category: Data Structures

Public Types

- typedef T CharType

  Stored characters type.

Public Member Functions

- wxScopedCharTypeBuffer ()

  Default constructor, creates NULL buffer.
- wxScopedCharTypeBuffer (const wxScopedCharTypeBuffer &src)

  Copy constructor.
- wxScopedCharTypeBuffer & operator= (const wxScopedCharTypeBuffer &src)

  Assignment operator behaves in the same way as the copy constructor.
- ~wxScopedCharTypeBuffer ()

  Destructor.
- CharType * release () const

  Returns the internal pointer and resets the buffer.
- void reset ()

  Resets the buffer to NULL, freeing the data if necessary.
- CharType * data ()

  Returns pointer to the stored data.
- const CharType * data () const

  Returns const pointer to the stored data.
- size_t length () const

  Returns length of the string stored.
- operator const CharType * () const

  Implicit conversion to C string.
- CharType operator[] (size_t n) const

  Random access to the stored C string.

Static Public Member Functions

- static const wxScopedCharTypeBuffer CreateNonOwned (const CharType *str, size_t len=wxNO_LEN)

  Creates non-owned buffer from string data str.
- static const wxScopedCharTypeBuffer CreateOwned (CharType *str, size_t len=wxNO_LEN)

  Creates owned buffer from str and takes ownership of it.

21.662.2 Member Typedef Documentation

template<typename T> typedef T wxScopedCharTypeBuffer< T >::CharType

Stored characters type.

21.662.3 Constructor & Destructor Documentation

template<typename T> wxScopedCharTypeBuffer< T >::wxScopedCharTypeBuffer ( )

Default constructor, creates NULL buffer.

template<typename T> wxScopedCharTypeBuffer< T >::wxScopedCharTypeBuffer (const wxScopedCharTypeBuffer< T > & *src*)

Copy constructor.

Increases reference count on the data, does *not* make wxStrdup() copy of the data.

template<typename T> wxScopedCharTypeBuffer< T >::~wxScopedCharTypeBuffer ( )

Destructor.

Frees stored data if it is in "owned" mode and data's reference count reaches zero.

21.662.4 Member Function Documentation

template<typename T> static const wxScopedCharTypeBuffer wxScopedCharTypeBuffer< T >::CreateNonOwned (const CharType * *str*, size_t *len* = wxNO_LEN) [static]

Creates non-owned buffer from string data *str*.

The buffer's destructor will not destroy *str*. The returned buffer's data is valid only as long as *str* is valid.

Parameters

str	String data.
len	If specified, length of the string, otherwise the string is considered to be NUL-terminated.

template<typename T> static const wxScopedCharTypeBuffer wxScopedCharTypeBuffer< T >::CreateOwned (CharType * *str*, size_t *len* = wxNO_LEN) [static]

Creates owned buffer from *str* and takes ownership of it.

The buffer's destructor will free *str* when its reference count reaches zero (initial count is 1).

Parameters

str	String data.
len	If specified, length of the string, otherwise the string is considered to be NUL-terminated.

template<typename T> CharType* wxScopedCharTypeBuffer< T >::data ( )

Returns pointer to the stored data.

template<typename T> const CharType* wxScopedCharTypeBuffer< T >::data ( ) const

Returns const pointer to the stored data.

template<typename T> size_t wxScopedCharTypeBuffer< T >::length ( ) const

Returns length of the string stored.

template<typename T> wxScopedCharTypeBuffer< T >::operator const CharType * ( ) const

Implicit conversion to C string.

template<typename T> wxScopedCharTypeBuffer& wxScopedCharTypeBuffer< T >::operator= (const wxScopedCharTypeBuffer< T > & *src*)

Assignment operator behaves in the same way as the copy constructor.

template<typename T> CharType wxScopedCharTypeBuffer< T >::operator[] (size_t *n*) const

Random access to the stored C string.

template<typename T> CharType* wxScopedCharTypeBuffer< T >::release ( ) const

Returns the internal pointer and resets the buffer.

It's the caller responsibility to deallocate the returned pointer using `free()` function.

Notice that this method is dangerous because it can only be called on a non-shared owning buffer. Calling it on any other kind of buffer object will result in a crash after the pointer is freed, so avoid using it unless absolutely necessary and you are absolutely certain that the buffer is not shared.

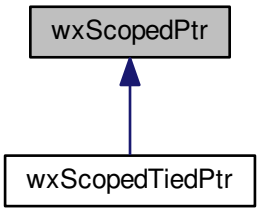
template<typename T> void wxScopedCharTypeBuffer< T >::reset ( )

Resets the buffer to NULL, freeing the data if necessary.

21.663 wxScopedPtr Class Reference

`#include <wx/scopedptr.h>`

Inheritance diagram for wxScopedPtr:



21.663.1 Detailed Description

This is a simple scoped smart pointer implementation that is similar to the Boost smart pointers (see http://www.boost.org) but rewritten to use macros instead.

Since wxWidgets 2.9.0 there is also a templated version of this class with the same name. See wxScopedPtr<T>.

A smart pointer holds a pointer to an object. The memory used by the object is deleted when the smart pointer goes out of scope. This class is different from the `std::auto_ptr<>` in so far as it doesn't provide copy constructor nor assignment operator. This limits what you can do with it but is much less surprising than the "destructive copy" behaviour of the standard class.

Example:

Below is an example of using a wxWidgets scoped smart pointer and pointer array.

```
class MyClass{ ... };

// declare a smart pointer to a MyClass called wxMyClassPtr
wxDECLARE_SCOPED_PTR(MyClass, wxMyClassPtr)
```

```
// declare a smart pointer to an array of chars
wxDECLARE_SCOPED_ARRAY(char, wxCharArray)

...

// define the first pointer class, must be complete
wxDEFINE_SCOPED_PTR(MyClass, wxMyClassPtr)
// define the second pointer class
wxDEFINE_SCOPED_ARRAY(char, wxCharArray)

// create an object with a new pointer to MyClass
wxMyClassPtr theObj(new MyClass());
// reset the pointer (deletes the previous one)
theObj.reset(new MyClass());

// access the pointer
theObj->MyFunc();

// create an object with a new array of chars
wxCharArray theCharObj(new char[100]);

// access the array
theCharObj[0] = "!";
```

21.663.2 Declaring new smart pointer types

To declare the smart pointer class `CLASSNAME` containing pointer to a (possibly incomplete) type `TYPE` you should use

```
wxDECLARE_SCOPED_PTR( TYPE,        // type of the values
                      CLASSNAME ); // name of the class
```

And later, when `TYPE` is fully defined, you must also use

```
wxDEFINE_SCOPED_PTR( TYPE, CLASSNAME );
```

to implement the scoped pointer class.

The first argument of these macro is the pointer type, the second is the name of the new smart pointer class being created. Below we will use wxScopedPtr to represent the scoped pointer class, but the user may create the class with any legal name.

Alternatively, if you don't have to separate the point of declaration and definition of this class and if you accept the standard naming convention, that is that the scoped pointer for the class `Foo` is called `FooPtr`, you can use a single macro which replaces two macros above:

```
wxDEFINE_SCOPED_PTR_TYPE( TYPE );
```

Once again, in this cass `CLASSNAME` will be `TYPEPtr`.

Library: wxBase

Category: Smart Pointers

See also

wxScopedArray

Public Member Functions

- wxScopedPtr (type *T=NULL)

  Creates the smart pointer with the given pointer or none if NULL.
- ~wxScopedPtr ()

Destructor frees the pointer help by this object if it is not NULL.

- T * get () const

  This operator gets the pointer stored in the smart pointer or returns NULL if there is none.

- T & operator* () const

  This operator works like the standard C++ pointer operator to return the object being pointed to by the pointer.

- T * operator-> () const

  Smart pointer member access.

- T * release ()

  Returns the currently hold pointer and resets the smart pointer object to NULL.

- reset (T *p=NULL)

  Deletes the currently held pointer and sets it to p or to NULL if no arguments are specified.

- swap (wxScopedPtr &other)

  Swap the pointer inside the smart pointer with other.

21.663.3 Constructor & Destructor Documentation

wxScopedPtr::wxScopedPtr (type * *T* = NULL) [explicit]

Creates the smart pointer with the given pointer or none if NULL.

On compilers that support it, this uses the explicit keyword.

wxScopedPtr::~wxScopedPtr ( )

Destructor frees the pointer help by this object if it is not NULL.

21.663.4 Member Function Documentation

T* wxScopedPtr::get ( ) const

This operator gets the pointer stored in the smart pointer or returns NULL if there is none.

T& wxScopedPtr::operator* ( ) const

This operator works like the standard C++ pointer operator to return the object being pointed to by the pointer.

If the internal pointer is NULL this method will cause an assert in debug mode.

T* wxScopedPtr::operator-> ( ) const

Smart pointer member access.

Returns pointer to its object.

If the internal pointer is NULL this method will cause an assert in debug mode.

T* wxScopedPtr::release ( )

Returns the currently hold pointer and resets the smart pointer object to NULL.

Remarks

After a call to this function the caller is responsible for deleting the pointer.

wxScopedPtr::reset (T * *p* = NULL)

Deletes the currently held pointer and sets it to *p* or to NULL if no arguments are specified.

Note

This function does check to make sure that the pointer you are assigning is not the same pointer that is already stored.

wxScopedPtr::swap (wxScopedPtr & *other*)

Swap the pointer inside the smart pointer with *other*.

The pointer being swapped must be of the same type (hence the same class name).

21.664 wxScopedPtr< T > Class Template Reference

`#include <wx/scopedptr.h>`

21.664.1 Detailed Description

template<typename T>class wxScopedPtr< T >

A scoped pointer template class.

It is the template version of the old-style scoped pointer macros.

Notice that objects of this class intentionally cannot be copied.

Library: wxBase

Category: Smart Pointers

See also

wxSharedPtr<T>, wxWeakRef<T>

Public Member Functions

- wxScopedPtr (T *ptr=NULL)

  Constructor takes ownership of the pointer.
- ~wxScopedPtr ()

  Destructor deletes the pointer.
- T * get () const

  Returns pointer to object or NULL.
- operator unspecified_bool_type () const

  Conversion to a boolean expression (in a variant which is not convertible to anything but a boolean expression).
- T & operator* () const

  Returns a reference to the object.
- T * operator-> () const

  Smart pointer member access.

- T * release ()

  Releases the current pointer and returns it.
- void reset (T *ptr=NULL)

  Reset pointer to the value of ptr.
- void swap (wxScopedPtr< T > &ot)

  Swaps pointers.

21.664.2 Constructor & Destructor Documentation

template<typename T > wxScopedPtr< T >::wxScopedPtr (T * *ptr* = NULL)

Constructor takes ownership of the pointer.

Parameters

ptr	Pointer allocated with `new` or NULL.

template<typename T > wxScopedPtr< T >::~wxScopedPtr ( )

Destructor deletes the pointer.

21.664.3 Member Function Documentation

template<typename T > T* wxScopedPtr< T >::get ( ) const

Returns pointer to object or NULL.

template<typename T > wxScopedPtr< T >::operator unspecified_bool_type ( ) const

Conversion to a boolean expression (in a variant which is not convertible to anything but a boolean expression).

If this class contains a valid pointer it will return true, if it contains a NULL pointer it will return false.

template<typename T > T& wxScopedPtr< T >::operator* ( ) const

Returns a reference to the object.

If the internal pointer is NULL this method will cause an assert in debug mode.

template<typename T > T* wxScopedPtr< T >::operator-> ( ) const

Smart pointer member access.

Returns pointer to object.

If the internal pointer is NULL this method will cause an assert in debug mode.

template<typename T > T* wxScopedPtr< T >::release ( )

Releases the current pointer and returns it.

Remarks

Afterwards the caller is responsible for deleting the data contained in the scoped pointer before.

template<typename T > void wxScopedPtr< T >::reset (T * *ptr* = NULL)

Reset pointer to the value of *ptr*.

The previous pointer will be deleted.

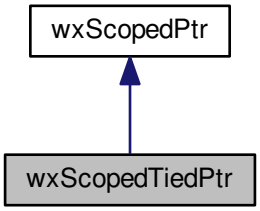
template<typename T > void wxScopedPtr< T >::swap (wxScopedPtr< T > & *ot*)

Swaps pointers.

21.665 wxScopedTiedPtr Class Reference

`#include <wx/scopedptr.h>`

Inheritance diagram for wxScopedTiedPtr:



21.665.1 Detailed Description

This is a variation on the topic of wxScopedPtr.

This class is also a smart pointer but in addition it "ties" the pointer value to another variable. In other words, during the life time of this class the value of that variable is set to be the same as the value of the pointer itself and it is reset to its old value when the object is destroyed. This class is especially useful when converting the existing code (which may already store the pointers value in some variable) to the smart pointers.

Library: wxBase

Category: Smart Pointers

Public Member Functions

- wxScopedTiedPtr (T **ppTie, T *ptr)

  Constructor creates a smart pointer initialized with ptr and stores ptr in the location specified by ppTie which must not be NULL.
- ~wxScopedTiedPtr ()

  Destructor frees the pointer help by this object and restores the value stored at the tied location (as specified in the wxScopedTiedPtr() constructor) to the old value.

21.665.2 Constructor & Destructor Documentation

wxScopedTiedPtr::wxScopedTiedPtr (T ∗∗ *ppTie*, T ∗ *ptr*)

Constructor creates a smart pointer initialized with *ptr* and stores *ptr* in the location specified by *ppTie* which must not be NULL.

wxScopedTiedPtr::∼wxScopedTiedPtr ( )

Destructor frees the pointer help by this object and restores the value stored at the tied location (as specified in the wxScopedTiedPtr() constructor) to the old value.

Warning

This location may now contain an uninitialized value if it hadn't been initialized previously, in particular don't count on it magically being NULL!

21.666 wxScopeGuard Class Reference

`#include <wx/scopeguard.h>`

21.666.1 Detailed Description

Scope guard is an object which allows executing an action on scope exit.

The objects of this class must be constructed using wxMakeGuard() function.

Library: None; this class implementation is entirely header-based.

Category: Miscellaneous

Public Member Functions

- void Dismiss ()

  Call this method to dismiss the execution of the action on scope exit.

21.666.2 Member Function Documentation

void wxScopeGuard::Dismiss ( )

Call this method to dismiss the execution of the action on scope exit.

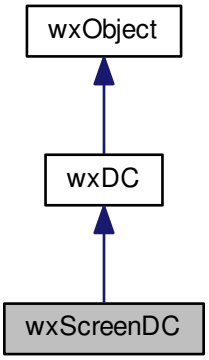
A typical example:

```
Update1();

// ensure that changes done so far are rolled back if the next
// operation throws
wxScopeGuard guard = wxMakeGuard(RollBack);
Update2();

// it didn't throw so commit the changes, i.e. avoid rolling back
guard.Dismiss();
```

21.667 wxScreenDC Class Reference

`#include <wx/dcscreen.h>`

Inheritance diagram for wxScreenDC:



21.667.1 Detailed Description

A wxScreenDC can be used to paint on the screen.

This should normally be constructed as a temporary stack object; don't store a wxScreenDC object.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxMemoryDC, wxPaintDC, wxClientDC, wxWindowDC

Public Member Functions

- wxScreenDC ()

  Constructor.

Static Public Member Functions

- static bool EndDrawingOnTop ()

  Use this in conjunction with StartDrawingOnTop().
- static bool StartDrawingOnTop (wxWindow *window)

  Use this in conjunction with EndDrawingOnTop() to ensure that drawing to the screen occurs on top of existing windows.
- static bool StartDrawingOnTop (wxRect *rect=NULL)

  Use this in conjunction with EndDrawingOnTop() to ensure that drawing to the screen occurs on top of existing windows.

Additional Inherited Members

21.667.2 Constructor & Destructor Documentation

wxScreenDC::wxScreenDC ( )

Constructor.

21.667.3 Member Function Documentation

static bool wxScreenDC::EndDrawingOnTop ( ) [static]

Use this in conjunction with StartDrawingOnTop().

This function destroys the temporary window created to implement on-top drawing (X only).

static bool wxScreenDC::StartDrawingOnTop (wxWindow * *window*) [static]

Use this in conjunction with EndDrawingOnTop() to ensure that drawing to the screen occurs on top of existing windows.

Without this, some window systems (such as X) only allow drawing to take place underneath other windows.

This version of StartDrawingOnTop() is used to specify that the area that will be drawn on coincides with the given window. It is recommended that an area of the screen is specified with StartDrawingOnTop(wxRect*) because with large regions, flickering effects are noticeable when destroying the temporary transparent window used to implement this feature.

You might use this function when implementing a drag feature, for example as in the wxSplitterWindow implementation.

Remarks

This function is probably obsolete since the X implementations allow drawing directly on the screen now. However, the fact that this function allows the screen to be refreshed afterwards, may be useful to some applications.

static bool wxScreenDC::StartDrawingOnTop (wxRect * *rect* = NULL) [static]

Use this in conjunction with EndDrawingOnTop() to ensure that drawing to the screen occurs on top of existing windows.

Without this, some window systems (such as X) only allow drawing to take place underneath other windows.

This version of StartDrawingOnTop() is used to specify an area of the screen which is to be drawn on. If NULL is passed, the whole screen is available. It is recommended that an area of the screen is specified with this function rather than with StartDrawingOnTop(wxWindow*), because with large regions, flickering effects are noticeable when destroying the temporary transparent window used to implement this feature.

You might use this function when implementing a drag feature, for example as in the wxSplitterWindow implementation.

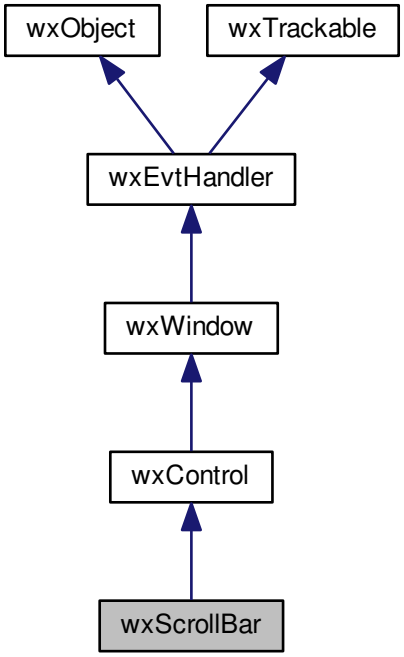
Remarks

This function is probably obsolete since the X implementations allow drawing directly on the screen now. However, the fact that this function allows the screen to be refreshed afterwards, may be useful to some applications.

21.668 wxScrollBar Class Reference

`#include <wx/scrolbar.h>`

Inheritance diagram for wxScrollBar:



21.668.1 Detailed Description

A wxScrollBar is a control that represents a horizontal or vertical scrollbar.

It is distinct from the two scrollbars that some windows provide automatically, but the two types of scrollbar share the way events are received.

Remarks

A scrollbar has the following main attributes: range, thumb size, page size, and position. The range is the total number of units associated with the view represented by the scrollbar. For a table with 15 columns, the range would be 15. The thumb size is the number of units that are currently visible. For the table example, the window might be sized so that only 5 columns are currently visible, in which case the application would set the thumb size to 5. When the thumb size becomes the same as or greater than the range, the scrollbar will be automatically hidden on most platforms. The page size is the number of units that the scrollbar should scroll by, when 'paging' through the data. This value is normally the same as the thumb size length, because it is natural to assume that the visible window size defines a page. The scrollbar position is the current thumb position. Most applications will find it convenient to provide a function called AdjustScrollbars() which can be called initially, from an OnSize event handler, and whenever the application data changes in size. It will adjust the view, object and page size according to the size of the window and the size of the data.

Styles

This class supports the following styles:

- wxSB_HORIZONTAL: Specifies a horizontal scrollbar.
- wxSB_VERTICAL: Specifies a vertical scrollbar.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxScrollEvent& event)

Event macros for events emitted by this class: You can use EVT_COMMAND_SCROLL... macros with window IDs for when intercepting scroll events from controls, or EVT_SCROLL... macros without window IDs for intercepting scroll events from the receiving window – except for this, the macros behave exactly the same.

- EVT_SCROLL(func): Process all scroll events.
- EVT_SCROLL_TOP(func): Process `wxEVT_SCROLL_TOP` scroll to top or leftmost (minimum) position events.
- EVT_SCROLL_BOTTOM(func): Process `wxEVT_SCROLL_BOTTOM` scroll to bottom or rightmost (maximum) position events.
- EVT_SCROLL_LINEUP(func): Process `wxEVT_SCROLL_LINEUP` line up or left events.
- EVT_SCROLL_LINEDOWN(func): Process `wxEVT_SCROLL_LINEDOWN` line down or right events.
- EVT_SCROLL_PAGEUP(func): Process `wxEVT_SCROLL_PAGEUP` page up or left events.
- EVT_SCROLL_PAGEDOWN(func): Process `wxEVT_SCROLL_PAGEDOWN` page down or right events.
- EVT_SCROLL_THUMBTRACK(func): Process `wxEVT_SCROLL_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_SCROLL_THUMBRELEASE(func): Process `wxEVT_SCROLL_THUMBRELEASE` thumb release events.
- EVT_SCROLL_CHANGED(func): Process `wxEVT_SCROLL_CHANGED` end of scrolling events (MSW only).
- EVT_COMMAND_SCROLL(id, func): Process all scroll events.
- EVT_COMMAND_SCROLL_TOP(id, func): Process `wxEVT_SCROLL_TOP` scroll to top or leftmost (minimum) position events.
- EVT_COMMAND_SCROLL_BOTTOM(id, func): Process `wxEVT_SCROLL_BOTTOM` scroll to bottom or rightmost (maximum) position events.
- EVT_COMMAND_SCROLL_LINEUP(id, func): Process `wxEVT_SCROLL_LINEUP` line up or left events.
- EVT_COMMAND_SCROLL_LINEDOWN(id, func): Process `wxEVT_SCROLL_LINEDOWN` line down or right events.
- EVT_COMMAND_SCROLL_PAGEUP(id, func): Process `wxEVT_SCROLL_PAGEUP` page up or left events.
- EVT_COMMAND_SCROLL_PAGEDOWN(id, func): Process `wxEVT_SCROLL_PAGEDOWN` page down or right events.
- EVT_COMMAND_SCROLL_THUMBTRACK(id, func): Process `wxEVT_SCROLL_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_COMMAND_SCROLL_THUMBRELEASE(func): Process `wxEVT_SCROLL_THUMBRELEASE` thumb release events.
- EVT_COMMAND_SCROLL_CHANGED(func): Process `wxEVT_SCROLL_CHANGED` end of scrolling events (MSW only).

21.668.2 The difference between EVT_SCROLL_THUMBRELEASE and EVT_SCROLL_CHANGED

The EVT_SCROLL_THUMBRELEASE event is only emitted when actually dragging the thumb using the mouse and releasing it (This EVT_SCROLL_THUMBRELEASE event is also followed by an EVT_SCROLL_CHANGED event).

The EVT_SCROLL_CHANGED event also occurs when using the keyboard to change the thumb position, and when clicking next to the thumb (In all these cases the EVT_SCROLL_THUMBRELEASE event does not happen).

In short, the EVT_SCROLL_CHANGED event is triggered when scrolling/moving has finished independently of the way it had started. Please see the widgets sample ("Slider" page) to see the difference between EVT_SCROLL_↩ THUMBRELEASE and EVT_SCROLL_CHANGED in action.

Library: wxCore

Category: Controls

See also

Scrolled Windows, Events and Event Handling, wxScrolled

Public Member Functions

- wxScrollBar ()

  Default constructor.

- wxScrollBar (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSB_HORIZONTAL, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxScrollBarNameStr)

  Constructor, creating and showing a scrollbar.

- virtual ~wxScrollBar ()

  Destructor, destroying the scrollbar.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSB_HORIZONTAL, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxScrollBarNameStr)

  Scrollbar creation function called by the scrollbar constructor.

- virtual int GetPageSize () const

  Returns the page size of the scrollbar.

- virtual int GetRange () const

  Returns the length of the scrollbar.

- virtual int GetThumbPosition () const

  Returns the current position of the scrollbar thumb.

- virtual int GetThumbSize () const

  Returns the thumb or 'view' size.

- virtual void SetScrollbar (int position, int thumbSize, int range, int pageSize, bool refresh=true)

  Sets the scrollbar properties.

- virtual void SetThumbPosition (int viewStart)

  Sets the position of the scrollbar.

- bool IsVertical () const

  Returns true for scrollbars that have the vertical style set.

Additional Inherited Members

21.668.3 Constructor & Destructor Documentation

wxScrollBar::wxScrollBar ( )

Default constructor.

wxScrollBar::wxScrollBar (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSB_HORIZONTAL, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxScrollBarNameStr`)

Constructor, creating and showing a scrollbar.

Parameters

parent	Parent window. Must be non-NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxScrollBar.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxScrollBar::~wxScrollBar ( ) `[virtual]`

Destructor, destroying the scrollbar.

21.668.4 Member Function Documentation

bool wxScrollBar::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSB_HORIZONTAL, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxScrollBarNameStr`)

Scrollbar creation function called by the scrollbar constructor.

See wxScrollBar() for details.

virtual int wxScrollBar::GetPageSize ( ) const `[virtual]`

Returns the page size of the scrollbar.

This is the number of scroll units that will be scrolled when the user pages up or down. Often it is the same as the thumb size.

See also

SetScrollbar()

virtual int wxScrollBar::GetRange ( ) const `[virtual]`

Returns the length of the scrollbar.

See also

SetScrollbar()

virtual int wxScrollBar::GetThumbPosition ( ) const `[virtual]`

Returns the current position of the scrollbar thumb.

See also

SetThumbPosition()

virtual int wxScrollBar::GetThumbSize ( ) const `[virtual]`

Returns the thumb or 'view' size.

See also

SetScrollbar()

bool wxScrollBar::IsVertical ( ) const

Returns true for scrollbars that have the vertical style set.

virtual void wxScrollBar::SetScrollbar (int *position,* int *thumbSize,* int *range,* int *pageSize,* bool *refresh* = `true`) `[virtual]`

Sets the scrollbar properties.

Parameters

position	The position of the scrollbar in scroll units.
thumbSize	The size of the thumb, or visible portion of the scrollbar, in scroll units.
range	The maximum position of the scrollbar.
pageSize	The size of the page size in scroll units. This is the number of units the scrollbar will scroll when it is paged up or down. Often it is the same as the thumb size.
refresh	true to redraw the scrollbar, false otherwise.

Remarks

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time. You would use:

```
scrollbar->SetScrollbar(0, 16, 50, 15);
```

The page size is 1 less than the thumb size so that the last line of the previous page will be visible on the next page, to help orient the user. Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34. You can determine how many lines are currently visible by dividing the current view size by the character height in pixels. When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and SetScrollbar() call into a function named AdjustScrollbars, which can be called initially and also from a wxSizeEvent event handler function.

Reimplemented from wxWindow.

virtual void wxScrollBar::SetThumbPosition (int *viewStart*) `[virtual]`

Sets the position of the scrollbar.

Parameters

viewStart	The position of the scrollbar thumb.

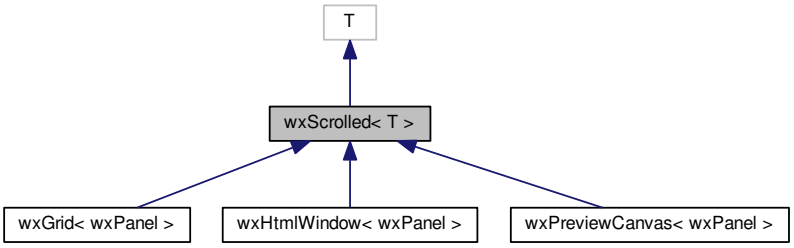
See also

GetThumbPosition()

21.669 wxScrolled< T > Class Template Reference

`#include <wx/scrolwin.h>`

Inheritance diagram for wxScrolled< T >:



21.669.1 Detailed Description

template<class T>class wxScrolled< T >

The wxScrolled class manages scrolling for its client area, transforming the coordinates according to the scrollbar positions, and setting the scroll positions, thumb sizes and ranges according to the area in view.

There are two commonly used (but not the only possible!) specializations of this class:

- wxScrolledWindow, aka wxScrolled<wxPanel>, is equivalent to wxScrolledWindow from earlier versions. Derived from wxPanel, it shares wxPanel's behaviour with regard to TAB traversal and focus handling. Use this if the scrolled window will have child controls.
- wxScrolledCanvas, aka wxScrolled<wxWindow>, derives from wxWindow and so doesn't handle children specially. This is suitable e.g. for implementing scrollable controls such as tree or list controls.

Starting from version 2.4 of wxWidgets, there are several ways to use a wxScrolledWindow (and now wxScrolled). In particular, there are three ways to set the size of the scrolling area:

One way is to set the scrollbars directly using a call to SetScrollbars(). This is the way it used to be in any previous version of wxWidgets and it will be kept for backwards compatibility.

An additional method of manual control, which requires a little less computation of your own, is to set the total size of the scrolling area by calling either wxWindow::SetVirtualSize(), or wxWindow::FitInside(), and setting the scrolling increments for it by calling SetScrollRate(). Scrolling in some orientation is enabled by setting a non-zero increment for it.

The most automatic and newest way is to simply let sizers determine the scrolling area. This is now the default when you set an interior sizer into a wxScrolled with wxWindow::SetSizer(). The scrolling area will be set to the size requested by the sizer and the scrollbars will be assigned for each orientation according to the need for them and the scrolling increment set by SetScrollRate(). As above, scrolling is only enabled in orientations with a non-zero

increment. You can influence the minimum size of the scrolled area controlled by a sizer by calling wxWindow::↩ SetVirtualSizeHints(). (Calling SetScrollbars() has analogous effects in wxWidgets 2.4 – in later versions it may not continue to override the sizer.)

Note that if maximum size hints are still supported by wxWindow::SetVirtualSizeHints(), use them at your own dire risk. They may or may not have been removed for 2.4, but it really only makes sense to set minimum size hints here. We should probably replace wxWindow::SetVirtualSizeHints() with wxWindow::SetMinVirtualSize() or similar and remove it entirely in future.

Todo review docs for this class replacing SetVirtualSizeHints() with SetMinClientSize().

As with all windows, an application can draw onto a wxScrolled using a device context.

You have the option of handling the OnPaint handler or overriding the wxScrolled::OnDraw() function, which is passed a pre-scrolled device context (prepared by wxScrolled::DoPrepareDC()).

If you don't wish to calculate your own scrolling, you must call DoPrepareDC() when not drawing from within On↩ Draw(), to set the device origin for the device context according to the current scroll position.

A wxScrolled will normally scroll itself and therefore its child windows as well. It might however be desired to scroll a different window than itself: e.g. when designing a spreadsheet, you will normally only have to scroll the (usually white) cell area, whereas the (usually grey) label area will scroll very differently. For this special purpose, you can call SetTargetWindow() which means that pressing the scrollbars will scroll a different window.

Note that the underlying system knows nothing about scrolling coordinates, so that all system functions (mouse events, expose events, refresh calls etc) as well as the position of subwindows are relative to the "physical" origin of the scrolled window. If the user insert a child window at position (10,10) and scrolls the window down 100 pixels (moving the child window out of the visible area), the child window will report a position of (10,-90).

Styles

This class supports the following styles:

- wxHSCROLL: If this style is specified and wxVSCROLL isn't, the window will be scrollable only in horizontal direction (by default, i.e. if neither this style nor wxVSCROLL is specified, it scrolls in both directions).
- wxVSCROLL: If this style is specified and wxHSCROLL isn't, the window will be scrollable only in vertical direction (by default, i.e. if neither this style nor wxHSCROLL is specified, it scrolls in both directions).
- wxALWAYS_SHOW_SB: Since wxWidgets 2.9.5, specifying this style makes the window always show its scrollbars, even if they are not used. See ShowScrollbars().
- wxRETAINED: Uses a backing pixmap to speed refreshes. Motif only.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxScrollWinEvent& event)

Event macros for events emitted by this class:

- EVT_SCROLLWIN(func): Process all scroll events.
- EVT_SCROLLWIN_TOP(func): Process `wxEVT_SCROLLWIN_TOP` scroll-to-top events.
- EVT_SCROLLWIN_BOTTOM(func): Process `wxEVT_SCROLLWIN_BOTTOM` scroll-to-bottom events.
- EVT_SCROLLWIN_LINEUP(func): Process `wxEVT_SCROLLWIN_LINEUP` line up events.
- EVT_SCROLLWIN_LINEDOWN(func): Process `wxEVT_SCROLLWIN_LINEDOWN` line down events.
- EVT_SCROLLWIN_PAGEUP(func): Process `wxEVT_SCROLLWIN_PAGEUP` page up events.

- EVT_SCROLLWIN_PAGEDOWN(func): Process `wxEVT_SCROLLWIN_PAGEDOWN` page down events.
- EVT_SCROLLWIN_THUMBTRACK(func): Process `wxEVT_SCROLLWIN_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_SCROLLWIN_THUMBRELEASE(func): Process `wxEVT_SCROLLWIN_THUMBRELEASE` thumb release events.

Note

Don't confuse wxScrollWinEvents generated by this class with wxScrollEvent objects generated by wxScrollBar and wxSlider.

Remarks

Use wxScrolled for applications where the user scrolls by a fixed amount, and where a 'page' can be interpreted to be the current visible portion of the window. For more sophisticated applications, use the wxScrolled implementation as a guide to build your own scroll behaviour or use wxVScrolledWindow or its variants.

Since

The wxScrolled template exists since version 2.9.0. In older versions, only wxScrolledWindow (equivalent of wxScrolled<wxPanel>) was available.

Library: wxCore

Category: Miscellaneous Windows

See also

wxScrollBar, wxClientDC, wxPaintDC, wxVScrolledWindow, wxHScrolledWindow, wxHVScrolledWindow,

Public Member Functions

- wxScrolled ()

  Default constructor.
- wxScrolled (wxWindow *parent, wxWindowID id=-1, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHSCROLL|wxVSCROLL, const wxString &name="scrolledWindow")

  Constructor.
- void CalcScrolledPosition (int x, int y, int *xx, int *yy) const

  Translates the logical coordinates to the device ones.
- wxPoint CalcScrolledPosition (const wxPoint &pt) const
- void CalcUnscrolledPosition (int x, int y, int *xx, int *yy) const

  Translates the device coordinates to the logical ones.
- wxPoint CalcUnscrolledPosition (const wxPoint &pt) const
- bool Create (wxWindow *parent, wxWindowID id=-1, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxHSCROLL|wxVSCROLL, const wxString &name="scrolledWindow")

  Creates the window for two-step construction.
- void DisableKeyboardScrolling ()

  Disable use of keyboard keys for scrolling.
- void DoPrepareDC (wxDC &dc)

  Call this function to prepare the device context for drawing a scrolled image.
- void EnableScrolling (bool xScrolling, bool yScrolling)

Enable or disable use of wxWindow::ScrollWindow() for scrolling.

- void ShowScrollbars (wxScrollbarVisibility horz, wxScrollbarVisibility vert)

  Set the scrollbar visibility.
- void GetScrollPixelsPerUnit (int *xUnit, int *yUnit) const

  Get the number of pixels per scroll unit (line), in each direction, as set by SetScrollbars().
- void GetViewStart (int *x, int *y) const

  Get the position at which the visible portion of the window starts.
- wxPoint GetViewStart () const

  This is a simple overload of GetViewStart(int,int*); see that function for more info.*
- void GetVirtualSize (int *x, int *y) const

  Gets the size in device units of the scrollable window area (as opposed to the client size, which is the area of the window currently visible).
- bool IsRetained () const

  Motif only: true if the window has a backing bitmap.
- virtual void OnDraw (wxDC &dc)

  Called by the default paint event handler to allow the application to define painting behaviour without having to worry about calling DoPrepareDC().
- void PrepareDC (wxDC &dc)

  This function is for backwards compatibility only and simply calls DoPrepareDC() now.
- void Scroll (int x, int y)

  Scrolls a window so the view start is at the given point.
- void Scroll (const wxPoint &pt)

  This is an overload of Scroll(int,int); see that function for more info.
- void SetScrollRate (int xstep, int ystep)

  Set the horizontal and vertical scrolling increment only.
- void SetScrollbars (int pixelsPerUnitX, int pixelsPerUnitY, int noUnitsX, int noUnitsY, int xPos=0, int yPos=0, bool noRefresh=false)

  Sets up vertical and/or horizontal scrollbars.
- void SetTargetWindow (wxWindow *window)

  Call this function to tell wxScrolled to perform the actual scrolling on a different window (and not on itself).
- wxWindow * GetTargetWindow () const
- void SetTargetRect (const wxRect &rect)
- wxRect GetTargetRect () const
- int GetScrollPageSize (int orient) const
- void SetScrollPageSize (int orient, int pageSize)
- int GetScrollLines (int orient) const
- void SetScale (double xs, double ys)
- double GetScaleX () const
- double GetScaleY () const
- virtual void AdjustScrollbars ()
- bool IsAutoScrolling () const

  Are we generating the autoscroll events?
- void StopAutoScrolling ()

  Stop generating the scroll events when mouse is held outside the window.
- virtual bool SendAutoScrollEvents (wxScrollWinEvent &event) const

  This method can be overridden in a derived class to forbid sending the auto scroll events - note that unlike StopAutoScrolling() it doesn't stop the timer, so it will be called repeatedly and will typically return different values depending on the current mouse position.

Protected Member Functions

- virtual wxSize GetSizeAvailableForScrollTarget (const wxSize &size)

  Function which must be overridden to implement the size available for the scroll target for the given size of the main window.

21.669.2 Constructor & Destructor Documentation

template<class T> wxScrolled< T >::wxScrolled ( )

Default constructor.

template<class T> wxScrolled< T >::wxScrolled (wxWindow * *parent*, wxWindowID *id* = -1, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHSCROLL|wxVSCROLL, const wxString & *name* = "scrolledWindow")

Constructor.

Parameters

parent	Parent window.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If a position of wxDefaultPosition is specified then a default position is chosen.
size	Window size. If a size of wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxScrolled.
name	Window name.

Remarks

The window is initially created without visible scrollbars. Call SetScrollbars() to specify how big the virtual window size should be.

21.669.3 Member Function Documentation

template<class T> virtual void wxScrolled< T >::AdjustScrollbars ( ) [virtual]

template<class T> void wxScrolled< T >::CalcScrolledPosition (int *x*, int *y*, int * *xx*, int * *yy*) const

Translates the logical coordinates to the device ones.

For example, if a window is scrolled 10 pixels to the bottom, the device coordinates of the origin are (0, 0) (as always), but the logical coordinates are (0, 10) and so the call to CalcScrolledPosition(0, 10, xx, yy) will return 0 in yy.

wxPerl Note: In wxPerl this method takes two parameters and returns a 2-element list (xx, yy).

See also

CalcUnscrolledPosition()

template<class T> wxPoint wxScrolled< T >::CalcScrolledPosition (const wxPoint & *pt*) const

template<class T> void wxScrolled< T >::CalcUnscrolledPosition (int *x*, int *y*, int * *xx*, int * *yy*) const

Translates the device coordinates to the logical ones.

For example, if a window is scrolled 10 pixels to the bottom, the device coordinates of the origin are (0, 0) (as always), but the logical coordinates are (0, 10) and so the call to CalcUnscrolledPosition(0, 0, xx, yy) will return 10 in yy.

wxPerl Note: In wxPerl this method takes two parameters and returns a 2-element list (xx, yy).

See also

CalcScrolledPosition()

template<class T > wxPoint wxScrolled< T >::CalcUnscrolledPosition (const wxPoint & *pt*) const

template<class T > bool wxScrolled< T >::Create (wxWindow * *parent*, wxWindowID *id* = -1, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxHSCROLL|wxVSCROLL, const wxString & *name* = "scrolledWindow")

Creates the window for two-step construction.

Derived classes should call or replace this function. See wxScrolled::wxScrolled() for details.

template<class T > void wxScrolled< T >::DisableKeyboardScrolling ( )

Disable use of keyboard keys for scrolling.

By default cursor movement keys (including Home, End, Page Up and Down) are used to scroll the window appropriately. If the derived class uses these keys for something else, e.g. changing the currently selected item, this function can be used to disable this behaviour as it's not only not necessary then but can actually be actively harmful if another object forwards a keyboard event corresponding to one of the above keys to us using ProcessWindowEvent() because the event will always be processed which can be undesirable.

Since

2.9.1

template<class T > void wxScrolled< T >::DoPrepareDC (wxDC & *dc*)

Call this function to prepare the device context for drawing a scrolled image.

It sets the device origin according to the current scroll position. DoPrepareDC() is called automatically within the default wxEVT_PAINT event handler, so your OnDraw() override will be passed an already 'pre-scrolled' device context. However, if you wish to draw from outside of OnDraw() (e.g. from your own wxEVT_PAINT handler), you must call this function yourself.

For example:

```
void MyWindow::OnEvent(wxMouseEvent& event)
{
  wxClientDC dc(this);
  DoPrepareDC(dc);

  dc.SetPen(*wxBLACK_PEN);
  float x, y;
  event.Position(&x, &y);
  if (xpos > -1 && ypos > -1 && event.Dragging())
  {
    dc.DrawLine(xpos, ypos, x, y);
  }
  xpos = x;
  ypos = y;
}
```

Notice that the function sets the origin by moving it relatively to the current origin position, so you shouldn't change the origin before calling DoPrepareDC() or, if you do, reset it to (0, 0) later. If you call DoPrepareDC() immediately after device context creation, as in the example above, this problem doesn't arise, of course, so it is customary to do it like this.

template<class T > void wxScrolled< T >::EnableScrolling (bool *xScrolling*, bool *yScrolling*)

Enable or disable use of wxWindow::ScrollWindow() for scrolling.

By default, when a scrolled window is logically scrolled, wxWindow::ScrollWindow() is called on the underlying window which scrolls the window contents and only invalidates the part of the window newly brought into view. If false is passed as an argument, then this "physical scrolling" is disabled and the window is entirely invalidated whenever it is scrolled by calling wxWindow::Refresh().

It should be rarely necessary to disable physical scrolling, so this method shouldn't be called in normal circumstances.

Parameters

xScrolling	If true, enables physical scrolling in the x direction.
yScrolling	If true, enables physical scrolling in the y direction.

template<class T > double wxScrolled< T >::GetScaleX ( ) const

template<class T > double wxScrolled< T >::GetScaleY ( ) const

template<class T > int wxScrolled< T >::GetScrollLines (int *orient*) const

template<class T > int wxScrolled< T >::GetScrollPageSize (int *orient*) const

template<class T > void wxScrolled< T >::GetScrollPixelsPerUnit (int * *xUnit*, int * *yUnit*) const

Get the number of pixels per scroll unit (line), in each direction, as set by SetScrollbars().

A value of zero indicates no scrolling in that direction.

Parameters

xUnit	Receives the number of pixels per horizontal unit.
yUnit	Receives the number of pixels per vertical unit.

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (xUnit, yUnit).

See also

SetScrollbars(), GetVirtualSize()

template<class T > virtual wxSize wxScrolled< T >::GetSizeAvailableForScrollTarget (const wxSize & *size*) [protected], [virtual]

Function which must be overridden to implement the size available for the scroll target for the given size of the main window.

This method must be overridden if SetTargetWindow() is used (it is never called otherwise). The implementation should decrease the *size* to account for the size of the non-scrollable parts of the main window and return only the size available for the scrollable window itself. E.g. in the example given in SetTargetWindow() documentation the function would subtract the height of the header window from the vertical component of *size*.

template<class T > wxRect wxScrolled< T >::GetTargetRect ( ) const

template<class T > wxWindow* wxScrolled< T >::GetTargetWindow ( ) const

template<class T > void wxScrolled< T >::GetViewStart (int * *x*, int * *y*) const

Get the position at which the visible portion of the window starts.

Parameters

x	Receives the first visible x position in scroll units.
y	Receives the first visible y position in scroll units.

Remarks

If either of the scrollbars is not at the home position, *x* and/or *y* will be greater than zero. Combined with wxWindow::GetClientSize(), the application can use this function to efficiently redraw only the visible portion of the window. The positions are in logical scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment.

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (x, y).

See also

SetScrollbars(), Scroll()

template<class T > wxPoint wxScrolled< T >::GetViewStart ( ) const

This is a simple overload of GetViewStart(int*,int*); see that function for more info.

template<class T > void wxScrolled< T >::GetVirtualSize (int * *x*, int * *y*) const

Gets the size in device units of the scrollable window area (as opposed to the client size, which is the area of the window currently visible).

Parameters

x	Receives the length of the scrollable window, in pixels.
y	Receives the height of the scrollable window, in pixels.

Remarks

Use wxDC::DeviceToLogicalX() and wxDC::DeviceToLogicalY() to translate these units to logical units.

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (xUnit, yUnit).

See also

SetScrollbars(), GetScrollPixelsPerUnit()

template<class T > bool wxScrolled< T >::IsAutoScrolling ( ) const

Are we generating the autoscroll events?

template<class T > bool wxScrolled< T >::IsRetained ( ) const

Motif only: true if the window has a backing bitmap.

template<class T > virtual void wxScrolled< T >::OnDraw (wxDC & *dc*) [virtual]

Called by the default paint event handler to allow the application to define painting behaviour without having to worry about calling DoPrepareDC().

Instead of overriding this function you may also just process the paint event in the derived class as usual, but then you will have to call DoPrepareDC() yourself.

template<class T > void wxScrolled< T >::PrepareDC (wxDC & *dc*)

This function is for backwards compatibility only and simply calls DoPrepareDC() now.

Notice that it is not called by the default paint event handle (DoPrepareDC() is), so overriding this method in your derived class is useless.

template<class T > void wxScrolled< T >::Scroll (int *x*, int *y*)

Scrolls a window so the view start is at the given point.

Parameters

x	The x position to scroll to, in scroll units.
y	The y position to scroll to, in scroll units.

Remarks

The positions are in scroll units, not pixels, so to convert to pixels you will have to multiply by the number of pixels per scroll increment. If either parameter is wxDefaultCoord (-1), that position will be ignored (no change in that direction).

See also

SetScrollbars(), GetScrollPixelsPerUnit()

template<class T > void wxScrolled< T >::Scroll (const wxPoint & *pt*)

This is an overload of Scroll(int,int); see that function for more info.

template<class T > virtual bool wxScrolled< T >::SendAutoScrollEvents (wxScrollWinEvent & *event*) const `[virtual]`

This method can be overridden in a derived class to forbid sending the auto scroll events - note that unlike StopAutoScrolling() it doesn't stop the timer, so it will be called repeatedly and will typically return different values depending on the current mouse position.

The base class version just returns true.

template<class T > void wxScrolled< T >::SetScale (double *xs*, double *ys*)

template<class T > void wxScrolled< T >::SetScrollbars (int *pixelsPerUnitX*, int *pixelsPerUnitY*, int *noUnitsX*, int *noUnitsY*, int *xPos* = `0`, int *yPos* = `0`, bool *noRefresh* = `false`)

Sets up vertical and/or horizontal scrollbars.

The first pair of parameters give the number of pixels per 'scroll step', i.e. amount moved when the up or down scroll arrows are pressed. The second pair gives the length of scrollbar in scroll steps, which sets the size of the virtual window.

xPos and *yPos* optionally specify a position to scroll to immediately.

For example, the following gives a window horizontal and vertical scrollbars with 20 pixels per scroll step, and a size of 50 steps (1000 pixels) in each direction:

```
window->SetScrollbars(20, 20, 50, 50);
```

wxScrolled manages the page size itself, using the current client window size as the page size.

Note that for more sophisticated scrolling applications, for example where scroll steps may be variable according to the position in the document, it will be necessary to derive a new class from wxWindow, overriding OnSize() and adjusting the scrollbars appropriately.

Parameters

pixelsPerUnitX	Pixels per scroll unit in the horizontal direction.
pixelsPerUnitY	Pixels per scroll unit in the vertical direction.
noUnitsX	Number of units in the horizontal direction.
noUnitsY	Number of units in the vertical direction.
xPos	Position to initialize the scrollbars in the horizontal direction, in scroll units.
yPos	Position to initialize the scrollbars in the vertical direction, in scroll units.
noRefresh	Will not refresh window if true.

See also

wxWindow::SetVirtualSize()

template<class T > void wxScrolled< T >::SetScrollPageSize (int *orient,* int *pageSize*)

template<class T > void wxScrolled< T >::SetScrollRate (int *xstep,* int *ystep*)

Set the horizontal and vertical scrolling increment only.

See the pixelsPerUnit parameter in SetScrollbars().

template<class T > void wxScrolled< T >::SetTargetRect (const wxRect & *rect*)

template<class T > void wxScrolled< T >::SetTargetWindow (wxWindow * *window*)

Call this function to tell wxScrolled to perform the actual scrolling on a different window (and not on itself).

This method is useful when only a part of the window should be scrolled. A typical example is a control consisting of a fixed header and the scrollable contents window: the scrollbars are attached to the main window itself, hence it, and not the contents window must be derived from wxScrolled, but only the contents window scrolls when the scrollbars are used. To implement such setup, you need to call this method with the contents window as argument.

Notice that if this method is used, GetSizeAvailableForScrollTarget() method must be overridden.

template<class T > void wxScrolled< T >::ShowScrollbars (wxScrollbarVisibility *horz,* wxScrollbarVisibility *vert*)

Set the scrollbar visibility.

By default the scrollbar in the corresponding direction is only shown if it is needed, i.e. if the virtual size of the scrolled window in this direction is greater than the current physical window size. Using this function the scrollbar visibility can be changed to be:

- wxSHOW_SB_ALWAYS: To always show the scrollbar, even if it is not needed currently (wxALWAYS_SHOW_SB style can be used during the window creation to achieve the same effect but it applies in both directions).
- wxSHOW_SB_NEVER: To never show the scrollbar at all. In this case the program should presumably provide some other way for the user to scroll the window.
- wxSHOW_SB_DEFAULT: To restore the default behaviour described above.

Parameters

horz	The desired visibility for the horizontal scrollbar.
vert	The desired visibility for the vertical scrollbar.

Since

2.9.0

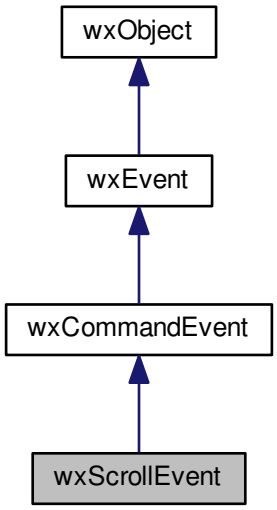
template<class T > void wxScrolled< T >::StopAutoScrolling ( )

Stop generating the scroll events when mouse is held outside the window.

21.670 wxScrollEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxScrollEvent:



21.670.1 Detailed Description

A scroll event holds information about events sent from stand-alone scrollbars (see wxScrollBar) and sliders (see wxSlider).

Note that scrolled windows send the wxScrollWinEvent which does not derive from wxCommandEvent, but from wxEvent directly - don't confuse these two kinds of events and use the event table macros mentioned below only for the scrollbar-like controls.

21.670.2 The difference between EVT_SCROLL_THUMBRELEASE and EVT_SCROLL_CHANGED

The EVT_SCROLL_THUMBRELEASE event is only emitted when actually dragging the thumb using the mouse and releasing it (This EVT_SCROLL_THUMBRELEASE event is also followed by an EVT_SCROLL_CHANGED event).

The EVT_SCROLL_CHANGED event also occurs when using the keyboard to change the thumb position, and when clicking next to the thumb (In all these cases the EVT_SCROLL_THUMBRELEASE event does not happen).

In short, the EVT_SCROLL_CHANGED event is triggered when scrolling/ moving has finished independently of the way it had started. Please see the widgets sample ("Slider" page) to see the difference between EVT_SCROLL_↩ THUMBRELEASE and EVT_SCROLL_CHANGED in action.

Remarks

Note that unless specifying a scroll control identifier, you will need to test for scrollbar orientation with wx↩ ScrollEvent::GetOrientation, since horizontal and vertical scroll events are processed using the same event handler.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxScrollEvent& event)

Event macros: You can use EVT_COMMAND_SCROLL... macros with window IDs for when intercepting scroll events from controls, or EVT_SCROLL... macros without window IDs for intercepting scroll events from the receiving window – except for this, the macros behave exactly the same.

- EVT_SCROLL(func): Process all scroll events.
- EVT_SCROLL_TOP(func): Process `wxEVT_SCROLL_TOP` scroll-to-top events (minimum position).
- EVT_SCROLL_BOTTOM(func): Process `wxEVT_SCROLL_BOTTOM` scroll-to-bottom events (maximum position).
- EVT_SCROLL_LINEUP(func): Process `wxEVT_SCROLL_LINEUP` line up events.
- EVT_SCROLL_LINEDOWN(func): Process `wxEVT_SCROLL_LINEDOWN` line down events.
- EVT_SCROLL_PAGEUP(func): Process `wxEVT_SCROLL_PAGEUP` page up events.
- EVT_SCROLL_PAGEDOWN(func): Process `wxEVT_SCROLL_PAGEDOWN` page down events.
- EVT_SCROLL_THUMBTRACK(func): Process `wxEVT_SCROLL_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_SCROLL_THUMBRELEASE(func): Process `wxEVT_SCROLL_THUMBRELEASE` thumb release events.
- EVT_SCROLL_CHANGED(func): Process `wxEVT_SCROLL_CHANGED` end of scrolling events (MSW only).
- EVT_COMMAND_SCROLL(id, func): Process all scroll events.
- EVT_COMMAND_SCROLL_TOP(id, func): Process `wxEVT_SCROLL_TOP` scroll-to-top events (minimum position).
- EVT_COMMAND_SCROLL_BOTTOM(id, func): Process `wxEVT_SCROLL_BOTTOM` scroll-to-bottom events (maximum position).
- EVT_COMMAND_SCROLL_LINEUP(id, func): Process `wxEVT_SCROLL_LINEUP` line up events.
- EVT_COMMAND_SCROLL_LINEDOWN(id, func): Process `wxEVT_SCROLL_LINEDOWN` line down events.

- EVT_COMMAND_SCROLL_PAGEUP(id, func): Process `wxEVT_SCROLL_PAGEUP` page up events.
- EVT_COMMAND_SCROLL_PAGEDOWN(id, func): Process `wxEVT_SCROLL_PAGEDOWN` page down events.
- EVT_COMMAND_SCROLL_THUMBTRACK(id, func): Process `wxEVT_SCROLL_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_COMMAND_SCROLL_THUMBRELEASE(func): Process `wxEVT_SCROLL_THUMBRELEASE` thumb release events.
- EVT_COMMAND_SCROLL_CHANGED(func): Process `wxEVT_SCROLL_CHANGED` end of scrolling events (MSW only).

Library: wxCore

Category: Events

See also

wxScrollBar, wxSlider, wxSpinButton, wxScrollWinEvent, Events and Event Handling

Public Member Functions

- wxScrollEvent (wxEventType commandType=wxEVT_NULL, int id=0, int pos=0, int orientation=0)

  Constructor.
- int GetOrientation () const

  Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.
- int GetPosition () const

  Returns the position of the scrollbar.
- void SetOrientation (int orient)
- void SetPosition (int pos)

Additional Inherited Members

21.670.3 Constructor & Destructor Documentation

wxScrollEvent::wxScrollEvent (wxEventType *commandType* = wxEVT_NULL, int *id* = 0, int *pos* = 0, int *orientation* = 0)

Constructor.

21.670.4 Member Function Documentation

int wxScrollEvent::GetOrientation ( ) const

Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.

int wxScrollEvent::GetPosition ( ) const

Returns the position of the scrollbar.

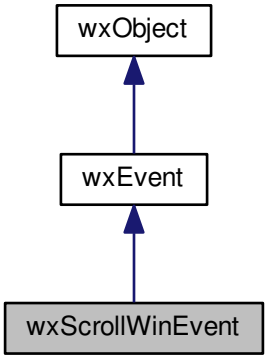
void wxScrollEvent::SetOrientation (int *orient*)

void wxScrollEvent::SetPosition (int *pos*)

21.671 wxScrollWinEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxScrollWinEvent:



21.671.1 Detailed Description

A scroll event holds information about events sent from scrolling windows.

Note that you can use the EVT_SCROLLWIN∗ macros for intercepting scroll window events from the receiving window.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxScrollWinEvent& event)

Event macros:

- EVT_SCROLLWIN(func): Process all scroll events.
- EVT_SCROLLWIN_TOP(func): Process `wxEVT_SCROLLWIN_TOP` scroll-to-top events.
- EVT_SCROLLWIN_BOTTOM(func): Process `wxEVT_SCROLLWIN_BOTTOM` scroll-to-bottom events.
- EVT_SCROLLWIN_LINEUP(func): Process `wxEVT_SCROLLWIN_LINEUP` line up events.
- EVT_SCROLLWIN_LINEDOWN(func): Process `wxEVT_SCROLLWIN_LINEDOWN` line down events.
- EVT_SCROLLWIN_PAGEUP(func): Process `wxEVT_SCROLLWIN_PAGEUP` page up events.
- EVT_SCROLLWIN_PAGEDOWN(func): Process `wxEVT_SCROLLWIN_PAGEDOWN` page down events.
- EVT_SCROLLWIN_THUMBTRACK(func): Process `wxEVT_SCROLLWIN_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).

- EVT_SCROLLWIN_THUMBRELEASE(func): Process `wxEVT_SCROLLWIN_THUMBRELEASE` thumb release events.

Library: wxCore

Category: Events

See also

wxScrollEvent, Events and Event Handling

Public Member Functions

- wxScrollWinEvent (wxEventType commandType=wxEVT_NULL, int pos=0, int orientation=0)

  Constructor.
- int GetOrientation () const

  Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.
- int GetPosition () const

  Returns the position of the scrollbar for the thumb track and release events.
- void SetOrientation (int orient)
- void SetPosition (int pos)

Additional Inherited Members

21.671.2 Constructor & Destructor Documentation

wxScrollWinEvent::wxScrollWinEvent (wxEventType *commandType* = wxEVT_NULL, int *pos* = 0, int *orientation* = 0)

Constructor.

21.671.3 Member Function Documentation

int wxScrollWinEvent::GetOrientation ( ) const

Returns wxHORIZONTAL or wxVERTICAL, depending on the orientation of the scrollbar.

Todo wxHORIZONTAL and wxVERTICAL should go in their own enum

int wxScrollWinEvent::GetPosition ( ) const

Returns the position of the scrollbar for the thumb track and release events.

Note that this field can't be used for the other events, you need to query the window itself for the current position in that case.

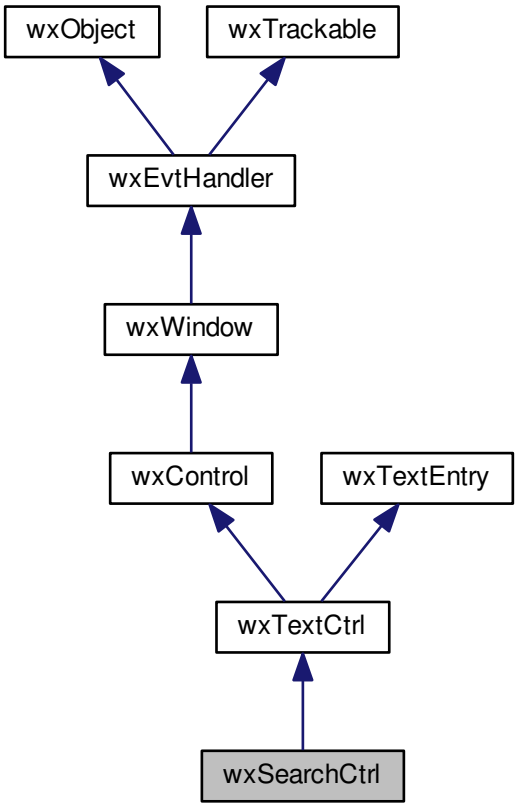
void wxScrollWinEvent::SetOrientation (int *orient*)

void wxScrollWinEvent::SetPosition (int *pos*)

21.672 wxSearchCtrl Class Reference

`#include <wx/srchctrl.h>`

Inheritance diagram for wxSearchCtrl:



21.672.1 Detailed Description

A search control is a composite control with a search button, a text control, and a cancel button.

Styles

This class supports the following styles:

- wxTE_PROCESS_ENTER: The control will generate the event `wxEVT_TEXT_ENTER` (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls).
- wxTE_PROCESS_TAB: The control will receive `wxEVT_CHAR` events for TAB pressed - normally, TAB is used for passing to the next control in a dialog instead. For the control created with this style, you can still use Ctrl-Enter to pass to the next control from the keyboard.
- wxTE_NOHIDESEL: By default, the Windows text control doesn't show the selection when it doesn't have focus - use this style to force it to always show it. It doesn't do anything under other platforms.

- wxTE_LEFT: The text in the control will be left-justified (default).
- wxTE_CENTRE: The text in the control will be centered (currently wxMSW and wxGTK2 only).
- wxTE_RIGHT: The text in the control will be right-justified (currently wxMSW and wxGTK2 only).
- wxTE_CAPITALIZE: On PocketPC and Smartphone, causes the first letter to be capitalized.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class: To retrieve actual search queries, use EVT_TEXT and EVT_TEXT↩ _ENTER events, just as you would with wxTextCtrl.

- EVT_SEARCHCTRL_SEARCH_BTN(id, func): Respond to a `wxEVT_SEARCHCTRL_SEARCH_BT↩ N` event, generated when the search button is clicked. Note that this does not initiate a search on its own, you need to perform the appropriate action in your event handler. You may use

  ```
  event.GetString()
  ```

  to retrieve the string to search for in the event handler code.
- EVT_SEARCHCTRL_CANCEL_BTN(id, func): Respond to a `wxEVT_SEARCHCTRL_CANCEL_BTN` event, generated when the cancel button is clicked.

Library: wxCore

Category: Controls

See also

wxTextCtrl::Create, wxValidator

Public Member Functions

- wxSearchCtrl ()

  Default constructor.
- wxSearchCtrl (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wx↩ Point &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &valida- tor=wxDefaultValidator, const wxString &name=wxSearchCtrlNameStr)

  Constructor, creating and showing a text control.
- virtual ~wxSearchCtrl ()

  Destructor, destroying the search control.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wx↩ Point &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &valida- tor=wxDefaultValidator, const wxString &name=wxSearchCtrlNameStr)
- virtual wxMenu * GetMenu ()

  Returns a pointer to the search control's menu object or NULL if there is no menu attached.
- virtual bool IsSearchButtonVisible () const

  Returns the search button visibility value.
- virtual bool IsCancelButtonVisible () const

  Returns the cancel button's visibility state.
- virtual void SetMenu (wxMenu *menu)

Sets the search control's menu object.

- virtual void ShowCancelButton (bool show)

  Shows or hides the cancel button.

- virtual void ShowSearchButton (bool show)

  Sets the search button visibility value on the search control.

- void SetDescriptiveText (const wxString &text)

  Set the text to be displayed in the search control when the user has not yet typed anything in it.

- wxString GetDescriptiveText () const

  Return the text displayed when there is not yet any user input.

Additional Inherited Members

21.672.2 Constructor & Destructor Documentation

wxSearchCtrl::wxSearchCtrl ( )

Default constructor.

wxSearchCtrl::wxSearchCtrl (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxSearchCtrlNameStr)

Constructor, creating and showing a text control.

Parameters

parent	Parent window. Should not be NULL.
id	Control identifier. A value of -1 denotes a default value.
value	Default text value.
pos	Text control position.
size	Text control size.
style	Window style. See wxSearchCtrl.
validator	Window validator.
name	Window name.

See also

wxTextCtrl::Create, wxValidator

virtual wxSearchCtrl::~wxSearchCtrl ( ) [virtual]

Destructor, destroying the search control.

21.672.3 Member Function Documentation

bool wxSearchCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxSearchCtrlNameStr)

wxString wxSearchCtrl::GetDescriptiveText ( ) const

Return the text displayed when there is not yet any user input.

virtual wxMenu* wxSearchCtrl::GetMenu ( ) [virtual]

Returns a pointer to the search control's menu object or NULL if there is no menu attached.

virtual bool wxSearchCtrl::IsCancelButtonVisible ( ) const [virtual]

Returns the cancel button's visibility state.

virtual bool wxSearchCtrl::IsSearchButtonVisible ( ) const [virtual]

Returns the search button visibility value.

If there is a menu attached, the search button will be visible regardless of the search button visibility value.

This always returns false in Mac OS X v10.3

void wxSearchCtrl::SetDescriptiveText (const wxString & *text*)

Set the text to be displayed in the search control when the user has not yet typed anything in it.

virtual void wxSearchCtrl::SetMenu (wxMenu * *menu*) [virtual]

Sets the search control's menu object.

If there is already a menu associated with the search control it is deleted.

Parameters

menu	Menu to attach to the search control.

virtual void wxSearchCtrl::ShowCancelButton (bool *show*) [virtual]

Shows or hides the cancel button.

virtual void wxSearchCtrl::ShowSearchButton (bool *show*) [virtual]

Sets the search button visibility value on the search control.

If there is a menu attached, the search button will be visible regardless of the search button visibility value.

This has no effect in Mac OS X v10.3

21.673 wxSemaphore Class Reference

`#include <wx/thread.h>`

21.673.1 Detailed Description

wxSemaphore is a counter limiting the number of threads concurrently accessing a shared resource.

This counter is always between 0 and the maximum value specified during the semaphore creation. When the counter is strictly greater than 0, a call to wxSemaphore::Wait() returns immediately and decrements the counter. As soon as it reaches 0, any subsequent calls to wxSemaphore::Wait block and only return when the semaphore counter becomes strictly positive again as the result of calling wxSemaphore::Post which increments the counter.

In general, semaphores are useful to restrict access to a shared resource which can only be accessed by some fixed number of clients at the same time. For example, when modeling a hotel reservation system a semaphore with the counter equal to the total number of available rooms could be created. Each time a room is reserved, the semaphore should be acquired by calling wxSemaphore::Wait and each time a room is freed it should be released by calling wxSemaphore::Post.

Library: wxBase

Category: Threading

Public Member Functions

- wxSemaphore (int initialcount=0, int maxcount=0)

  Specifying a maxcount of 0 actually makes wxSemaphore behave as if there is no upper limit.
- ~wxSemaphore ()

  Destructor is not virtual, don't use this class polymorphically.
- wxSemaError Post ()

  Increments the semaphore count and signals one of the waiting threads in an atomic way.
- wxSemaError TryWait ()

  Same as Wait(), but returns immediately.
- wxSemaError Wait ()

  Wait indefinitely until the semaphore count becomes strictly positive and then decrement it and return.
- wxSemaError WaitTimeout (unsigned long timeout_millis)

  Same as Wait(), but with a timeout limit.

21.673.2 Constructor & Destructor Documentation

wxSemaphore::wxSemaphore (int *initialcount* = 0, int *maxcount* = 0)

Specifying a *maxcount* of 0 actually makes wxSemaphore behave as if there is no upper limit.

If *maxcount* is 1, the semaphore behaves almost as a mutex (but unlike a mutex it can be released by a thread different from the one which acquired it).

initialcount is the initial value of the semaphore which must be between 0 and *maxcount* (if it is not set to 0).

wxSemaphore::~wxSemaphore ( )

Destructor is not virtual, don't use this class polymorphically.

21.673.3 Member Function Documentation

wxSemaError wxSemaphore::Post ( )

Increments the semaphore count and signals one of the waiting threads in an atomic way.

Returns *wxSEMA_OVERFLOW* if the count would increase the counter past the maximum.

Returns

One of:

- wxSEMA_NO_ERROR: There was no error.
- wxSEMA_INVALID : Semaphore hasn't been initialized successfully.
- wxSEMA_OVERFLOW: Post() would increase counter past the max.
- wxSEMA_MISC_ERROR: Miscellaneous error.

wxSemaError wxSemaphore::TryWait ( )

Same as Wait(), but returns immediately.

Returns

One of:

- wxSEMA_NO_ERROR: There was no error.
- wxSEMA_INVALID: Semaphore hasn't been initialized successfully.
- wxSEMA_BUSY: Returned by TryWait() if Wait() would block, i.e. the count is zero.
- wxSEMA_MISC_ERROR: Miscellaneous error.

wxSemaError wxSemaphore::Wait ( )

Wait indefinitely until the semaphore count becomes strictly positive and then decrement it and return.

Returns

One of:

- wxSEMA_NO_ERROR: There was no error.
- wxSEMA_INVALID: Semaphore hasn't been initialized successfully.
- wxSEMA_MISC_ERROR: Miscellaneous error.

wxSemaError wxSemaphore::WaitTimeout (unsigned long *timeout_millis*)

Same as Wait(), but with a timeout limit.

Returns

One of:

- wxSEMA_NO_ERROR: There was no error.
- wxSEMA_INVALID: Semaphore hasn't been initialized successfully.
- wxSEMA_TIMEOUT: Timeout occurred without receiving semaphore.
- wxSEMA_MISC_ERROR: Miscellaneous error.

21.674 wxServer Class Reference

`#include <wx/ipc.h>`

21.674.1 Detailed Description

A wxServer object represents the server part of a client-server DDE-like (Dynamic Data Exchange) conversation.

The actual DDE-based implementation using wxDDEServer is available on Windows only, but a platform-independent, socket-based version of this API is available using wxTCPServer, which has the same API.

To create a server which can communicate with a suitable client, you need to derive a class from wxConnection and another from wxServer. The custom wxConnection class will intercept communications in a 'conversation' with a client, and the custom wxServer is required so that a user-overridden wxServer::OnAcceptConnection member can return a wxConnection of the required class, when a connection is made. Look at the IPC sample and the Interprocess Communication for an example of how to do this.

Library: wxBase

Category: Interprocess Communication

See also

wxClient, wxConnection, IPC, Interprocess Communication

Public Member Functions

- wxServer ()

  Constructs a server object.
- bool Create (const wxString &service)

  Registers the server using the given service name.
- virtual wxConnectionBase * OnAcceptConnection (const wxString &topic)

  *When a client calls **MakeConnection**, the server receives the message and this member is called.*

21.674.2 Constructor & Destructor Documentation

wxServer::wxServer ( )

Constructs a server object.

21.674.3 Member Function Documentation

bool wxServer::Create (const wxString & *service*)

Registers the server using the given service name.

Under Unix, the service name may be either an integer port identifier in which case an Internet domain socket will be used for the communications, or a valid file name (which shouldn't exist and will be deleted afterwards) in which case a Unix domain socket is created.

false is returned if the call failed (for example, the port number is already in use).

virtual wxConnectionBase* wxServer::OnAcceptConnection (const wxString & *topic*) `[virtual]`

When a client calls **MakeConnection**, the server receives the message and this member is called.

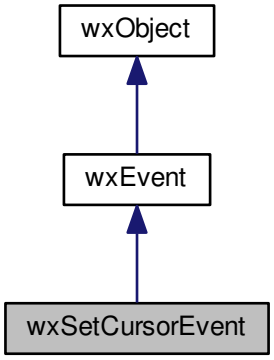
The application should derive a member to intercept this message and return a connection object of either the standard wxConnection type, or (more likely) of a user-derived type.

If the topic is **STDIO**, the application may wish to refuse the connection. Under UNIX, when a server is created the OnAcceptConnection() message is always sent for standard input and output, but in the context of DDE messages it doesn't make a lot of sense.

21.675 wxSetCursorEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxSetCursorEvent:



21.675.1 Detailed Description

A wxSetCursorEvent is generated from wxWindow when the mouse cursor is about to be set as a result of mouse motion.

This event gives the application the chance to perform specific mouse cursor processing based on the current position of the mouse within the window. Use wxSetCursorEvent::SetCursor to specify the cursor you want to be displayed.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSetCursorEvent& event)

Event macros:

- EVT_SET_CURSOR(func): Process a `wxEVT_SET_CURSOR` event.

Library: wxCore

Category: Events

See also

wxSetCursor, wxWindow::SetCursor

Public Member Functions

- wxSetCursorEvent (wxCoord x=0, wxCoord y=0)

  Constructor, used by the library itself internally to initialize the event object.
- const wxCursor & GetCursor () const

  Returns a reference to the cursor specified by this event.
- wxCoord GetX () const

  Returns the X coordinate of the mouse in client coordinates.
- wxCoord GetY () const

  Returns the Y coordinate of the mouse in client coordinates.
- bool HasCursor () const

  Returns true if the cursor specified by this event is a valid cursor.
- void SetCursor (const wxCursor &cursor)

  Sets the cursor associated with this event.

Additional Inherited Members

21.675.2 Constructor & Destructor Documentation

wxSetCursorEvent::wxSetCursorEvent (wxCoord *x* = 0, wxCoord *y* = 0)

Constructor, used by the library itself internally to initialize the event object.

21.675.3 Member Function Documentation

const wxCursor& wxSetCursorEvent::GetCursor ( ) const

Returns a reference to the cursor specified by this event.

wxCoord wxSetCursorEvent::GetX ( ) const

Returns the X coordinate of the mouse in client coordinates.

wxCoord wxSetCursorEvent::GetY ( ) const

Returns the Y coordinate of the mouse in client coordinates.

bool wxSetCursorEvent::HasCursor ( ) const

Returns true if the cursor specified by this event is a valid cursor.

Remarks

You cannot specify wxNullCursor with this event, as it is not considered a valid cursor.

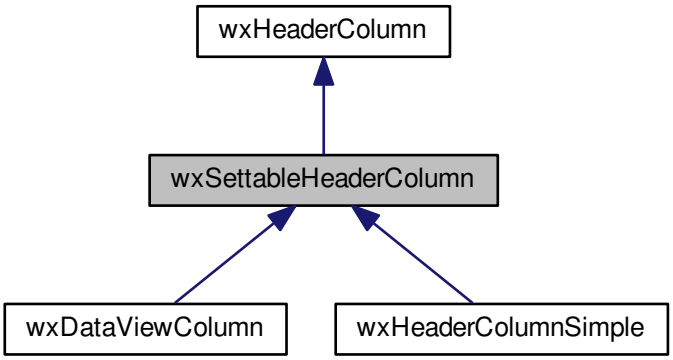
void wxSetCursorEvent::SetCursor (const wxCursor & *cursor*)

Sets the cursor associated with this event.

21.676 wxSettableHeaderColumn Class Reference

`#include <wx/headercol.h>`

Inheritance diagram for wxSettableHeaderColumn:



21.676.1 Detailed Description

Adds methods to set the column attributes to wxHeaderColumn.

This class adds setters for the column attributes defined by wxHeaderColumn. It is still an abstract base class and needs to be implemented before using it with wxHeaderCtrl.

Library: wxCore

Category: Controls

Public Member Functions

- virtual void SetTitle (const wxString &title)=0

  Set the text to display in the column header.
- virtual void SetBitmap (const wxBitmap &bitmap)=0

  Set the bitmap to be displayed in the column header.
- virtual void SetWidth (int width)=0

  Set the column width.
- virtual void SetMinWidth (int minWidth)=0

  Set the minimal column width.
- virtual void SetAlignment (wxAlignment align)=0

  Set the alignment of the column header.
- virtual void SetFlags (int flags)=0

  Set the column flags.
- void ChangeFlag (int flag, bool set)

  Set or clear the given flag.

- void SetFlag (int flag)

  Set the specified flag for the column.
- void ClearFlag (int flag)

  Clear the specified flag for the column.
- void ToggleFlag (int flag)

  Toggle the specified flag for the column.
- virtual void SetResizeable (bool resizable)

  Call this to enable or disable interactive resizing of the column by the user.
- virtual void SetSortable (bool sortable)

  Allow clicking the column to sort the control contents by the field in this column.
- virtual void SetReorderable (bool reorderable)

  Allow changing the column order by dragging it.
- virtual void SetHidden (bool hidden)

  Hide or show the column.
- void UnsetAsSortKey ()

  Don't use this column for sorting.
- virtual void SetSortOrder (bool ascending)=0

  Sets this column as the sort key for the associated control.
- void ToggleSortOrder ()

  Inverses the sort order.

21.676.2 Member Function Documentation

void wxSettableHeaderColumn::ChangeFlag (int *flag*, bool *set*)

Set or clear the given flag.

Parameters

flag	The flag to set or clear.
set	If true, set the flag, i.e. equivalent to calling SetFlag(), otherwise clear it, as ClearFlag().

See also

SetFlags()

void wxSettableHeaderColumn::ClearFlag (int *flag*)

Clear the specified flag for the column.

See also

SetFlags()

virtual void wxSettableHeaderColumn::SetAlignment (wxAlignment *align*) `[pure virtual]`

Set the alignment of the column header.

Parameters

align	The text alignment in horizontal direction only or wxALIGN_NOT to use the default alignment, The possible values here are wxALIGN_CENTRE, wxALIGN_LEFT or wxALIGN_RIGHT with wxALIGN_CENTRE_HORIZONTAL being also supported as synonym for wxALIGN_CEN↩ TRE for consistency (but notice that GetAlignment() never returns it).

Implemented in wxHeaderColumnSimple.

virtual void wxSettableHeaderColumn::SetBitmap (const wxBitmap & *bitmap*) `[pure virtual]`

Set the bitmap to be displayed in the column header.

Notice that the bitmaps displayed in different columns of the same control must all be of the same size.

Implemented in wxHeaderColumnSimple.

void wxSettableHeaderColumn::SetFlag (int *flag*)

Set the specified flag for the column.

See also

SetFlags()

virtual void wxSettableHeaderColumn::SetFlags (int *flags*) `[pure virtual]`

Set the column flags.

This method allows to set all flags at once, see also generic ChangeFlag(), SetFlag(), ClearFlag() and ToggleFlag() methods below as well as specific SetResizeable(), SetSortable(), SetReorderable() and SetHidden() ones.

Parameters

flags	Combination of wxCOL_RESIZABLE, wxCOL_SORTABLE, wxCOL_REORDERABLE and wxCOL_HIDDEN bit flags.

Implemented in wxHeaderColumnSimple.

virtual void wxSettableHeaderColumn::SetHidden (bool *hidden*) `[virtual]`

Hide or show the column.

By default all columns are shown but some of them can be completely hidden from view by calling this function.

Equivalent to ChangeFlag(wxCOL_HIDDEN, hidden).

virtual void wxSettableHeaderColumn::SetMinWidth (int *minWidth*) `[pure virtual]`

Set the minimal column width.

This method can be used with resizable columns (i.e. those for which wxCOL_RESIZABLE flag is set in GetFlags() or, alternatively, IsResizeable() returns true) to prevent the user from making them narrower than the given width.

Parameters

minWidth	The minimal column width in pixels, may be 0 to remove any previously set restrictions.

Implemented in wxHeaderColumnSimple.

virtual void wxSettableHeaderColumn::SetReorderable (bool *reorderable*) `[virtual]`

Allow changing the column order by dragging it.

Equivalent to ChangeFlag(wxCOL_REORDERABLE, reorderable).

virtual void wxSettableHeaderColumn::SetResizeable (bool *resizable*) `[virtual]`

Call this to enable or disable interactive resizing of the column by the user.

By default, the columns are resizable.

Equivalent to ChangeFlag(wxCOL_RESIZABLE, resizable).

virtual void wxSettableHeaderColumn::SetSortable (bool *sortable*) `[virtual]`

Allow clicking the column to sort the control contents by the field in this column.

By default, the columns are not sortable so you need to explicitly call this function to allow sorting by the field corresponding to this column.

Equivalent to ChangeFlag(wxCOL_SORTABLE, sortable).

virtual void wxSettableHeaderColumn::SetSortOrder (bool *ascending*) `[pure virtual]`

Sets this column as the sort key for the associated control.

This function indicates that this column is currently used for sorting the control and also sets the sorting direction. Notice that actual sorting is only done in the control associated with the header, this function doesn't do any sorting on its own.

Don't confuse this function with SetSortable() which should be used to indicate that the column *may* be used for sorting while this one is used to indicate that it currently *is* used for sorting. Of course, SetSortOrder() can be only called for sortable columns.

Parameters

ascending	If true, sort in ascending order, otherwise in descending order.

Implemented in wxHeaderColumnSimple.

virtual void wxSettableHeaderColumn::SetTitle (const wxString & *title*) `[pure virtual]`

Set the text to display in the column header.

Implemented in wxHeaderColumnSimple.

virtual void wxSettableHeaderColumn::SetWidth (int *width*) `[pure virtual]`

Set the column width.

Parameters

width	The column width in pixels or the special wxCOL_WIDTH_DEFAULT (meaning to use default width) or wxCOL_WIDTH_AUTOSIZE (size to fit the content) value.

Implemented in wxHeaderColumnSimple.

void wxSettableHeaderColumn::ToggleFlag (int *flag*)

Toggle the specified flag for the column.

If the flag is currently set, equivalent to ClearFlag(), otherwise – to SetFlag().

See also

SetFlags()

void wxSettableHeaderColumn::ToggleSortOrder ( )

Inverses the sort order.

This function is typically called when the user clicks on a column used for sorting to change sort order from ascending to descending or vice versa.

See also

SetSortOrder(), IsSortOrderAscending()

void wxSettableHeaderColumn::UnsetAsSortKey ( )

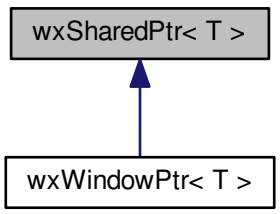
Don't use this column for sorting.

This is the reverse of SetSortOrder() and is called to indicate that this column is not used for sorting any longer.

21.677 wxSharedPtr< T > Class Template Reference

`#include <wx/sharedptr.h>`

Inheritance diagram for wxSharedPtr< T >:



21.677.1 Detailed Description

template<typename T>class wxSharedPtr< T >

A smart pointer with non-intrusive reference counting.

It is modelled after `boost::shared_ptr<>` and can be used with STL containers and wxVector<T> unlike `std::auto_ptr<>` and wxScopedPtr<T>.

Library: wxBase

Category: Smart Pointers

See also

wxScopedPtr<T>, wxWeakRef<T>, wxObjectDataPtr<T>

Public Member Functions

- wxEXPLICIT wxSharedPtr (T *ptr=NULL)

  Constructor.
- template<typename Deleter >
  wxEXPLICIT wxSharedPtr (T *ptr, Deleter d)

  Constructor.
- wxSharedPtr (const wxSharedPtr< T > &tocopy)

  Copy constructor.
- ~wxSharedPtr ()

  Destructor.
- T * get () const

  Returns pointer to its object or NULL.
- operator unspecified_bool_type () const

  Conversion to a boolean expression (in a variant which is not convertible to anything but a boolean expression).
- T operator* () const

  Returns a reference to the object.
- T * operator-> () const

  Smart pointer member access.
- wxSharedPtr< T > & operator= (T *ptr)

  Assignment operator.
- wxSharedPtr< T > & operator= (const wxSharedPtr< T > &tocopy)

  Assignment operator.
- void reset (T *ptr=NULL)

  Reset pointer to ptr.
- template<typename Deleter >
  void reset (T *ptr, Deleter d)

  Reset pointer to ptr.
- bool unique () const

  Returns true if this is the only pointer pointing to its object.
- long use_count () const

  Returns the number of pointers pointing to its object.

21.677.2 Constructor & Destructor Documentation

template<typename T > wxEXPLICIT wxSharedPtr< T >::wxSharedPtr (T * *ptr* = NULL)

Constructor.

Creates shared pointer from the raw pointer *ptr* and takes ownership of it.

template<typename T > template<typename Deleter > wxEXPLICIT wxSharedPtr< T >::wxSharedPtr (T * *ptr*, Deleter *d*)

Constructor.

Creates shared pointer from the raw pointer *ptr* and deleter *d* and takes ownership of it.

Parameters

ptr	The raw pointer.
d	Deleter - a functor that is called instead of delete to free the *ptr* raw pointer when its reference count drops to zero.

Since

3.0

template<typename T > wxSharedPtr< T >::wxSharedPtr (const wxSharedPtr< T > & *tocopy*)

Copy constructor.

template<typename T > wxSharedPtr< T >::~wxSharedPtr ( )

Destructor.

21.677.3 Member Function Documentation

template<typename T > T* wxSharedPtr< T >::get ( ) const

Returns pointer to its object or NULL.

template<typename T > wxSharedPtr< T >::operator unspecified_bool_type ( ) const

Conversion to a boolean expression (in a variant which is not convertible to anything but a boolean expression).

If this class contains a valid pointer it will return true, if it contains a NULL pointer it will return false.

template<typename T > T wxSharedPtr< T >::operator* ( ) const

Returns a reference to the object.

If the internal pointer is NULL this method will cause an assert in debug mode.

template<typename T > T* wxSharedPtr< T >::operator-> ( ) const

Smart pointer member access.

Returns pointer to its object.

If the internal pointer is NULL this method will cause an assert in debug mode.

template<typename T > wxSharedPtr<T>& wxSharedPtr< T >::operator= (T * *ptr*)

Assignment operator.

Releases any previously held pointer and creates a reference to *ptr*.

template<typename T > wxSharedPtr<T>& wxSharedPtr< T >::operator= (const wxSharedPtr< T > & *tocopy*)

Assignment operator.

Releases any previously held pointer and creates a reference to the same object as *topcopy*.

template<typename T > void wxSharedPtr< T >::reset (T * *ptr* = NULL)

Reset pointer to *ptr*.

If the reference count of the previously owned pointer was 1 it will be deleted.

template<typename T > template<typename Deleter > void wxSharedPtr< T >::reset (T * *ptr,* Deleter *d*)

Reset pointer to *ptr*.

If the reference count of the previously owned pointer was 1 it will be deleted.

Parameters

ptr	The new raw pointer.
d	Deleter - a functor that is called instead of delete to free the *ptr* raw pointer when its reference count drops to zero.

Since

3.0

template<typename T > bool wxSharedPtr< T >::unique ( ) const

Returns true if this is the only pointer pointing to its object.

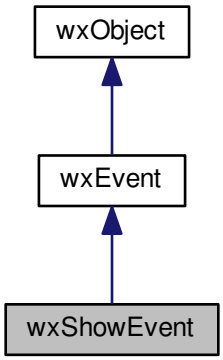
template<typename T > long wxSharedPtr< T >::use_count ( ) const

Returns the number of pointers pointing to its object.

21.678 wxShowEvent Class Reference

```
#include <wx/event.h>
```

Inheritance diagram for wxShowEvent:



21.678.1 Detailed Description

An event being sent when the window is shown or hidden.

The event is triggered by calls to wxWindow::Show(), and any user action showing a previously hidden window or vice versa (if allowed by the current platform and/or window manager). Notice that the event is not triggered when the application is iconized (minimized) or restored under wxMSW.

Availability: only available for the wxMSW, wxGTK ports.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxShowEvent& event)

Event macros:

- EVT_SHOW(func): Process a `wxEVT_SHOW` event.

Library: wxCore

Category: Events

See also

Events and Event Handling, wxWindow::Show, wxWindow::IsShown

Public Member Functions

- wxShowEvent (int winid=0, bool show=false)

  Constructor.
- void SetShow (bool show)

  Set whether the windows was shown or hidden.

- bool IsShown () const

  Return true if the window has been shown, false if it has been hidden.

- bool GetShow () const

Additional Inherited Members

21.678.2 Constructor & Destructor Documentation

wxShowEvent::wxShowEvent (int *winid* = `0`, bool *show* = `false`)

Constructor.

21.678.3 Member Function Documentation

bool wxShowEvent::GetShow ( ) const

Deprecated This function is deprecated in favour of IsShown().

bool wxShowEvent::IsShown ( ) const

Return true if the window has been shown, false if it has been hidden.

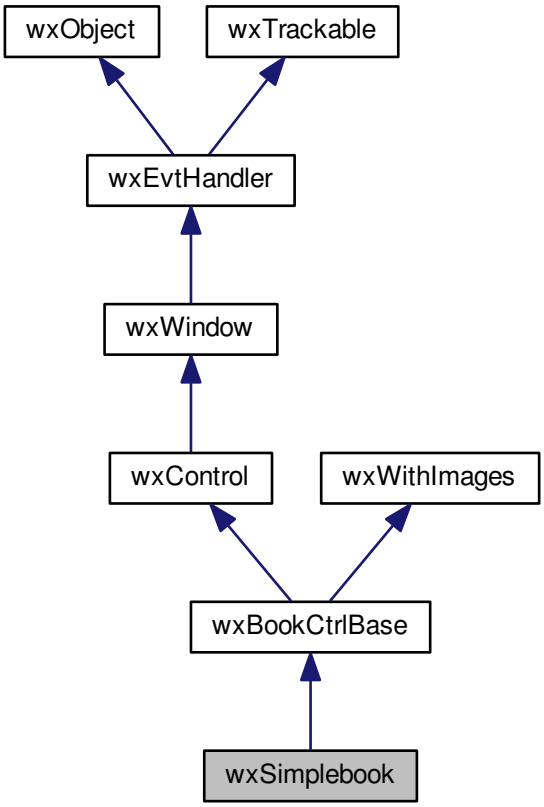
void wxShowEvent::SetShow (bool *show*)

Set whether the windows was shown or hidden.

21.679 wxSimplebook Class Reference

```
#include <wx/simplebook.h>
```

Inheritance diagram for wxSimplebook:



21.679.1 Detailed Description

wxSimplebook is a control showing exactly one of its several pages.

It implements wxBookCtrlBase class interface but doesn't allow the user to change the page being displayed, unlike all the other book control classes, only the program can do it.

This class is created in the same manner as any other wxBookCtrl but then the program will typically call ChangeSelection() to show different pages. See the Notebook Sample for an example of wxSimplebook in action.

Notice that is often convenient to use ShowNewPage() instead of the base class AddPage().

There are no special styles defined for this class as it has no visual appearance of its own.

There are also no special events, this class reuses `wxEVT_BOOKCTRL_PAGE_CHANGING` and `wxEVT_BOOKCTRL_PAGE_CHANGED` events for the events it generates if the program calls SetSelection().

Library: wxCore

Category: Book Controls

See also

wxBookCtrl, wxNotebook, Notebook Sample

Since

2.9.5

Public Member Functions

- wxSimplebook ()

  Default constructor.
- wxSimplebook (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Constructs a simple book control.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Really create the window of an object created using default constructor.
- void SetEffects (wxShowEffect showEffect, wxShowEffect hideEffect)

  Set the effects to use for showing and hiding the pages.
- void SetEffect (wxShowEffect effect)

  Set the same effect to use for both showing and hiding the pages.
- void SetEffectsTimeouts (unsigned showTimeout, unsigned hideTimeout)

  Set the effect timeout to use for showing and hiding the pages.
- void SetEffectTimeout (unsigned timeout)

  Set the same effect timeout to use for both showing and hiding the pages.
- bool ShowNewPage (wxWindow *page)

  Add a new page and show it immediately.

Additional Inherited Members

21.679.2 Constructor & Destructor Documentation

wxSimplebook::wxSimplebook ( )

Default constructor.

Use Create() later to really create the control.

wxSimplebook::wxSimplebook (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Constructs a simple book control.

21.679.3 Member Function Documentation

bool wxSimplebook::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Really create the window of an object created using default constructor.

Since

3.0.2

void wxSimplebook::SetEffect (wxShowEffect *effect*)

Set the same effect to use for both showing and hiding the pages.

This is the same as `SetEffects(effect, effect)`.

See also

SetEffectTimeout()

void wxSimplebook::SetEffects (wxShowEffect *showEffect*, wxShowEffect *hideEffect*)

Set the effects to use for showing and hiding the pages.

This method allows to specify the effects passed to wxWindow::ShowWithEffect() and wxWindow::HideWithEffect() respectively when the pages need to be shown or hidden.

By default, no effects are used, but as the pages are only changed by the program and not the user himself, it may be useful to use some visual effects to make the changes more noticeable.

Parameters

showEffect	The effect to use for showing the newly selected page.
hideEffect	The effect to use for hiding the previously selected page.

See also

SetEffectsTimeouts()

void wxSimplebook::SetEffectsTimeouts (unsigned *showTimeout*, unsigned *hideTimeout*)

Set the effect timeout to use for showing and hiding the pages.

This method allows to configure the timeout arguments passed to wxWindow::ShowWithEffect() and wxWindow::↩ HideWithEffect() if a non-default effect is used.

If this method is not called, default, system-dependent timeout is used.

Parameters

showTimeout	Timeout of the show effect, in milliseconds.
hideTimeout	Timeout of the hide effect, in milliseconds.

See also

SetEffects()

void wxSimplebook::SetEffectTimeout (unsigned *timeout*)

Set the same effect timeout to use for both showing and hiding the pages.

This is the same as `SetEffectsTimeouts(timeout, timeout)`.

See also

SetEffect()

bool wxSimplebook::ShowNewPage (wxWindow * *page*)

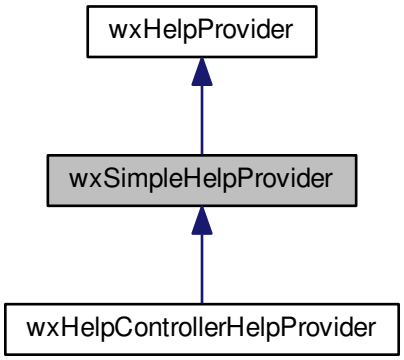
Add a new page and show it immediately.

This is simply a thin wrapper around the base class wxBookCtrlBase::AddPage() method using empty label (which is unused by this class anyhow) and selecting the new page immediately.

21.680 wxSimpleHelpProvider Class Reference

`#include <wx/cshelp.h>`

Inheritance diagram for wxSimpleHelpProvider:



21.680.1 Detailed Description

wxSimpleHelpProvider is an implementation of wxHelpProvider which supports only plain text help strings, and shows the string associated with the control (if any) in a tooltip.

Library: wxCore

Category: Help

See also

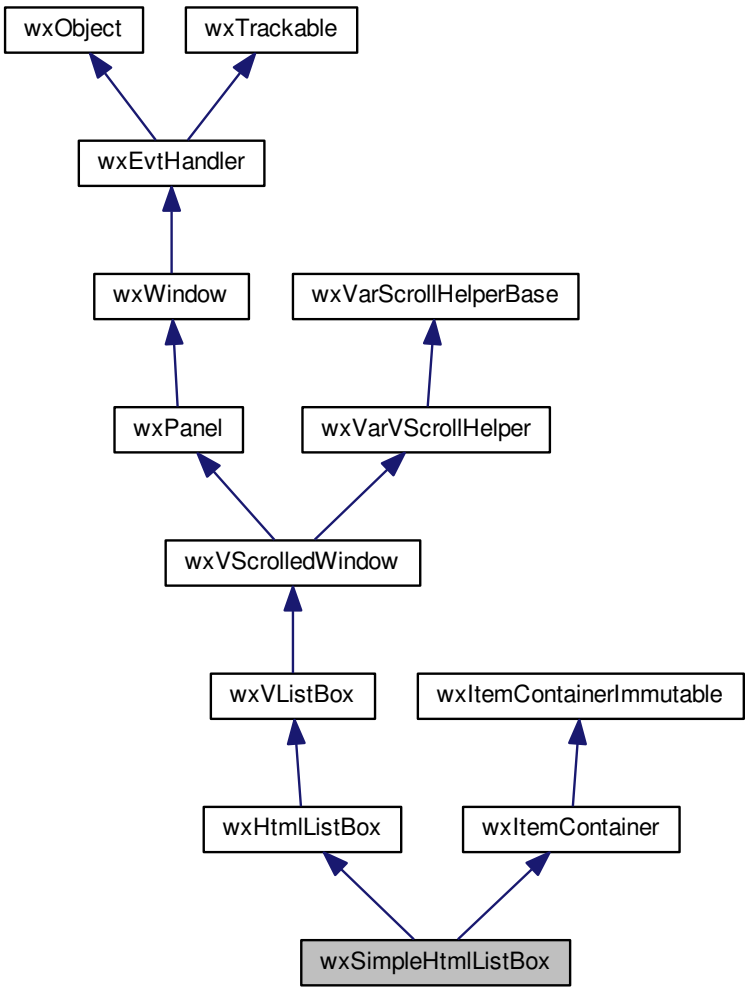
wxHelpProvider, wxHelpControllerHelpProvider, wxContextHelp, wxWindow::SetHelpText()(, wxWindow::↩ GetHelpTextAtPoint()

Additional Inherited Members

21.681 wxSimpleHtmlListBox Class Reference

`#include <wx/htmllbox.h>`

Inheritance diagram for wxSimpleHtmlListBox:



21.681.1 Detailed Description

wxSimpleHtmlListBox is an implementation of wxHtmlListBox which shows HTML content in the listbox rows.

Unlike wxHtmlListBox, this is not an abstract class and thus it has the advantage that you can use it without deriving your own class from it. However, it also has the disadvantage that this is not a virtual control and thus it's not well-suited for those cases where you need to show a huge number of items: every time you add/insert a string, it will be stored internally and thus will take memory.

The interface exposed by wxSimpleHtmlListBox fully implements the wxControlWithItems interface, thus you should refer to wxControlWithItems's documentation for the API reference for adding/removing/retrieving items in the listbox. Also note that the wxVListBox::SetItemCount function is `protected` in wxSimpleHtmlListBox's context so that you cannot call it directly, wxSimpleHtmlListBox will do it for you.

Note: in case you need to append a lot of items to the control at once, make sure to use the Append(const wx↩ ArrayString&) function.

Thus the only difference between a wxListBox and a wxSimpleHtmlListBox is that the latter stores strings which can contain HTML fragments (see the list of tags supported by wxHTML).

Note that the HTML strings you fetch to wxSimpleHtmlListBox should not contain the <html> or <body> tags.

Styles

This class supports the following styles:

- wxHLB_DEFAULT_STYLE: The default style: wxBORDER_SUNKEN
- wxHLB_MULTIPLE: Multiple-selection list: the user can toggle multiple items on and off.

A wxSimpleHtmlListBox emits the same events used by wxListBox and by wxHtmlListBox.

Events emitted by this class

Event macros for events emitted by this class:

- EVT_LISTBOX(id, func): Process a wxEVT_LISTBOX event, when an item on the list is selected. See wxCommandEvent.
- EVT_LISTBOX_DCLICK(id, func): Process a wxEVT_LISTBOX_DCLICK event, when the listbox is double-clicked. See wxCommandEvent.
- EVT_HTML_CELL_CLICKED(id, func): A wxHtmlCell was clicked. See wxHtmlCellEvent.
- EVT_HTML_CELL_HOVER(id, func): The mouse passed over a wxHtmlCell. See wxHtmlCellEvent.
- EVT_HTML_LINK_CLICKED(id, func): A wxHtmlCell which contains an hyperlink was clicked. See wxHtmlLinkEvent

Library: wxHTML

Category: Controls

See also

wxSimpleHtmlListBox::Create

Public Member Functions

- wxSimpleHtmlListBox (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=wxHLB_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxSimpleHtmlListBoxNameStr)

  Constructor, creating and showing the HTML list box.
- wxSimpleHtmlListBox (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=wxHLB_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxSimpleHtmlListBoxNameStr)

  Constructor, creating and showing the HTML list box.
- wxSimpleHtmlListBox ()

  Default constructor, you must call Create() later.
- virtual ~wxSimpleHtmlListBox ()

  Frees the array of stored items and relative client data.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, int n=0, const wxString choices[]=NULL, long style=wxHLB_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxSimpleHtmlListBoxNameStr)

Creates the HTML listbox for two-step construction.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos, const wxSize &size, const wxArrayString &choices, long style=wxHLB_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxSimpleHtmlListBoxNameStr)

  Creates the HTML listbox for two-step construction.

Additional Inherited Members

21.681.2 Constructor & Destructor Documentation

wxSimpleHtmlListBox::wxSimpleHtmlListBox (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = wxHLB_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxSimpleHtmlListBoxNameStr)

Constructor, creating and showing the HTML list box.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. A value of -1 indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
n	Number of strings with which to initialise the control.
choices	An array of strings with which to initialise the control.
style	Window style. See wxHLB_* flags.
validator	Window validator.
name	Window name.

wxSimpleHtmlListBox::wxSimpleHtmlListBox (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = wxHLB_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxSimpleHtmlListBoxNameStr)

Constructor, creating and showing the HTML list box.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. A value of -1 indicates a default value.
pos	Window position.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
choices	An array of strings with which to initialise the control.
style	Window style. See wxHLB_* flags.
validator	Window validator.
name	Window name.

wxSimpleHtmlListBox::wxSimpleHtmlListBox ( )

Default constructor, you must call Create() later.

virtual wxSimpleHtmlListBox::~wxSimpleHtmlListBox ( ) [virtual]

Frees the array of stored items and relative client data.

21.681.3 Member Function Documentation

bool wxSimpleHtmlListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, int *n* = 0, const wxString *choices[]* = NULL, long *style* = wxHLB_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxSimpleHtmlListBoxNameStr)

Creates the HTML listbox for two-step construction.

See wxSimpleHtmlListBox() for further details.

bool wxSimpleHtmlListBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos*, const wxSize & *size*, const wxArrayString & *choices*, long *style* = wxHLB_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxSimpleHtmlListBoxNameStr)

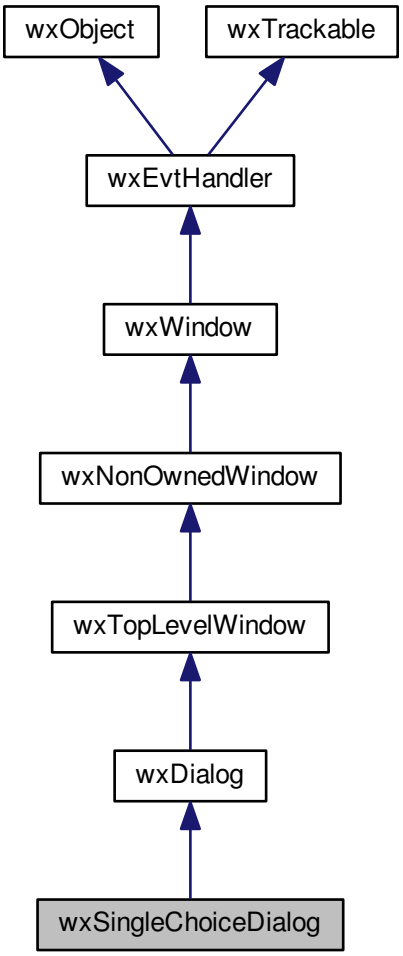
Creates the HTML listbox for two-step construction.

See wxSimpleHtmlListBox() for further details.

21.682 wxSingleChoiceDialog Class Reference

```
#include <wx/choicdlg.h>
```

Inheritance diagram for wxSingleChoiceDialog:



21.682.1 Detailed Description

This class represents a dialog that shows a list of strings, and allows the user to select one.

Double-clicking on a list item is equivalent to single-clicking and then pressing OK.

Styles

This class supports the following styles:

- wxOK: Show an OK button.
- wxCANCEL: Show a Cancel button.
- wxCENTRE: Centre the message. Not Windows.

Library: wxBase

Category: Common Dialogs

See also

wxSingleChoiceDialog Overview, wxMultiChoiceDialog

Public Member Functions

- int GetSelection () const

  Returns the index of selected item.
- void * GetSelectionData () const

  Returns the client data associated with the selection.
- wxString GetStringSelection () const

  Returns the selected string.
- void SetSelection (int selection)

  Sets the index of the initially selected item.
- int ShowModal ()

  Shows the dialog, returning either wxID_OK or wxID_CANCEL.

- wxSingleChoiceDialog (wxWindow *parent, const wxString &message, const wxString &caption, int n, const wxString *choices, void **clientData=NULL, long style=wxCHOICEDLG_STYLE, const wxPoint &pos=wxDefaultPosition)

  Constructor, taking an array of wxString choices and optional client data.
- wxSingleChoiceDialog (wxWindow *parent, const wxString &message, const wxString &caption, const wxArrayString &choices, void **clientData=NULL, long style=wxCHOICEDLG_STYLE, const wxPoint &pos=wxDefaultPosition)

  Constructor, taking an array of wxString choices and optional client data.

Additional Inherited Members

21.682.2 Constructor & Destructor Documentation

wxSingleChoiceDialog::wxSingleChoiceDialog (wxWindow * *parent,* const wxString & *message,* const wxString & *caption,* int *n,* const wxString * *choices,* void ** *clientData* = NULL, long *style* = wxCHOICEDLG_STYLE, const wxPoint & *pos* = wxDefaultPosition)

Constructor, taking an array of wxString choices and optional client data.

Parameters

parent	Parent window.
message	Message to show on the dialog.
caption	The dialog caption.
n	The number of choices.
choices	An array of strings, or a string list, containing the choices.
clientData	An array of client data to be associated with the items. See GetSelectionData().

style	A dialog style (bitlist) containing flags chosen from standard dialog styles and the ones listed in the class documentation. The default value is equivalent to wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER | wxOK | wxCANCEL | wxCENTRE.
pos	Dialog position. Not Windows.

Remarks

Use ShowModal() to show the dialog.

wxPerl Note: Not supported by wxPerl.

wxSingleChoiceDialog::wxSingleChoiceDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption*, const wxArrayString & *choices*, void ** *clientData* = NULL, long *style* = wxCHOICEDLG_STYLE, const wxPoint & *pos* = wxDefaultPosition)

Constructor, taking an array of wxString choices and optional client data.

Parameters

parent	Parent window.
message	Message to show on the dialog.
caption	The dialog caption.
choices	An array of strings, or a string list, containing the choices.
clientData	An array of client data to be associated with the items. See GetSelectionData().
style	A dialog style (bitlist) containing flags chosen from standard dialog styles and the ones listed in the class documentation. The default value is equivalent to wxDEFAULT_DIALOG_STYLE | wxRESIZE_BORDER | wxOK | wxCANCEL | wxCENTRE.
pos	Dialog position. Not Windows.

Remarks

Use ShowModal() to show the dialog.

wxPerl Note: Use an array reference for the *choices* parameter.

21.682.3 Member Function Documentation

int wxSingleChoiceDialog::GetSelection ( ) const

Returns the index of selected item.

void* wxSingleChoiceDialog::GetSelectionData ( ) const

Returns the client data associated with the selection.

Since

2.9.4

wxString wxSingleChoiceDialog::GetStringSelection ( ) const

Returns the selected string.

void wxSingleChoiceDialog::SetSelection (int *selection*)

Sets the index of the initially selected item.

int wxSingleChoiceDialog::ShowModal ( ) [virtual]

Shows the dialog, returning either wxID_OK or wxID_CANCEL.

Reimplemented from wxDialog.

21.683 wxSingleInstanceChecker Class Reference

`#include <wx/snglinst.h>`

21.683.1 Detailed Description

wxSingleInstanceChecker class allows to check that only a single instance of a program is running.

To do it, you should create an object of this class. As long as this object is alive, calls to wxSingleInstanceChecker::IsAnotherRunning from other processes will return true.

As the object should have the life span as big as possible, it makes sense to create it either as a global or in wxApp::OnInit. For example:

```
bool MyApp::OnInit()
{
    m_checker = new wxSingleInstanceChecker;
    if ( m_checker->IsAnotherRunning() )
    {
        wxLogError(_("Another program instance is already running, aborting."));

        delete m_checker; // OnExit() won't be called if we return false
        m_checker = NULL;

        return false;
    }

    ... more initializations ...

    return true;
}

int MyApp::OnExit()
{
    delete m_checker;

    return 0;
}
```

Note that by default wxSingleInstanceChecker::CreateDefault() is used to create the checker meaning that it will be initialized differently for different users and thus will allow different users to run the application concurrently which is usually the required behaviour. However if only a single instance of a program should be running system-wide, you need to call Create() with a custom name which does *not* include wxGetUserId().

This class is implemented for Win32 and Unix platforms (supporting `fcntl()` system call, but almost all of modern Unix systems do) only.

Library: wxBase

Category: Application and Process Management

Public Member Functions

- wxSingleInstanceChecker ()

  Default constructor.
- wxSingleInstanceChecker (const wxString &name, const wxString &path=wxEmptyString)

Constructor calling Create().

- ~wxSingleInstanceChecker ()

  Destructor frees the associated resources.

- bool Create (const wxString &name, const wxString &path=wxEmptyString)

  Initialize the object if it had been created using the default constructor.

- bool CreateDefault ()

  Calls Create() with default name.

- bool IsAnotherRunning () const

  Returns true if another copy of this program is already running and false otherwise.

21.683.2 Constructor & Destructor Documentation

wxSingleInstanceChecker::wxSingleInstanceChecker ( )

Default constructor.

You may call Create() after using it or directly call IsAnotherRunning() in which case CreateDefault() will be implicitly used.

wxSingleInstanceChecker::wxSingleInstanceChecker (const wxString & *name*, const wxString & *path* = wxEmptyString)

Constructor calling Create().

This constructor does exactly the same thing as Create() but doesn't allow to check for errors.

wxSingleInstanceChecker::~wxSingleInstanceChecker ( )

Destructor frees the associated resources.

Note that it is not virtual, this class is not meant to be used polymorphically.

21.683.3 Member Function Documentation

bool wxSingleInstanceChecker::Create (const wxString & *name*, const wxString & *path* = wxEmptyString)

Initialize the object if it had been created using the default constructor.

Note that you can't call Create() more than once, so calling it if the non default ctor had been used is an error.

Parameters

name	Must be given and be as unique as possible. It is used as the mutex name under Win32 and the lock file name under Unix. wxApp::GetAppName() and wxGetUserId() are commonly used to construct this parameter.
path	The path is optional and is ignored under Win32 and used as the directory to create the lock file in under Unix (default is wxGetHomeDir()).

Returns

Returns false if initialization failed, it doesn't mean that another instance is running – use IsAnotherRunning() to check for it.

Note

One of possible reasons while Create() may fail on Unix is that the lock file used for checking already exists but was not created by the user. Therefore applications shouldn't treat failure of this function as fatal condition, because doing so would open them to the possibility of a Denial of Service attack. Instead, they should alert the user about the problem and offer to continue execution without checking if another instance is running.

bool wxSingleInstanceChecker::CreateDefault ( )

Calls Create() with default name.

This method simply calls Create() with a string composed of wxApp::GetAppName() and wxGetUserId().

Because this method uses wxApp::GetAppName(), it may only be called after the global application was constructed.

Since

2.9.1

bool wxSingleInstanceChecker::IsAnotherRunning ( ) const

Returns true if another copy of this program is already running and false otherwise.

Notice that if the object was created using the default constructor Create() hadn't been called before this method, it will call CreateDefault() automatically.

21.684 wxSize Class Reference

```
#include <wx/gdicmn.h>
```

21.684.1 Detailed Description

A wxSize is a useful data structure for graphics operations.

It simply contains integer *width* and *height* members.

Note that the width and height stored inside a wxSize object may be negative and that wxSize functions do not perform any check against negative values (this is used to e.g. store the special -1 value in wxDefaultSize instance). See also IsFullySpecified() and SetDefaults() for utility functions regarding the special -1 value.

wxSize is used throughout wxWidgets as well as wxPoint which, although almost equivalent to wxSize, has a different meaning: wxPoint represents a position while wxSize represents the size.

Library: wxCore

Category: Data Structures

Predefined objects/pointers: wxDefaultSize

See also

wxPoint, wxRealPoint

Public Member Functions

- wxSize ()

  Initializes this size object with zero width and height.
- wxSize (int width, int height)

  Initializes this size object with the given width and height.
- void DecTo (const wxSize &size)

  Decrements this object so that both of its dimensions are not greater than the corresponding dimensions of the size.
- void DecToIfSpecified (const wxSize &size)

  Decrements this object to be not bigger than the given size ignoring non-specified components.
- int GetHeight () const

  Gets the height member.
- int GetWidth () const

  Gets the width member.
- void IncTo (const wxSize &size)

  Increments this object so that both of its dimensions are not less than the corresponding dimensions of the size.
- bool IsFullySpecified () const

  Returns true if neither of the size object components is equal to -1, which is used as default for the size values in wxWidgets (hence the predefined wxDefaultSize has both of its components equal to -1).
- wxSize & Scale (float xscale, float yscale)

  Scales the dimensions of this object by the given factors.
- void Set (int width, int height)

  Sets the width and height members.
- void SetDefaults (const wxSize &sizeDefault)

  Combine this size object with another one replacing the default (i.e. equal to -1) components of this object with those of the other.
- void SetHeight (int height)

  Sets the height.
- void SetWidth (int width)

  Sets the width.

- void DecBy (const wxPoint &pt)

  Decreases the size in both x and y directions.
- void DecBy (const wxSize &size)

  Decreases the size in both x and y directions.
- void DecBy (int dx, int dy)

  Decreases the size in both x and y directions.
- void DecBy (int d)

  Decreases the size in both x and y directions.

- void IncBy (const wxPoint &pt)

  Increases the size in both x and y directions.
- void IncBy (const wxSize &size)

  Increases the size in both x and y directions.
- void IncBy (int dx, int dy)

  Increases the size in both x and y directions.
- void IncBy (int d)

  Increases the size in both x and y directions.

Miscellaneous operators

Note that these operators are documented as class members (to make them easier to find) but, as their prototype shows, they are implemented as global operators; note that this is transparent to the user but it helps to understand why the following functions are documented to take the wxSize they operate on as an explicit argument.

- wxSize & operator= (const wxSize &sz)
- bool operator== (const wxSize &s1, const wxSize &s2)
- bool operator!= (const wxSize &s1, const wxSize &s2)
- wxSize operator+ (const wxSize &s1, const wxSize &s2)
- wxSize operator- (const wxSize &s1, const wxSize &s2)
- wxSize & operator+= (const wxSize &sz)
- wxSize & operator-= (const wxSize &sz)
- wxSize operator/ (const wxSize &sz, int factor)
- wxSize operator* (const wxSize &sz, int factor)
- wxSize operator* (int factor, const wxSize &sz)
- wxSize & operator/= (int factor)
- wxSize & operator*= (int factor)

21.684.2 Constructor & Destructor Documentation

wxSize::wxSize ( )

Initializes this size object with zero width and height.

wxSize::wxSize (int *width*, int *height*)

Initializes this size object with the given *width* and *height*.

21.684.3 Member Function Documentation

void wxSize::DecBy (const wxPoint & *pt*)

Decreases the size in both x and y directions.

See also

IncBy()

void wxSize::DecBy (const wxSize & *size*)

Decreases the size in both x and y directions.

See also

IncBy()

void wxSize::DecBy (int *dx*, int *dy*)

Decreases the size in both x and y directions.

See also

IncBy()

void wxSize::DecBy (int *d*)

Decreases the size in both x and y directions.

See also

IncBy()

void wxSize::DecTo (const wxSize & *size*)

Decrements this object so that both of its dimensions are not greater than the corresponding dimensions of the *size*.

See also

IncTo()

void wxSize::DecToIfSpecified (const wxSize & *size*)

Decrements this object to be not bigger than the given size ignoring non-specified components.

This is similar to DecTo() but doesn't do anything for x or y component if the same component of *size* is not specified, i.e. set to wxDefaultCoord.

Since

2.9.5

int wxSize::GetHeight ( ) const

Gets the height member.

int wxSize::GetWidth ( ) const

Gets the width member.

void wxSize::IncBy (const wxPoint & *pt*)

Increases the size in both x and y directions.

See also

DecBy()

void wxSize::IncBy (const wxSize & *size*)

Increases the size in both x and y directions.

See also

DecBy()

void wxSize::IncBy (int *dx,* int *dy*)

Increases the size in both x and y directions.

See also

DecBy()

void wxSize::IncBy (int *d*)

Increases the size in both x and y directions.

See also

DecBy()

void wxSize::IncTo (const wxSize & *size*)

Increments this object so that both of its dimensions are not less than the corresponding dimensions of the *size*.

See also

DecTo()

bool wxSize::IsFullySpecified ( ) const

Returns true if neither of the size object components is equal to -1, which is used as default for the size values in wxWidgets (hence the predefined wxDefaultSize has both of its components equal to -1).

This method is typically used before calling SetDefaults().

bool wxSize::operator!= (const wxSize & *s1,* const wxSize & *s2*)

wxSize wxSize::operator∗ (const wxSize & *sz,* int *factor*)

wxSize wxSize::operator∗ (int *factor,* const wxSize & *sz*)

wxSize& wxSize::operator∗= (int *factor*)

wxSize wxSize::operator+ (const wxSize & *s1,* const wxSize & *s2*)

wxSize& wxSize::operator+= (const wxSize & *sz*)

wxSize wxSize::operator- (const wxSize & *s1,* const wxSize & *s2*)

wxSize& wxSize::operator-= (const wxSize & *sz*)

wxSize wxSize::operator/ (const wxSize & *sz,* int *factor*)

wxSize& wxSize::operator/= (int *factor*)

wxSize& wxSize::operator= (const wxSize & *sz*)

bool wxSize::operator== (const wxSize & *s1,* const wxSize & *s2*)

wxSize& wxSize::Scale (float *xscale,* float *yscale*)

Scales the dimensions of this object by the given factors.

If you want to scale both dimensions by the same factor you can also use operator*=().

Returns

A reference to this object (so that you can concatenate other operations in the same line).

void wxSize::Set (int *width,* int *height*)

Sets the width and height members.

void wxSize::SetDefaults (const wxSize & *sizeDefault*)

Combine this size object with another one replacing the default (i.e. equal to -1) components of this object with those of the other.

It is typically used like this:

```
if ( !size.IsFullySpecified() )
{
    size.SetDefaults(GetDefaultSize());
}
```

See also

IsFullySpecified()

void wxSize::SetHeight (int *height*)

Sets the height.

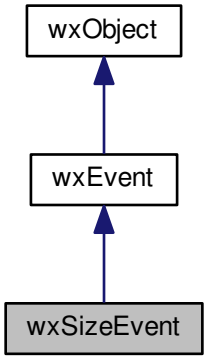
void wxSize::SetWidth (int *width*)

Sets the width.

21.685 wxSizeEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxSizeEvent:



21.685.1 Detailed Description

A size event holds information about size change events of wxWindow.

The EVT_SIZE handler function will be called when the window has been resized.

You may wish to use this for frames to resize their child windows as appropriate.

Note that the size passed is of the whole window: call wxWindow::GetClientSize() for the area which may be used by the application.

When a window is resized, usually only a small part of the window is damaged and you may only need to repaint that area. However, if your drawing depends on the size of the window, you may need to clear the DC explicitly and repaint the whole window. In which case, you may need to call wxWindow::Refresh to invalidate the entire window.

Important : Sizers (see Sizers Overview) rely on size events to function correctly. Therefore, in a sizer-based layout, do not forget to call Skip on all size events you catch (and don't catch size events at all when you don't need to).

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSizeEvent& event)

Event macros:

- EVT_SIZE(func): Process a `wxEVT_SIZE` event.

Library: wxCore

Category: Events

See also

wxSize, Events and Event Handling

Public Member Functions

- wxSizeEvent (const wxSize &sz, int id=0)

  Constructor.

- wxSize GetSize () const

  Returns the entire size of the window generating the size change event.

- void SetSize (wxSize size)
- wxRect GetRect () const
- void SetRect (wxRect rect)

Additional Inherited Members

21.685.2 Constructor & Destructor Documentation

wxSizeEvent::wxSizeEvent (const wxSize & *sz*, int *id* = 0)

Constructor.

21.685.3 Member Function Documentation

wxRect wxSizeEvent::GetRect ( ) const

wxSize wxSizeEvent::GetSize ( ) const

Returns the entire size of the window generating the size change event.

This is the new total size of the window, i.e. the same size as would be returned by wxWindow::GetSize() if it were called now. Use wxWindow::GetClientSize() if you catch this event in a top level window such as wxFrame to find the size available for the window contents.

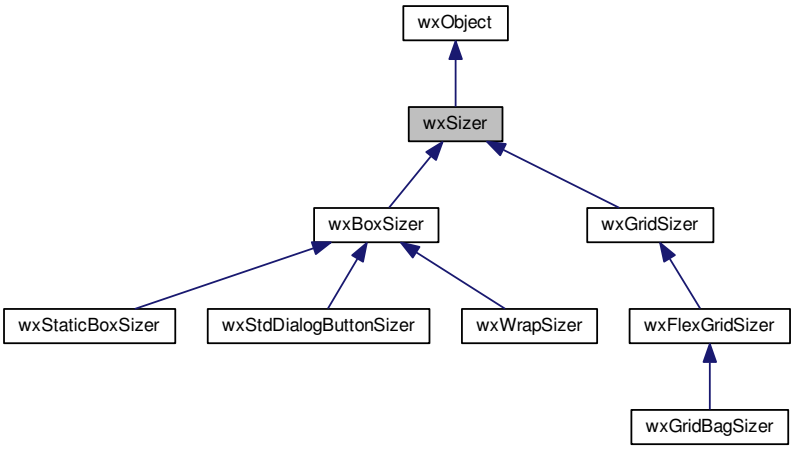
void wxSizeEvent::SetRect (wxRect *rect*)

void wxSizeEvent::SetSize (wxSize *size*)

21.686 wxSizer Class Reference

`#include <wx/sizer.h>`

Inheritance diagram for wxSizer:



21.686.1 Detailed Description

wxSizer is the abstract base class used for laying out subwindows in a window.

You cannot use wxSizer directly; instead, you will have to use one of the sizer classes derived from it. Currently there are wxBoxSizer, wxStaticBoxSizer, wxGridSizer, wxFlexGridSizer, wxWrapSizer and wxGridBagSizer.

The layout algorithm used by sizers in wxWidgets is closely related to layout in other GUI toolkits, such as Java's AWT, the GTK toolkit or the Qt toolkit. It is based upon the idea of the individual subwindows reporting their minimal required size and their ability to get stretched if the size of the parent window has changed.

This will most often mean that the programmer does not set the original size of a dialog in the beginning, rather the dialog will be assigned a sizer and this sizer will be queried about the recommended size. The sizer in turn will query its children, which can be normal windows, empty space or other sizers, so that a hierarchy of sizers can be constructed. Note that wxSizer does not derive from wxWindow and thus does not interfere with tab ordering and requires very little resources compared to a real window on screen.

What makes sizers so well fitted for use in wxWidgets is the fact that every control reports its own minimal size and the algorithm can handle differences in font sizes or different window (dialog item) sizes on different platforms without problems. If e.g. the standard font as well as the overall design of Motif widgets requires more space than on Windows, the initial dialog size will automatically be bigger on Motif than on Windows.

Sizers may also be used to control the layout of custom drawn items on the window. The wxSizer::Add(), wxSizer::Insert(), and wxSizer::Prepend() functions return a pointer to the newly added wxSizerItem. Just add empty space of the desired size and attributes, and then use the wxSizerItem::GetRect() method to determine where the drawing operations should take place.

Please notice that sizers, like child windows, are owned by the library and will be deleted by it which implies that they must be allocated on the heap. However if you create a sizer and do not add it to another sizer or window, the library wouldn't be able to delete such an orphan sizer and in this, and only this, case it should be deleted explicitly.

21.686.2 wxSizer flags

The "flag" argument accepted by wxSizeItem constructors and other functions, e.g. wxSizer::Add(), is OR-combination of the following flags. Two main behaviours are defined using these flags. One is the border around a window: the border parameter determines the border width whereas the flags given here determine which side(s) of the item that the border will be added. The other flags determine how the sizer item behaves when the space

allotted to the sizer changes, and is somewhat dependent on the specific kind of sizer used.

wxTOP wxBOTTOM wxLEFT wxRIGHT wxALL	These flags are used to specify which side(s) of the sizer item the border width will apply to.
wxEXPAND	The item will be expanded to fill the space assigned to the item.
wxSHAPED	The item will be expanded as much as possible while also maintaining its aspect ratio.
wxFIXED_MINSIZE	Normally wxSizers will use GetAdjustedBestSize() to determine what the minimal size of window items should be, and will use that size to calculate the layout. This allows layouts to adjust when an item changes and its best size becomes different. If you would rather have a window item stay the size it started with then use `wxFIXED_MINSIZE`.
wxRESERVE_SPACE_EVEN_IF_HIDDEN	Normally wxSizers don't allocate space for hidden windows or other items. This flag overrides this behaviour so that sufficient space is allocated for the window even if it isn't visible. This makes it possible to dynamically show and hide controls without resizing parent dialog, for example. (Available since 2.8.8.)
wxALIGN_CENTER wxALIGN_CENTRE wxALIGN_LEFT wxALIGN_RIGHT wxALIGN_TOP wxALIGN_BOTTOM wxALIGN_CENTER_VERTICAL wxALIGN_CENTRE_VERTICAL wxALIGN_CENTER_HORIZONTAL wxALIGN_CENTRE_HORIZONTAL	The `wxALIGN_*` flags allow you to specify the alignment of the item within the space allotted to it by the sizer, adjusted for the border if any.

Library: wxCore

Category: Window Layout

See also

Sizers Overview

Public Member Functions

- wxSizer ()

  The constructor.
- virtual ~wxSizer ()

  The destructor.
- wxSizerItem * Add (wxWindow *window, const wxSizerFlags &flags)

  Appends a child to the sizer.
- wxSizerItem * Add (wxWindow *window, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Appends a child to the sizer.
- wxSizerItem * Add (wxSizer *sizer, const wxSizerFlags &flags)

  Appends a child to the sizer.
- wxSizerItem * Add (wxSizer *sizer, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

Appends a child to the sizer.

- wxSizerItem * Add (int width, int height, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Appends a spacer child to the sizer.
- wxSizerItem * Add (int width, int height, const wxSizerFlags &flags)

  Appends a spacer child to the sizer.
- wxSizerItem * Add (wxSizerItem *item)
- virtual wxSizerItem * AddSpacer (int size)

  This base function adds non-stretchable space to both the horizontal and vertical orientation of the sizer.
- wxSizerItem * AddStretchSpacer (int prop=1)

  Adds stretchable space to the sizer.
- virtual wxSize CalcMin ()=0

  This method is abstract and has to be overwritten by any derived class.
- virtual void Clear (bool delete_windows=false)

  Detaches all children from the sizer.
- wxSize ComputeFittingClientSize (wxWindow *window)

  Computes client area size for window so that it matches the sizer's minimal size.
- wxSize ComputeFittingWindowSize (wxWindow *window)

  Like ComputeFittingClientSize(), but converts the result into window size.
- virtual bool Detach (wxWindow *window)

  Detach the child window from the sizer without destroying it.
- virtual bool Detach (wxSizer *sizer)

  Detach the child sizer from the sizer without destroying it.
- virtual bool Detach (int index)

  Detach a item at position index from the sizer without destroying it.
- wxSize Fit (wxWindow *window)

  Tell the sizer to resize the window so that its client area matches the sizer's minimal size (ComputeFittingClientSize() is called to determine it).
- void FitInside (wxWindow *window)

  Tell the sizer to resize the virtual size of the window to match the sizer's minimal size.
- virtual bool InformFirstDirection (int direction, int size, int availableOtherDir)

  Inform sizer about the first direction that has been decided (by parent item).
- wxWindow * GetContainingWindow () const

  Returns the window this sizer is used in or NULL if none.
- void SetContainingWindow (wxWindow *window)

  Set the window this sizer is used in.
- size_t GetItemCount () const

  Returns the number of items in the sizer.
- wxSizerItem * GetItem (wxWindow *window, bool recursive=false)

  Finds wxSizerItem which holds the given window.
- wxSizerItem * GetItem (wxSizer *sizer, bool recursive=false)

  Finds wxSizerItem which holds the given sizer.
- wxSizerItem * GetItem (size_t index)

  Finds wxSizerItem which is located in the sizer at position index.
- wxSizerItem * GetItemById (int id, bool recursive=false)

  Finds item of the sizer which has the given id.
- wxSize GetMinSize ()

  Returns the minimal size of the sizer.
- wxPoint GetPosition () const

  Returns the current position of the sizer.
- wxSize GetSize () const

  Returns the current size of the sizer.

- bool Hide (wxWindow *window, bool recursive=false)

  Hides the child window.
- bool Hide (wxSizer *sizer, bool recursive=false)

  Hides the child sizer.
- bool Hide (size_t index)

  Hides the item at position index.
- wxSizerItem * Insert (size_t index, wxWindow *window, const wxSizerFlags &flags)

  Insert a child into the sizer before any existing item at index.
- wxSizerItem * Insert (size_t index, wxWindow *window, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Insert a child into the sizer before any existing item at index.
- wxSizerItem * Insert (size_t index, wxSizer *sizer, const wxSizerFlags &flags)

  Insert a child into the sizer before any existing item at index.
- wxSizerItem * Insert (size_t index, wxSizer *sizer, int proportion=0, int flag=0, int border=0, wxObject *user↩ Data=NULL)

  Insert a child into the sizer before any existing item at index.
- wxSizerItem * Insert (size_t index, int width, int height, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Insert a child into the sizer before any existing item at index.
- wxSizerItem * Insert (size_t index, int width, int height, const wxSizerFlags &flags)

  Insert a child into the sizer before any existing item at index.
- wxSizerItem * Insert (size_t index, wxSizerItem *item)
- wxSizerItem * InsertSpacer (size_t index, int size)

  Inserts non-stretchable space to the sizer.
- wxSizerItem * InsertStretchSpacer (size_t index, int prop=1)

  Inserts stretchable space to the sizer.
- bool IsEmpty () const

  Return true if the sizer has no elements.
- bool IsShown (wxWindow *window) const

  Returns true if the window is shown.
- bool IsShown (wxSizer *sizer) const

  Returns true if the sizer is shown.
- bool IsShown (size_t index) const

  Returns true if the item at index is shown.
- virtual void Layout ()

  Call this to force layout of the children anew, e.g. after having added a child to or removed a child (window, other sizer or space) from the sizer while keeping the current dimension.
- wxSizerItem * Prepend (wxWindow *window, const wxSizerFlags &flags)

  Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.
- wxSizerItem * Prepend (wxWindow *window, int proportion=0, int flag=0, int border=0, wxObject *user↩ Data=NULL)

  Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.
- wxSizerItem * Prepend (wxSizer *sizer, const wxSizerFlags &flags)

  Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.
- wxSizerItem * Prepend (wxSizer *sizer, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.
- wxSizerItem * Prepend (int width, int height, int proportion=0, int flag=0, int border=0, wxObject *user↩ Data=NULL)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

- wxSizerItem * Prepend (int width, int height, const wxSizerFlags &flags)

  Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.
- wxSizerItem * Prepend (wxSizerItem *item)
- wxSizerItem * PrependSpacer (int size)

  Prepends non-stretchable space to the sizer.
- wxSizerItem * PrependStretchSpacer (int prop=1)

  Prepends stretchable space to the sizer.
- virtual void RecalcSizes ()=0

  This method is abstract and has to be overwritten by any derived class.
- virtual bool Remove (wxWindow *window)

  *Removes a child window from the sizer, but does **not** destroy it (because windows are owned by their parent window, not the sizer).*
- virtual bool Remove (wxSizer *sizer)

  Removes a sizer child from the sizer and destroys it.
- virtual bool Remove (int index)

  Removes a child from the sizer and destroys it if it is a sizer or a spacer, but not if it is a window (because windows are owned by their parent window, not the sizer).
- virtual bool Replace (wxWindow *oldwin, wxWindow *newwin, bool recursive=false)

  Detaches the given oldwin from the sizer and replaces it with the given newwin.
- virtual bool Replace (wxSizer *oldsz, wxSizer *newsz, bool recursive=false)

  Detaches the given oldsz from the sizer and replaces it with the given newsz.
- virtual bool Replace (size_t index, wxSizerItem *newitem)

  Detaches the given item at position index from the sizer and replaces it with the given wxSizerItem newitem.
- void SetDimension (int x, int y, int width, int height)

  Call this to force the sizer to take the given dimension and thus force the items owned by the sizer to resize themselves according to the rules defined by the parameter in the Add() and Prepend() methods.
- void SetDimension (const wxPoint &pos, const wxSize &size)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetMinSize (const wxSize &size)

  Call this to give the sizer a minimal size.
- void SetMinSize (int width, int height)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetSizeHints (wxWindow *window)

  This method first calls Fit() and then wxTopLevelWindow::SetSizeHints() on the window passed to it.
- void SetVirtualSizeHints (wxWindow *window)

  Tell the sizer to set the minimal size of the window virtual area to match the sizer's minimal size.
- bool Show (wxWindow *window, bool show=true, bool recursive=false)

  Shows or hides the window.
- bool Show (wxSizer *sizer, bool show=true, bool recursive=false)

  Shows or hides sizer.
- bool Show (size_t index, bool show=true)

  Shows the item at index.
- virtual void ShowItems (bool show)

  Show or hide all items managed by the sizer.

- wxSizerItemList & GetChildren ()

  Returns the list of the items in this sizer.

- const wxSizerItemList & GetChildren () const

  Returns the list of the items in this sizer.

- bool SetItemMinSize (wxWindow *window, int width, int height)

  Set an item's minimum size by window, sizer, or position.
- bool SetItemMinSize (wxWindow *window, const wxSize &size)

  Set an item's minimum size by window, sizer, or position.
- bool SetItemMinSize (wxSizer *sizer, int width, int height)

  Set an item's minimum size by window, sizer, or position.
- bool SetItemMinSize (wxSizer *sizer, const wxSize &size)

  Set an item's minimum size by window, sizer, or position.
- bool SetItemMinSize (size_t index, int width, int height)

  Set an item's minimum size by window, sizer, or position.
- bool SetItemMinSize (size_t index, const wxSize &size)

  Set an item's minimum size by window, sizer, or position.

Additional Inherited Members

21.686.3 Constructor & Destructor Documentation

wxSizer::wxSizer ( )

The constructor.

Note that wxSizer is an abstract base class and may not be instantiated.

virtual wxSizer::~wxSizer ( ) [virtual]

The destructor.

21.686.4 Member Function Documentation

wxSizerItem* wxSizer::Add (wxWindow * *window,* const wxSizerFlags & *flags*)

Appends a child to the sizer.

wxSizer itself is an abstract class, but the parameters are equivalent in the derived classes that you will instantiate to use it so they are described here:

Parameters

window	The window to be added to the sizer. Its initial size (either set explicitly by the user or calculated internally when using wxDefaultSize) is interpreted as the minimal and in many cases also the initial size.
flags	A wxSizerFlags object that enables you to specify most of the above parameters more conveniently.

wxSizerItem* wxSizer::Add (wxWindow * *window,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Appends a child to the sizer.

wxSizer itself is an abstract class, but the parameters are equivalent in the derived classes that you will instantiate to use it so they are described here:

Parameters

window	The window to be added to the sizer. Its initial size (either set explicitly by the user or calculated internally when using wxDefaultSize) is interpreted as the minimal and in many cases also the initial size.
proportion	Although the meaning of this parameter is undefined in wxSizer, it is used in wxBoxSizer to indicate if a child of a sizer can change its size in the main orientation of the wxBoxSizer - where 0 stands for not changeable and a value of more than zero is interpreted relative to the value of other children of the same wxBoxSizer. For example, you might have a horizontal wxBoxSizer with three children, two of which are supposed to change their size with the sizer. Then the two stretchable windows would get a value of 1 each to make them grow and shrink equally with the sizer's horizontal dimension.
flag	OR-combination of flags affecting sizer's behaviour. See wxSizer flags list for details.
border	Determines the border width, if the flag parameter is set to include any border flag.
userData	Allows an extra object to be attached to the sizer item, for use in derived classes when sizing information is more complex than the proportion and flag will allow for.

wxSizerItem* wxSizer::Add (wxSizer * *sizer,* const wxSizerFlags & *flags*)

Appends a child to the sizer.

wxSizer itself is an abstract class, but the parameters are equivalent in the derived classes that you will instantiate to use it so they are described here:

Parameters

sizer	The (child-)sizer to be added to the sizer. This allows placing a child sizer in a sizer and thus to create hierarchies of sizers (typically a vertical box as the top sizer and several horizontal boxes on the level beneath).
flags	A wxSizerFlags object that enables you to specify most of the above parameters more conveniently.

wxSizerItem* wxSizer::Add (wxSizer * *sizer,* int *proportion* = `0`, int *flag* = `0`, int *border* = `0`, wxObject * *userData* = `NULL`)

Appends a child to the sizer.

wxSizer itself is an abstract class, but the parameters are equivalent in the derived classes that you will instantiate to use it so they are described here:

Parameters

sizer	The (child-)sizer to be added to the sizer. This allows placing a child sizer in a sizer and thus to create hierarchies of sizers (typically a vertical box as the top sizer and several horizontal boxes on the level beneath).
proportion	Although the meaning of this parameter is undefined in wxSizer, it is used in wxBoxSizer to indicate if a child of a sizer can change its size in the main orientation of the wxBoxSizer - where 0 stands for not changeable and a value of more than zero is interpreted relative to the value of other children of the same wxBoxSizer. For example, you might have a horizontal wxBoxSizer with three children, two of which are supposed to change their size with the sizer. Then the two stretchable windows would get a value of 1 each to make them grow and shrink equally with the sizer's horizontal dimension.

flag	OR-combination of flags affecting sizer's behaviour. See wxSizer flags list for details.
border	Determines the border width, if the flag parameter is set to include any border flag.
userData	Allows an extra object to be attached to the sizer item, for use in derived classes when sizing information is more complex than the proportion and flag will allow for.

wxSizerItem* wxSizer::Add (int *width*, int *height*, int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Appends a spacer child to the sizer.

wxSizer itself is an abstract class, but the parameters are equivalent in the derived classes that you will instantiate to use it so they are described here.

width and *height* specify the dimension of a spacer to be added to the sizer. Adding spacers to sizers gives more flexibility in the design of dialogs; imagine for example a horizontal box with two buttons at the bottom of a dialog: you might want to insert a space between the two buttons and make that space stretchable using the proportion flag and the result will be that the left button will be aligned with the left side of the dialog and the right button with the right side - the space in between will shrink and grow with the dialog.

Parameters

width	Width of the spacer.
height	Height of the spacer.
proportion	Although the meaning of this parameter is undefined in wxSizer, it is used in wxBoxSizer to indicate if a child of a sizer can change its size in the main orientation of the wxBoxSizer - where 0 stands for not changeable and a value of more than zero is interpreted relative to the value of other children of the same wxBoxSizer. For example, you might have a horizontal wxBoxSizer with three children, two of which are supposed to change their size with the sizer. Then the two stretchable windows would get a value of 1 each to make them grow and shrink equally with the sizer's horizontal dimension.
flag	OR-combination of flags affecting sizer's behaviour. See wxSizer flags list for details.
border	Determines the border width, if the flag parameter is set to include any border flag.
userData	Allows an extra object to be attached to the sizer item, for use in derived classes when sizing information is more complex than the proportion and flag will allow for.

wxSizerItem* wxSizer::Add (int *width*, int *height*, const wxSizerFlags & *flags*)

Appends a spacer child to the sizer.

Parameters

width	Width of the spacer.
height	Height of the spacer.
flags	A wxSizerFlags object that enables you to specify most of the other parameters more conveniently.

wxSizerItem* wxSizer::Add (wxSizerItem * *item*)

virtual wxSizerItem* wxSizer::AddSpacer (int *size*) [virtual]

This base function adds non-stretchable space to both the horizontal and vertical orientation of the sizer.

More readable way of calling:

```
wxSizer::Add(size, size, 0).
```

See also

wxBoxSizer::AddSpacer()

Reimplemented in wxBoxSizer.

wxSizerItem* wxSizer::AddStretchSpacer (int *prop* = 1)

Adds stretchable space to the sizer.

More readable way of calling:

```
wxSizer::Add(0, 0, prop).
```

virtual wxSize wxSizer::CalcMin ( ) [pure virtual]

This method is abstract and has to be overwritten by any derived class.

Here, the sizer will do the actual calculation of its children's minimal sizes.

Implemented in wxBoxSizer, wxStaticBoxSizer, wxGridSizer, wxFlexGridSizer, wxStdDialogButtonSizer, wxGridBagSizer, and wxWrapSizer.

virtual void wxSizer::Clear (bool *delete_windows* = false) [virtual]

Detaches all children from the sizer.

If *delete_windows* is true then child windows will also be deleted.

Notice that child sizers are always deleted, as a general consequence of the principle that sizers own their sizer children, but don't own their window children (because they are already owned by their parent windows).

wxSize wxSizer::ComputeFittingClientSize (wxWindow * *window*)

Computes client area size for *window* so that it matches the sizer's minimal size.

Unlike GetMinSize(), this method accounts for other constraints imposed on *window*, namely display's size (returned size will never be too large for the display) and maximum window size if previously set by wxWindow::SetMaxSize().

The returned value is suitable for passing to wxWindow::SetClientSize() or wxWindow::SetMinClientSize().

Since

2.8.8

See also

ComputeFittingWindowSize(), Fit()

wxSize wxSizer::ComputeFittingWindowSize (wxWindow * *window*)

Like ComputeFittingClientSize(), but converts the result into window size.

The returned value is suitable for passing to wxWindow::SetSize() or wxWindow::SetMinSize().

Since

2.8.8

See also

ComputeFittingClientSize(), Fit()

virtual bool wxSizer::Detach (wxWindow * *window*) [virtual]

Detach the child *window* from the sizer without destroying it.

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after detaching a child from the sizer.

Returns true if the child item was found and detached, false otherwise.

See also

Remove()

virtual bool wxSizer::Detach (wxSizer * *sizer*) [virtual]

Detach the child *sizer* from the sizer without destroying it.

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after detaching a child from the sizer.

Returns true if the child item was found and detached, false otherwise.

See also

Remove()

virtual bool wxSizer::Detach (int *index*) [virtual]

Detach a item at position *index* from the sizer without destroying it.

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after detaching a child from the sizer. Returns true if the child item was found and detached, false otherwise.

See also

Remove()

wxSize wxSizer::Fit (wxWindow * *window*)

Tell the sizer to resize the *window* so that its client area matches the sizer's minimal size (ComputeFittingClientSize() is called to determine it).

This is commonly done in the constructor of the window itself, see sample in the description of wxBoxSizer.

Returns

The new window size.

See also

ComputeFittingClientSize(), ComputeFittingWindowSize()

void wxSizer::FitInside (wxWindow * *window*)

Tell the sizer to resize the virtual size of the *window* to match the sizer's minimal size.

This will not alter the on screen size of the window, but may cause the addition/removal/alteration of scrollbars required to view the virtual area in windows which manage it.

See also

wxScrolled::SetScrollbars(), SetVirtualSizeHints()

wxSizerItemList& wxSizer::GetChildren ( )

Returns the list of the items in this sizer.

The elements of type-safe wxList `wxSizerItemList` are pointers to objects of type wxSizerItem.

const wxSizerItemList& wxSizer::GetChildren ( ) const

Returns the list of the items in this sizer.

The elements of type-safe wxList `wxSizerItemList` are pointers to objects of type wxSizerItem.

wxWindow* wxSizer::GetContainingWindow ( ) const

Returns the window this sizer is used in or NULL if none.

wxSizerItem* wxSizer::GetItem (wxWindow * *window*, bool *recursive* = `false`)

Finds wxSizerItem which holds the given *window*.

Use parameter *recursive* to search in subsizers too. Returns pointer to item or NULL.

wxSizerItem* wxSizer::GetItem (wxSizer * *sizer*, bool *recursive* = `false`)

Finds wxSizerItem which holds the given *sizer*.

Use parameter *recursive* to search in subsizers too. Returns pointer to item or NULL.

wxSizerItem* wxSizer::GetItem (size_t *index*)

Finds wxSizerItem which is located in the sizer at position *index*.

Use parameter *recursive* to search in subsizers too. Returns pointer to item or NULL.

wxSizerItem* wxSizer::GetItemById (int *id*, bool *recursive* = `false`)

Finds item of the sizer which has the given *id*.

This *id* is not the window id but the id of the wxSizerItem itself. This is mainly useful for retrieving the sizers created from XRC resources. Use parameter *recursive* to search in subsizers too. Returns pointer to item or NULL.

size_t wxSizer::GetItemCount ( ) const

Returns the number of items in the sizer.

If you just need to test whether the sizer is empty or not you can also use IsEmpty() function.

wxSize wxSizer::GetMinSize ( )

Returns the minimal size of the sizer.

This is either the combined minimal size of all the children and their borders or the minimal size set by SetMinSize(), depending on which is bigger. Note that the returned value is client size, not window size. In particular, if you use the value to set toplevel window's minimal or actual size, use wxWindow::SetMinClientSize() or wxWindow::SetClientSize(), not wxWindow::SetMinSize() or wxWindow::SetSize().

wxPoint wxSizer::GetPosition ( ) const

Returns the current position of the sizer.

wxSize wxSizer::GetSize ( ) const

Returns the current size of the sizer.

bool wxSizer::Hide (wxWindow * *window,* bool *recursive* = `false`)

Hides the child *window*.

To make a sizer item disappear, use Hide() followed by Layout().

Use parameter *recursive* to hide elements found in subsizers. Returns true if the child item was found, false otherwise.

See also

IsShown(), Show()

bool wxSizer::Hide (wxSizer * *sizer,* bool *recursive* = `false`)

Hides the child *sizer*.

To make a sizer item disappear, use Hide() followed by Layout().

Use parameter *recursive* to hide elements found in subsizers. Returns true if the child item was found, false otherwise.

See also

IsShown(), Show()

bool wxSizer::Hide (size_t *index*)

Hides the item at position *index*.

To make a sizer item disappear, use Hide() followed by Layout().

Use parameter *recursive* to hide elements found in subsizers. Returns true if the child item was found, false otherwise.

See also

IsShown(), Show()

virtual bool wxSizer::InformFirstDirection (int *direction,* int *size,* int *availableOtherDir*) `[virtual]`

Inform sizer about the first direction that has been decided (by parent item).

Returns true if it made use of the information (and recalculated min size).

Reimplemented in wxWrapSizer.

wxSizerItem* wxSizer::Insert (size_t *index,* wxWindow * *window,* const wxSizerFlags & *flags*)

Insert a child into the sizer before any existing item at *index*.

See Add() for the meaning of the other parameters.

wxSizerItem* wxSizer::Insert (size_t *index,* wxWindow * *window,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Insert a child into the sizer before any existing item at *index*.

See Add() for the meaning of the other parameters.

wxSizerItem* wxSizer::Insert (size_t *index,* wxSizer * *sizer,* const wxSizerFlags & *flags*)

Insert a child into the sizer before any existing item at *index*.

See Add() for the meaning of the other parameters.

wxSizerItem* wxSizer::Insert (size_t *index,* wxSizer * *sizer,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Insert a child into the sizer before any existing item at *index*.

See Add() for the meaning of the other parameters.

wxSizerItem* wxSizer::Insert (size_t *index,* int *width,* int *height,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Insert a child into the sizer before any existing item at *index*.

See Add() for the meaning of the other parameters.

wxSizerItem* wxSizer::Insert (size_t *index,* int *width,* int *height,* const wxSizerFlags & *flags*)

Insert a child into the sizer before any existing item at *index*.

See Add() for the meaning of the other parameters.

wxSizerItem* wxSizer::Insert (size_t *index,* wxSizerItem * *item*)

wxSizerItem* wxSizer::InsertSpacer (size_t *index,* int *size*)

Inserts non-stretchable space to the sizer.

More readable way of calling wxSizer::Insert(index, size, size).

wxSizerItem* wxSizer::InsertStretchSpacer (size_t *index*, int *prop* = 1)

Inserts stretchable space to the sizer.

More readable way of calling wxSizer::Insert(0, 0, prop).

bool wxSizer::IsEmpty ( ) const

Return true if the sizer has no elements.

See also

GetItemCount()

bool wxSizer::IsShown (wxWindow * *window*) const

Returns true if the *window* is shown.

See also

Hide(), Show(), wxSizerItem::IsShown()

bool wxSizer::IsShown (wxSizer * *sizer*) const

Returns true if the *sizer* is shown.

See also

Hide(), Show(), wxSizerItem::IsShown()

bool wxSizer::IsShown (size_t *index*) const

Returns true if the item at *index* is shown.

See also

Hide(), Show(), wxSizerItem::IsShown()

virtual void wxSizer::Layout ( ) [virtual]

Call this to force layout of the children anew, e.g. after having added a child to or removed a child (window, other sizer or space) from the sizer while keeping the current dimension.

wxSizerItem* wxSizer::Prepend (wxWindow * *window*, const wxSizerFlags & *flags*)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

wxSizerItem* wxSizer::Prepend (wxWindow * *window*, int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

wxSizerItem* wxSizer::Prepend (wxSizer * *sizer,* const wxSizerFlags & *flags*)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

wxSizerItem* wxSizer::Prepend (wxSizer * *sizer,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

wxSizerItem* wxSizer::Prepend (int *width,* int *height,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

wxSizerItem* wxSizer::Prepend (int *width,* int *height,* const wxSizerFlags & *flags*)

Same as Add(), but prepends the items to the beginning of the list of items (windows, subsizers or spaces) owned by this sizer.

wxSizerItem* wxSizer::Prepend (wxSizerItem * *item*)

wxSizerItem* wxSizer::PrependSpacer (int *size*)

Prepends non-stretchable space to the sizer.

More readable way of calling wxSizer::Prepend(size, size, 0).

wxSizerItem* wxSizer::PrependStretchSpacer (int *prop* = 1)

Prepends stretchable space to the sizer.

More readable way of calling wxSizer::Prepend(0, 0, prop).

virtual void wxSizer::RecalcSizes ( ) [pure virtual]

This method is abstract and has to be overwritten by any derived class.

Here, the sizer will do the actual calculation of its children's positions and sizes.

Implemented in wxBoxSizer, wxStaticBoxSizer, wxGridSizer, wxFlexGridSizer, wxStdDialogButtonSizer, wxGridBagSizer, and wxWrapSizer.

virtual bool wxSizer::Remove (wxWindow * *window*) [virtual]

Removes a child window from the sizer, but does **not** destroy it (because windows are owned by their parent window, not the sizer).

Deprecated The overload of this method taking a wxWindow* parameter is deprecated as it does not destroy the window as would usually be expected from Remove(). You should use Detach() in new code instead. There is currently no wxSizer method that will both detach and destroy a wxWindow item.

Note

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after removing a child from the sizer.

Returns

true if the child item was found and removed, false otherwise.

virtual bool wxSizer::Remove (wxSizer * *sizer*) [virtual]

Removes a sizer child from the sizer and destroys it.

Note

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after removing a child from the sizer.

Parameters

sizer	The wxSizer to be removed.

Returns

true if the child item was found and removed, false otherwise.

virtual bool wxSizer::Remove (int *index*) [virtual]

Removes a child from the sizer and destroys it if it is a sizer or a spacer, but not if it is a window (because windows are owned by their parent window, not the sizer).

Note

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after removing a child from the sizer.

Parameters

index	The position of the child in the sizer, e.g. 0 for the first item.

Returns

true if the child item was found and removed, false otherwise.

virtual bool wxSizer::Replace (wxWindow * *oldwin*, wxWindow * *newwin*, bool *recursive* = false) [virtual]

Detaches the given *oldwin* from the sizer and replaces it with the given *newwin*.

The detached child window is **not** deleted (because windows are owned by their parent window, not the sizer).

Use parameter *recursive* to search the given element recursively in subsizers.

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after replacing a child from the sizer.

Returns true if the child item was found and removed, false otherwise.

virtual bool wxSizer::Replace (wxSizer * *oldsz*, wxSizer * *newsz*, bool *recursive* = false) [virtual]

Detaches the given *oldsz* from the sizer and replaces it with the given *newsz*.

The detached child sizer is deleted.

Use parameter *recursive* to search the given element recursively in subsizers.

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after replacing a child from the sizer.

Returns true if the child item was found and removed, false otherwise.

virtual bool wxSizer::Replace (size_t *index*, wxSizerItem * *newitem*) [virtual]

Detaches the given item at position *index* from the sizer and replaces it with the given wxSizerItem *newitem*.

The detached child is deleted **only** if it is a sizer or a spacer (but not if it is a wxWindow because windows are owned by their parent window, not the sizer).

This method does not cause any layout or resizing to take place, call Layout() to update the layout "on screen" after replacing a child from the sizer.

Returns true if the child item was found and removed, false otherwise.

void wxSizer::SetContainingWindow (wxWindow * *window*)

Set the window this sizer is used in.

void wxSizer::SetDimension (int *x*, int *y*, int *width*, int *height*)

Call this to force the sizer to take the given dimension and thus force the items owned by the sizer to resize themselves according to the rules defined by the parameter in the Add() and Prepend() methods.

void wxSizer::SetDimension (const wxPoint & *pos*, const wxSize & *size*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxSizer::SetItemMinSize (wxWindow * *window*, int *width*, int *height*)

Set an item's minimum size by window, sizer, or position.

This function enables an application to set the size of an item after initial creation.

The *window* or *sizer* will be found recursively in the sizer's descendants.

See also

wxSizerItem::SetMinSize()

Returns

true if the minimal size was successfully set or false if the item was not found.

bool wxSizer::SetItemMinSize (wxWindow * *window,* const wxSize & *size*)

Set an item's minimum size by window, sizer, or position.

This function enables an application to set the size of an item after initial creation.

The *window* or *sizer* will be found recursively in the sizer's descendants.

See also

wxSizerItem::SetMinSize()

Returns

true if the minimal size was successfully set or false if the item was not found.

bool wxSizer::SetItemMinSize (wxSizer * *sizer,* int *width,* int *height*)

Set an item's minimum size by window, sizer, or position.

This function enables an application to set the size of an item after initial creation.

The *window* or *sizer* will be found recursively in the sizer's descendants.

See also

wxSizerItem::SetMinSize()

Returns

true if the minimal size was successfully set or false if the item was not found.

bool wxSizer::SetItemMinSize (wxSizer * *sizer,* const wxSize & *size*)

Set an item's minimum size by window, sizer, or position.

This function enables an application to set the size of an item after initial creation.

The *window* or *sizer* will be found recursively in the sizer's descendants.

See also

wxSizerItem::SetMinSize()

Returns

true if the minimal size was successfully set or false if the item was not found.

bool wxSizer::SetItemMinSize (size_t *index,* int *width,* int *height*)

Set an item's minimum size by window, sizer, or position.

This function enables an application to set the size of an item after initial creation.

The *window* or *sizer* will be found recursively in the sizer's descendants.

See also

wxSizerItem::SetMinSize()

Returns

true if the minimal size was successfully set or false if the item was not found.

bool wxSizer::SetItemMinSize (size_t *index*, const wxSize & *size*)

Set an item's minimum size by window, sizer, or position.

This function enables an application to set the size of an item after initial creation.

The *window* or *sizer* will be found recursively in the sizer's descendants.

See also

wxSizerItem::SetMinSize()

Returns

true if the minimal size was successfully set or false if the item was not found.

void wxSizer::SetMinSize (const wxSize & *size*)

Call this to give the sizer a minimal size.

Normally, the sizer will calculate its minimal size based purely on how much space its children need. After calling this method GetMinSize() will return either the minimal size as requested by its children or the minimal size set here, depending on which is bigger.

void wxSizer::SetMinSize (int *width*, int *height*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxSizer::SetSizeHints (wxWindow * *window*)

This method first calls Fit() and then wxTopLevelWindow::SetSizeHints() on the *window* passed to it.

This only makes sense when *window* is actually a wxTopLevelWindow such as a wxFrame or a wxDialog, since SetSizeHints only has any effect in these classes. It does nothing in normal windows or controls.

This method is implicitly used by wxWindow::SetSizerAndFit() which is commonly invoked in the constructor of a toplevel window itself (see the sample in the description of wxBoxSizer) if the toplevel window is resizable.

void wxSizer::SetVirtualSizeHints (wxWindow * *window*)

Tell the sizer to set the minimal size of the *window* virtual area to match the sizer's minimal size.

For windows with managed scrollbars this will set them appropriately.

Deprecated This is exactly the same as FitInside() in wxWidgets 2.9 and later, please replace calls to it with FitInside().

See also

wxScrolled::SetScrollbars()

bool wxSizer::Show (wxWindow * *window*, bool *show* = `true`, bool *recursive* = `false`)

Shows or hides the *window*.

To make a sizer item disappear or reappear, use Show() followed by Layout().

Use parameter *recursive* to show or hide elements found in subsizers.

Returns true if the child item was found, false otherwise.

See also

Hide(), IsShown()

bool wxSizer::Show (wxSizer * *sizer*, bool *show* = `true`, bool *recursive* = `false`)

Shows or hides *sizer*.

To make a sizer item disappear or reappear, use Show() followed by Layout().

Use parameter *recursive* to show or hide elements found in subsizers.

Returns true if the child item was found, false otherwise.

See also

Hide(), IsShown()

bool wxSizer::Show (size_t *index*, bool *show* = `true`)

Shows the item at *index*.

To make a sizer item disappear or reappear, use Show() followed by Layout().

Returns true if the child item was found, false otherwise.

See also

Hide(), IsShown()

virtual void wxSizer::ShowItems (bool *show*) `[virtual]`

Show or hide all items managed by the sizer.

21.687 wxSizerFlags Class Reference

`#include <wx/sizer.h>`

21.687.1 Detailed Description

Container for sizer items flags providing readable names for them.

Normally, when you add an item to a sizer via wxSizer::Add, you have to specify a lot of flags and parameters which can be unwieldy. This is where wxSizerFlags comes in: it allows you to specify all parameters using the named methods instead. For example, instead of

```
sizer->Add(ctrl, 0, wxEXPAND | wxALL, 10);
```

you can now write

```
sizer->Add(ctrl, wxSizerFlags().Expand().Border(wxALL, 10));
```

This is more readable and also allows you to create wxSizerFlags objects which can be reused for several sizer items.

```
wxSizerFlags flagsExpand(1);
    flagsExpand.Expand().Border(wxALL, 10);

    sizer->Add(ctrl1, flagsExpand);
    sizer->Add(ctrl2, flagsExpand);
```

Note that by specification, all methods of wxSizerFlags return the wxSizerFlags object itself to allowing chaining multiple methods calls like in the examples above.

Library: wxCore

Category: Window Layout

See also

wxSizer

Public Member Functions

- wxSizerFlags (int proportion=0)

  Creates the wxSizer with the proportion specified by proportion.
- wxSizerFlags & Align (int alignment)

  Sets the alignment of this wxSizerFlags to align.
- wxSizerFlags & Border (int direction, int borderinpixels)

  Sets the wxSizerFlags to have a border of a number of pixels specified by borderinpixels with the directions specified by direction.
- wxSizerFlags & Border (int direction=wxALL)

  Sets the wxSizerFlags to have a border with size as returned by GetDefaultBorder().
- wxSizerFlags & Bottom ()

  Aligns the object to the bottom, similar for `Align(wxALIGN_BOTTOM)`.
- wxSizerFlags & Center ()

  Sets the object of the wxSizerFlags to center itself in the area it is given.
- wxSizerFlags & Centre ()

  Center() for people with the other dialect of English.
- wxSizerFlags & DoubleBorder (int direction=wxALL)

  Sets the border in the given direction having twice the default border size.
- wxSizerFlags & DoubleHorzBorder ()

  Sets the border in left and right directions having twice the default border size.
- wxSizerFlags & Expand ()

  Sets the object of the wxSizerFlags to expand to fill as much area as it can.
- wxSizerFlags & FixedMinSize ()

  Set the `wxFIXED_MINSIZE` flag which indicates that the initial size of the window should be also set as its minimal size.
- wxSizerFlags & ReserveSpaceEvenIfHidden ()

  Set the `wxRESERVE_SPACE_EVEN_IF_HIDDEN` flag.
- wxSizerFlags & Left ()

Aligns the object to the left, similar for `Align(wxALIGN_LEFT)`.

- wxSizerFlags & Proportion (int proportion)

  Sets the proportion of this wxSizerFlags to proportion.

- wxSizerFlags & Right ()

  Aligns the object to the right, similar for `Align(wxALIGN_RIGHT)`.

- wxSizerFlags & Shaped ()

  Set the `wx_SHAPED` flag which indicates that the elements should always keep the fixed width to height ratio equal to its original value.

- wxSizerFlags & Top ()

  Aligns the object to the top, similar for `Align(wxALIGN_TOP)`.

- wxSizerFlags & TripleBorder (int direction=wxALL)

  Sets the border in the given direction having thrice the default border size.

Static Public Member Functions

- static int GetDefaultBorder ()

  Returns the border used by default in Border() method.

21.687.2 Constructor & Destructor Documentation

wxSizerFlags::wxSizerFlags (int *proportion* = `0`)

Creates the wxSizer with the proportion specified by *proportion*.

21.687.3 Member Function Documentation

wxSizerFlags& wxSizerFlags::Align (int *alignment*)

Sets the alignment of this wxSizerFlags to *align*.

This method replaces the previously set alignment with the specified one.

Parameters

alignment	Combination of `wxALIGN_XXX` bit masks.

See also

Top(), Left(), Right(), Bottom(), Centre()

wxSizerFlags& wxSizerFlags::Border (int *direction*, int *borderinpixels*)

Sets the wxSizerFlags to have a border of a number of pixels specified by *borderinpixels* with the directions specified by *direction*.

wxSizerFlags& wxSizerFlags::Border (int *direction* = wxALL)

Sets the wxSizerFlags to have a border with size as returned by GetDefaultBorder().

Parameters

direction	Direction(s) to apply the border in.

wxSizerFlags& wxSizerFlags::Bottom ( )

Aligns the object to the bottom, similar for `Align(wxALIGN_BOTTOM)`.

Unlike Align(), this method doesn't change the horizontal alignment of the item.

wxSizerFlags& wxSizerFlags::Center ( )

Sets the object of the wxSizerFlags to center itself in the area it is given.

wxSizerFlags& wxSizerFlags::Centre ( )

Center() for people with the other dialect of English.

wxSizerFlags& wxSizerFlags::DoubleBorder (int *direction* = **wxALL**)

Sets the border in the given *direction* having twice the default border size.

wxSizerFlags& wxSizerFlags::DoubleHorzBorder ( )

Sets the border in left and right directions having twice the default border size.

wxSizerFlags& wxSizerFlags::Expand ( )

Sets the object of the wxSizerFlags to expand to fill as much area as it can.

wxSizerFlags& wxSizerFlags::FixedMinSize ( )

Set the `wxFIXED_MINSIZE` flag which indicates that the initial size of the window should be also set as its minimal size.

static int wxSizerFlags::GetDefaultBorder ( ) [static]

Returns the border used by default in Border() method.

wxSizerFlags& wxSizerFlags::Left ( )

Aligns the object to the left, similar for `Align(wxALIGN_LEFT)`.

Unlike Align(), this method doesn't change the vertical alignment of the item.

wxSizerFlags& wxSizerFlags::Proportion (int *proportion*)

Sets the proportion of this wxSizerFlags to *proportion*.

wxSizerFlags& wxSizerFlags::ReserveSpaceEvenIfHidden ( )

Set the `wxRESERVE_SPACE_EVEN_IF_HIDDEN` flag.

Normally wxSizers don't allocate space for hidden windows or other items. This flag overrides this behaviour so that sufficient space is allocated for the window even if it isn't visible. This makes it possible to dynamically show and hide controls without resizing parent dialog, for example.

Since

2.8.8

wxSizerFlags& wxSizerFlags::Right ( )

Aligns the object to the right, similar for `Align(wxALIGN_RIGHT)`.

Unlike Align(), this method doesn't change the vertical alignment of the item.

wxSizerFlags& wxSizerFlags::Shaped ( )

Set the `wx_SHAPED` flag which indicates that the elements should always keep the fixed width to height ratio equal to its original value.

wxSizerFlags& wxSizerFlags::Top ( )

Aligns the object to the top, similar for `Align(wxALIGN_TOP)`.

Unlike Align(), this method doesn't change the horizontal alignment of the item.

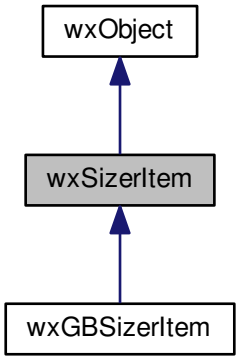
wxSizerFlags& wxSizerFlags::TripleBorder (int *direction* = wxALL)

Sets the border in the given *direction* having thrice the default border size.

21.688 wxSizerItem Class Reference

`#include <wx/sizer.h>`

Inheritance diagram for wxSizerItem:



21.688.1 Detailed Description

The wxSizerItem class is used to track the position, size and other attributes of each item managed by a wxSizer.

It is not usually necessary to use this class because the sizer elements can also be identified by their positions or window or sizer pointers but sometimes it may be more convenient to use it directly.

Library: wxCore

Category: Window Layout

Public Member Functions

- wxSizerItem (int width, int height, int proportion=0, int flag=0, int border=0, wxObject ∗userData=NULL)

  Construct a sizer item for tracking a spacer.
- virtual ∼wxSizerItem ()

  Deletes the user data and subsizer, if any.
- void AssignWindow (wxWindow ∗window)

  Set the window to be tracked by this item.
- void AssignSizer (wxSizer ∗sizer)

  Set the sizer tracked by this item.
- virtual wxSize CalcMin ()

  Calculates the minimum desired size for the item, including any space needed by borders.
- virtual void DeleteWindows ()

  Destroy the window or the windows in a subsizer, depending on the type of item.
- void DetachSizer ()

  Enable deleting the SizerItem without destroying the contained sizer.
- int GetBorder () const

  Return the border attribute.
- int GetFlag () const

Return the flags attribute.

- int GetId () const

  Return the numeric id of wxSizerItem, or `wxID_NONE` if the id has not been set.

- wxSize GetMinSize () const

  Get the minimum size needed for the item.

- void SetMinSize (const wxSize &size)

  Sets the minimum size to be allocated for this item.

- void SetMinSize (int x, int y)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- wxPoint GetPosition () const

  What is the current position of the item, as set in the last Layout.

- int GetProportion () const

  Get the proportion item attribute.

- float GetRatio () const

  Get the ration item attribute.

- virtual wxRect GetRect ()

  Get the rectangle of the item on the parent window, excluding borders.

- virtual wxSize GetSize () const

  Get the current size of the item, as set in the last Layout.

- wxSizer * GetSizer () const

  If this item is tracking a sizer, return it.

- wxSize GetSpacer () const

  If this item is tracking a spacer, return its size.

- wxObject * GetUserData () const

  Get the userData item attribute.

- wxWindow * GetWindow () const

  If this item is tracking a window then return it.

- bool IsShown () const

  Returns true if this item is a window or a spacer and it is shown or if this item is a sizer and not all of its elements are hidden.

- bool IsSizer () const

  Is this item a sizer?

- bool IsSpacer () const

  Is this item a spacer?

- bool IsWindow () const

  Is this item a window?

- void SetBorder (int border)

  Set the border item attribute.

- virtual void SetDimension (const wxPoint &pos, const wxSize &size)

  Set the position and size of the space allocated to the sizer, and adjust the position and size of the item to be within that space taking alignment and borders into account.

- void SetFlag (int flag)

  Set the flag item attribute.

- void SetId (int id)

  Sets the numeric id of the wxSizerItem to id.

- void SetInitSize (int x, int y)

- void SetProportion (int proportion)

  Set the proportion item attribute.

- void SetSizer (wxSizer *sizer)

  Set the sizer tracked by this item.

- void SetSpacer (const wxSize &size)

  Set the size of the spacer tracked by this item.
- void SetUserData (wxObject *userData)
- void SetWindow (wxWindow *window)

  Set the window to be tracked by this item.
- void Show (bool show)

  Set the show item attribute, which sizers use to determine if the item is to be made part of the layout or not.

- wxSizerItem (wxWindow *window, const wxSizerFlags &flags)

  Construct a sizer item for tracking a window.
- wxSizerItem (wxWindow *window, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Construct a sizer item for tracking a window.

- wxSizerItem (wxSizer *sizer, const wxSizerFlags &flags)

  Construct a sizer item for tracking a subsizer.
- wxSizerItem (wxSizer *sizer, int proportion=0, int flag=0, int border=0, wxObject *userData=NULL)

  Construct a sizer item for tracking a subsizer.

- void AssignSpacer (const wxSize &size)

  Set the size of the spacer tracked by this item.
- void AssignSpacer (int w, int h)

  Set the size of the spacer tracked by this item.

- void SetRatio (int width, int height)

  Set the ratio item attribute.
- void SetRatio (wxSize size)

  Set the ratio item attribute.
- void SetRatio (float ratio)

  Set the ratio item attribute.

Additional Inherited Members

21.688.2 Constructor & Destructor Documentation

wxSizerItem::wxSizerItem (int *width*, int *height*, int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Construct a sizer item for tracking a spacer.

wxSizerItem::wxSizerItem (wxWindow * *window*, const wxSizerFlags & *flags*)

Construct a sizer item for tracking a window.

wxSizerItem::wxSizerItem (wxWindow * *window*, int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Construct a sizer item for tracking a window.

wxSizerItem::wxSizerItem (wxSizer * *sizer*, const wxSizerFlags & *flags*)

Construct a sizer item for tracking a subsizer.

wxSizerItem::wxSizerItem (wxSizer * *sizer,* int *proportion* = 0, int *flag* = 0, int *border* = 0, wxObject * *userData* = NULL)

Construct a sizer item for tracking a subsizer.

virtual wxSizerItem::~wxSizerItem ( ) [virtual]

Deletes the user data and subsizer, if any.

21.688.3 Member Function Documentation

void wxSizerItem::AssignSizer (wxSizer * *sizer*)

Set the sizer tracked by this item.

Old sizer, if any, is deleted.

void wxSizerItem::AssignSpacer (const wxSize & *size*)

Set the size of the spacer tracked by this item.

Old spacer, if any, is deleted.

void wxSizerItem::AssignSpacer (int *w,* int *h*)

Set the size of the spacer tracked by this item.

Old spacer, if any, is deleted.

void wxSizerItem::AssignWindow (wxWindow * *window*)

Set the window to be tracked by this item.

The old window isn't deleted as it is now owned by the sizer item.

virtual wxSize wxSizerItem::CalcMin ( ) [virtual]

Calculates the minimum desired size for the item, including any space needed by borders.

virtual void wxSizerItem::DeleteWindows ( ) [virtual]

Destroy the window or the windows in a subsizer, depending on the type of item.

void wxSizerItem::DetachSizer ( )

Enable deleting the SizerItem without destroying the contained sizer.

int wxSizerItem::GetBorder ( ) const

Return the border attribute.

int wxSizerItem::GetFlag ( ) const

Return the flags attribute.

See wxSizer flags list for details.

int wxSizerItem::GetId ( ) const

Return the numeric id of wxSizerItem, or `wxID_NONE` if the id has not been set.

wxSize wxSizerItem::GetMinSize ( ) const

Get the minimum size needed for the item.

wxPoint wxSizerItem::GetPosition ( ) const

What is the current position of the item, as set in the last Layout.

int wxSizerItem::GetProportion ( ) const

Get the proportion item attribute.

float wxSizerItem::GetRatio ( ) const

Get the ration item attribute.

virtual wxRect wxSizerItem::GetRect ( ) [virtual]

Get the rectangle of the item on the parent window, excluding borders.

virtual wxSize wxSizerItem::GetSize ( ) const [virtual]

Get the current size of the item, as set in the last Layout.

wxSizer* wxSizerItem::GetSizer ( ) const

If this item is tracking a sizer, return it.

NULL otherwise.

wxSize wxSizerItem::GetSpacer ( ) const

If this item is tracking a spacer, return its size.

wxObject* wxSizerItem::GetUserData ( ) const

Get the userData item attribute.

wxWindow* wxSizerItem::GetWindow ( ) const

If this item is tracking a window then return it.

NULL otherwise.

bool wxSizerItem::IsShown ( ) const

Returns true if this item is a window or a spacer and it is shown or if this item is a sizer and not all of its elements are hidden.

In other words, for sizer items, all of the child elements must be hidden for the sizer itself to be considered hidden.

As an exception, if the `wxRESERVE_SPACE_EVEN_IF_HIDDEN` flag was used for this sizer item, then IsShown() always returns true for it (see wxSizerFlags::ReserveSpaceEvenIfHidden()).

bool wxSizerItem::IsSizer ( ) const

Is this item a sizer?

bool wxSizerItem::IsSpacer ( ) const

Is this item a spacer?

bool wxSizerItem::IsWindow ( ) const

Is this item a window?

void wxSizerItem::SetBorder (int *border*)

Set the border item attribute.

virtual void wxSizerItem::SetDimension (const wxPoint & *pos*, const wxSize & *size*) [virtual]

Set the position and size of the space allocated to the sizer, and adjust the position and size of the item to be within that space taking alignment and borders into account.

void wxSizerItem::SetFlag (int *flag*)

Set the flag item attribute.

void wxSizerItem::SetId (int *id*)

Sets the numeric id of the wxSizerItem to *id*.

void wxSizerItem::SetInitSize (int *x*, int *y*)

Todo docme.

void wxSizerItem::SetMinSize (const wxSize & *size*)

Sets the minimum size to be allocated for this item.

If this item is a window, the *size* is also passed to wxWindow::SetMinSize().

void wxSizerItem::SetMinSize (int *x,* int *y*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxSizerItem::SetProportion (int *proportion*)

Set the proportion item attribute.

void wxSizerItem::SetRatio (int *width,* int *height*)

Set the ratio item attribute.

void wxSizerItem::SetRatio (wxSize *size*)

Set the ratio item attribute.

void wxSizerItem::SetRatio (float *ratio*)

Set the ratio item attribute.

void wxSizerItem::SetSizer (wxSizer * *sizer*)

Set the sizer tracked by this item.

Deprecated This function does not free the old sizer which may result in memory leaks, use AssignSizer() which does free it instead.

void wxSizerItem::SetSpacer (const wxSize & *size*)

Set the size of the spacer tracked by this item.

Deprecated This function does not free the old spacer which may result in memory leaks, use AssignSpacer() which does free it instead.

void wxSizerItem::SetUserData (wxObject * *userData*)

void wxSizerItem::SetWindow (wxWindow * *window*)

Set the window to be tracked by this item.

Deprecated

Todo provide deprecation description

void wxSizerItem::Show (bool *show*)

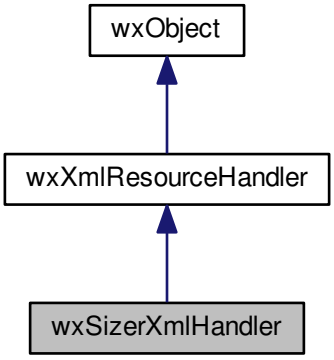
Set the show item attribute, which sizers use to determine if the item is to be made part of the layout or not.

If the item is tracking a window then it is shown or hidden as needed.

21.689 wxSizerXmlHandler Class Reference

`#include <wx/xrc/xh_sizer.h>`

Inheritance diagram for wxSizerXmlHandler:



Public Member Functions

- wxSizerXmlHandler ()

  Constructor.
- virtual wxObject * DoCreateResource ()

  Creates a sizer, sizeritem or spacer object, depending on the current handled node.
- virtual bool CanHandle (wxXmlNode *node)

  Returns true if the given node can be handled by this class.

Protected Member Functions

- virtual wxSizer * DoCreateSizer (const wxString &name)

  Creates an object of type wxSizer from the XML node content.
- virtual bool IsSizerNode (wxXmlNode *node) const

  Used by CanHandle() to know if the given node contains a sizer supported by this class.

Additional Inherited Members

21.689.1 Constructor & Destructor Documentation

wxSizerXmlHandler::wxSizerXmlHandler ( )

Constructor.

Initializes the attributes and adds the supported styles.

21.689.2 Member Function Documentation

virtual bool wxSizerXmlHandler::CanHandle (wxXmlNode * *node*) [virtual]

Returns true if the given node can be handled by this class.

If the node concerns a sizer object, the method IsSizerNode is called to know if the class is managed or not. If the node concerns a sizer item or a spacer, true is returned. Otherwise false is returned.

See also

wxXmlResourceHandler::CanHandle().

Implements wxXmlResourceHandler.

virtual wxObject* wxSizerXmlHandler::DoCreateResource ( ) [virtual]

Creates a sizer, sizeritem or spacer object, depending on the current handled node.

See also

wxXmlResourceHandler::DoCreateResource().

Implements wxXmlResourceHandler.

virtual wxSizer* wxSizerXmlHandler::DoCreateSizer (const wxString & *name*) [protected], [virtual]

Creates an object of type wxSizer from the XML node content.

This virtual method can be overridden to add support for custom sizer classes to the derived handler.

Notice that if you override this method you would typically overload IsSizerNode() as well.

Example of use of this method:

```
class MySizerXmlHandler : public wxSizerXmlHandler
{
    ...

protected:
    bool IsSizerNode(wxXmlNode *node) const
    {
        return IsOfClass(node, "MySizer") ||
               wxSizerXmlHandler::IsSizerNode(node));
    }

    void DoCreateSizer(const wxString& name)
    {
        if ( name == "MySizer" )
            return Handle_MySizer();
        else
            return wxSizerXmlHandler::DoCreateSizer(name);
    }

private:
    wxSizer* Handle_MySizer()
    {
        // Create your own sizer here from XRC content (see
        // wxXmlResource methods) and return the instance.
    }
};
```

Since

2.9.2

virtual bool wxSizerXmlHandler::IsSizerNode (wxXmlNode * *node*) const `[protected], [virtual]`

Used by CanHandle() to know if the given node contains a sizer supported by this class.

This method should be overridden to allow this handler to be used for the custom sizer types.

See the example in DoCreateSizer() description for how it can be used.

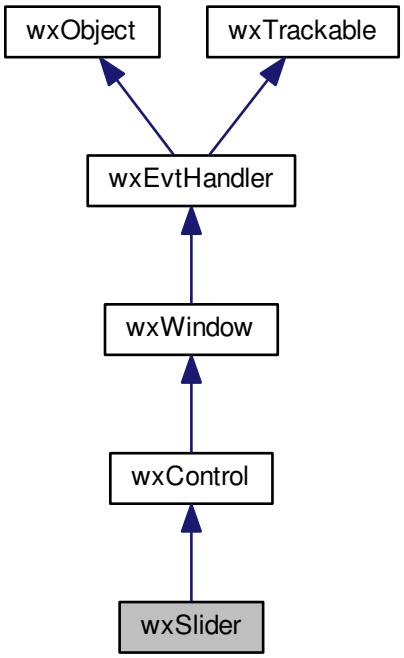
Since

2.9.2

21.690 wxSlider Class Reference

`#include <wx/slider.h>`

Inheritance diagram for wxSlider:



21.690.1 Detailed Description

A slider is a control with a handle which can be pulled back and forth to change the value.

On Windows, the track bar control is used.

Slider generates the same events as wxScrollBar but in practice the most convenient way to process wxSlider updates is by handling the slider-specific `wxEVT_SLIDER` event which carries wxCommandEvent containing just the latest slider position.

Styles

This class supports the following styles:

- wxSL_HORIZONTAL: Displays the slider horizontally (this is the default).
- wxSL_VERTICAL: Displays the slider vertically.
- wxSL_AUTOTICKS: Displays tick marks. Windows only.
- wxSL_MIN_MAX_LABELS: Displays minimum, maximum labels (new since wxWidgets 2.9.1).
- wxSL_VALUE_LABEL: Displays value label (new since wxWidgets 2.9.1).
- wxSL_LABELS: Displays minimum, maximum and value labels (same as wxSL_VALUE_LABEL and wxSL↩ _MIN_MAX_LABELS together).
- wxSL_LEFT: Displays ticks on the left and forces the slider to be vertical.
- wxSL_RIGHT: Displays ticks on the right and forces the slider to be vertical.
- wxSL_TOP: Displays ticks on the top.
- wxSL_BOTTOM: Displays ticks on the bottom (this is the default).
- wxSL_SELRANGE: Allows the user to select a range on the slider. Windows only.
- wxSL_INVERSE: Inverses the minimum and maximum endpoints on the slider. Not compatible with wxSL↩ _SELRANGE.

Notice that `wxSL_LEFT`, `wxSL_TOP`, `wxSL_RIGHT` and `wxSL_BOTTOM` specify the position of the slider ticks in MSW implementation and that the slider labels, if any, are positioned on the opposite side. So, to have a label on the left side of a vertical slider, **wxSL_RIGHT** must be used (or none of these styles at all should be specified as left and top are default positions for the vertical and horizontal sliders respectively).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxScrollEvent& event)

Event macros for events emitted by this class: You can use EVT_COMMAND_SCROLL... macros with window IDs for when intercepting scroll events from controls, or EVT_SCROLL... macros without window IDs for intercepting scroll events from the receiving window – except for this, the macros behave exactly the same.

- EVT_SCROLL(func): Process all scroll events.
- EVT_SCROLL_TOP(func): Process `wxEVT_SCROLL_TOP` scroll-to-top events (minimum position).
- EVT_SCROLL_BOTTOM(func): Process `wxEVT_SCROLL_BOTTOM` scroll-to-bottom events (maximum position).
- EVT_SCROLL_LINEUP(func): Process `wxEVT_SCROLL_LINEUP` line up events.
- EVT_SCROLL_LINEDOWN(func): Process `wxEVT_SCROLL_LINEDOWN` line down events.
- EVT_SCROLL_PAGEUP(func): Process `wxEVT_SCROLL_PAGEUP` page up events.
- EVT_SCROLL_PAGEDOWN(func): Process `wxEVT_SCROLL_PAGEDOWN` page down events.

- EVT_SCROLL_THUMBTRACK(func): Process `wxEVT_SCROLL_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_SCROLL_THUMBRELEASE(func): Process `wxEVT_SCROLL_THUMBRELEASE` thumb release events.
- EVT_SCROLL_CHANGED(func): Process `wxEVT_SCROLL_CHANGED` end of scrolling events (MSW only).
- EVT_COMMAND_SCROLL(id, func): Process all scroll events.
- EVT_COMMAND_SCROLL_TOP(id, func): Process `wxEVT_SCROLL_TOP` scroll-to-top events (minimum position).
- EVT_COMMAND_SCROLL_BOTTOM(id, func): Process `wxEVT_SCROLL_BOTTOM` scroll-to-bottom events (maximum position).
- EVT_COMMAND_SCROLL_LINEUP(id, func): Process `wxEVT_SCROLL_LINEUP` line up events.
- EVT_COMMAND_SCROLL_LINEDOWN(id, func): Process `wxEVT_SCROLL_LINEDOWN` line down events.
- EVT_COMMAND_SCROLL_PAGEUP(id, func): Process `wxEVT_SCROLL_PAGEUP` page up events.
- EVT_COMMAND_SCROLL_PAGEDOWN(id, func): Process `wxEVT_SCROLL_PAGEDOWN` page down events.
- EVT_COMMAND_SCROLL_THUMBTRACK(id, func): Process `wxEVT_SCROLL_THUMBTRACK` thumbtrack events (frequent events sent as the user drags the thumbtrack).
- EVT_COMMAND_SCROLL_THUMBRELEASE(func): Process `wxEVT_SCROLL_THUMBRELEASE` thumb release events.
- EVT_COMMAND_SCROLL_CHANGED(func): Process `wxEVT_SCROLL_CHANGED` end of scrolling events (MSW only).
- EVT_SLIDER(id, func): Process `wxEVT_SLIDER` which is generated after any change of wxSlider position in addition to one of the events above. Notice that the handler of this event receives a wxCommandEvent as argument and not wxScrollEvent, as all the other handlers.

21.690.2 The difference between EVT_SCROLL_THUMBRELEASE and EVT_SCROLL_CHANGED

The EVT_SCROLL_THUMBRELEASE event is only emitted when actually dragging the thumb using the mouse and releasing it (This EVT_SCROLL_THUMBRELEASE event is also followed by an EVT_SCROLL_CHANGED event).

The EVT_SCROLL_CHANGED event also occurs when using the keyboard to change the thumb position, and when clicking next to the thumb (In all these cases the EVT_SCROLL_THUMBRELEASE event does not happen). In short, the EVT_SCROLL_CHANGED event is triggered when scrolling/ moving has finished independently of the way it had started. Please see the widgets sample ("Slider" page) to see the difference between EVT_SCROLL_↩ THUMBRELEASE and EVT_SCROLL_CHANGED in action.

Library: wxCore

Category: Controls

See also

Events and Event Handling, wxScrollBar

Public Member Functions

- wxSlider ()

  Default constructor.
- wxSlider (wxWindow *parent, wxWindowID id, int value, int minValue, int maxValue, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSL_HORIZONTAL, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxSliderNameStr)

  Constructor, creating and showing a slider.
- virtual ~wxSlider ()

  Destructor, destroying the slider.
- virtual void ClearSel ()

  *Clears the selection, for a slider with the **wxSL_SELRANGE** style.*
- virtual void ClearTicks ()

  Clears the ticks.
- bool Create (wxWindow *parent, wxWindowID id, int value, int minValue, int maxValue, const wxPoint &point=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSL_HORIZONTAL, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxSliderNameStr)

  Used for two-step slider construction.
- virtual int GetLineSize () const

  Returns the line size.
- virtual int GetMax () const

  Gets the maximum slider value.
- virtual int GetMin () const

  Gets the minimum slider value.
- virtual int GetPageSize () const

  Returns the page size.
- virtual int GetSelEnd () const

  Returns the selection end point.
- virtual int GetSelStart () const

  Returns the selection start point.
- virtual int GetThumbLength () const

  Returns the thumb length.
- virtual int GetTickFreq () const

  Returns the tick frequency.
- virtual int GetValue () const

  Gets the current slider value.
- virtual void SetLineSize (int lineSize)

  Sets the line size for the slider.
- void SetMin (int minValue)

  Sets the minimum slider value.
- void SetMax (int maxValue)

  Sets the maximum slider value.
- virtual void SetPageSize (int pageSize)

  Sets the page size for the slider.
- virtual void SetRange (int minValue, int maxValue)

  Sets the minimum and maximum slider values.
- virtual void SetSelection (int startPos, int endPos)

  Sets the selection.
- virtual void SetThumbLength (int len)

  Sets the slider thumb length.
- virtual void SetTick (int tickPos)

Sets a tick position.

- virtual void SetTickFreq (int n)

  Sets the tick mark frequency and position.

- virtual void SetValue (int value)

  Sets the slider position.

Additional Inherited Members

21.690.3 Constructor & Destructor Documentation

wxSlider::wxSlider ( )

Default constructor.

wxSlider::wxSlider (wxWindow * *parent*, wxWindowID *id*, int *value*, int *minValue*, int *maxValue*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSL_HORIZONTAL, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxSliderNameStr`)

Constructor, creating and showing a slider.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
value	Initial position for the slider.
minValue	Minimum slider position.
maxValue	Maximum slider position.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxSlider.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxSlider::~wxSlider ( ) [virtual]

Destructor, destroying the slider.

21.690.4 Member Function Documentation

virtual void wxSlider::ClearSel ( ) [virtual]

Clears the selection, for a slider with the **wxSL_SELRANGE** style.

Availability: only available for the wxMSW port.

virtual void wxSlider::ClearTicks ( ) [virtual]

Clears the ticks.

Availability: only available for the wxMSW port.

bool wxSlider::Create (wxWindow * *parent*, wxWindowID *id*, int *value*, int *minValue*, int *maxValue*, const wxPoint & *point* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSL_HORIZONTAL, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxSliderNameStr`)

Used for two-step slider construction.

See wxSlider() for further details.

virtual int wxSlider::GetLineSize ( ) const `[virtual]`

Returns the line size.

See also

SetLineSize()

virtual int wxSlider::GetMax ( ) const `[virtual]`

Gets the maximum slider value.

See also

GetMin(), SetRange()

virtual int wxSlider::GetMin ( ) const `[virtual]`

Gets the minimum slider value.

See also

GetMin(), SetRange()

virtual int wxSlider::GetPageSize ( ) const `[virtual]`

Returns the page size.

See also

SetPageSize()

virtual int wxSlider::GetSelEnd ( ) const `[virtual]`

Returns the selection end point.

Availability: only available for the wxMSW port.

See also

GetSelStart(), SetSelection()

virtual int wxSlider::GetSelStart ( ) const [virtual]

Returns the selection start point.

Availability: only available for the wxMSW port.

See also

GetSelEnd(), SetSelection()

virtual int wxSlider::GetThumbLength ( ) const [virtual]

Returns the thumb length.

Availability: only available for the wxMSW port.

See also

SetThumbLength()

virtual int wxSlider::GetTickFreq ( ) const [virtual]

Returns the tick frequency.

Availability: only available for the wxMSW port.

See also

SetTickFreq()

virtual int wxSlider::GetValue ( ) const [virtual]

Gets the current slider value.

See also

GetMin(), GetMax(), SetValue()

virtual void wxSlider::SetLineSize (int *lineSize*) [virtual]

Sets the line size for the slider.

Parameters

lineSize	The number of steps the slider moves when the user moves it up or down a line.

See also

GetLineSize()

void wxSlider::SetMax (int *maxValue*)

Sets the maximum slider value.

Parameters

maxValue	The new top end of the slider range.

See also

GetMax(), SetRange()

void wxSlider::SetMin (int *minValue*)

Sets the minimum slider value.

Parameters

minValue	The new bottom end of the slider range.

See also

GetMin(), SetRange()

virtual void wxSlider::SetPageSize (int *pageSize*) `[virtual]`

Sets the page size for the slider.

Parameters

pageSize	The number of steps the slider moves when the user pages up or down.

See also

GetPageSize()

virtual void wxSlider::SetRange (int *minValue*, int *maxValue*) `[virtual]`

Sets the minimum and maximum slider values.

See also

GetMin(), GetMax()

virtual void wxSlider::SetSelection (int *startPos*, int *endPos*) `[virtual]`

Sets the selection.

Parameters

startPos	The selection start position.
endPos	The selection end position.

Availability: only available for the wxMSW port.

See also

GetSelStart(), GetSelEnd()

virtual void wxSlider::SetThumbLength (int *len*) `[virtual]`

Sets the slider thumb length.

Parameters

len	The thumb length.

Availability: only available for the wxMSW port.

See also

GetThumbLength()

virtual void wxSlider::SetTick (int *tickPos*) [virtual]

Sets a tick position.

Parameters

tickPos	The tick position.

Availability: only available for the wxMSW port.

See also

SetTickFreq()

virtual void wxSlider::SetTickFreq (int *n*) [virtual]

Sets the tick mark frequency and position.

Parameters

n	Frequency. For example, if the frequency is set to two, a tick mark is displayed for every other increment in the slider's range.

Availability: only available for the wxMSW port.

See also

GetTickFreq()

virtual void wxSlider::SetValue (int *value*) [virtual]

Sets the slider position.

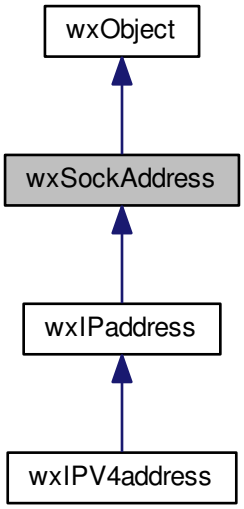
Parameters

value	The slider position.

21.691 wxSockAddress Class Reference

```
#include <wx/socket.h>
```

Inheritance diagram for wxSockAddress:



21.691.1 Detailed Description

You are unlikely to need to use this class: only wxSocketBase uses it.

Library: wxNet

Category: Networking

See also

wxSocketBase, wxIPaddress, wxIPV4address

Public Member Functions

- wxSockAddress ()

  Default constructor.
- virtual ~wxSockAddress ()

  Default destructor.
- virtual void Clear ()

  Delete all information about the address.
- int SockAddrLen ()

  Returns the length of the socket address.
- const sockaddr * GetAddressData () const

  Returns the pointer to the low-level representation of the address.
- int GetAddressDataLen () const

  Returns the length of the buffer retrieved by GetAddressData().

Additional Inherited Members

21.691.2 Constructor & Destructor Documentation

wxSockAddress::wxSockAddress ( )

Default constructor.

virtual wxSockAddress::~wxSockAddress ( ) [virtual]

Default destructor.

21.691.3 Member Function Documentation

virtual void wxSockAddress::Clear ( ) [virtual]

Delete all information about the address.

const sockaddr* wxSockAddress::GetAddressData ( ) const

Returns the pointer to the low-level representation of the address.

This can be used to pass socket address information to a 3rd party library.

Returns

Pointer to a sockaddr-derived struct.

int wxSockAddress::GetAddressDataLen ( ) const

Returns the length of the buffer retrieved by GetAddressData().

Returns

The size of the sockaddr-derived struct corresponding to this address.

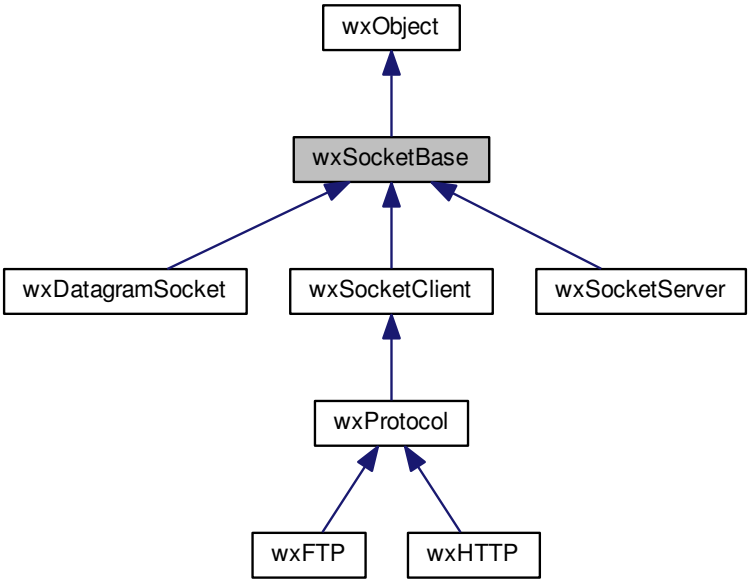
int wxSockAddress::SockAddrLen ( )

Returns the length of the socket address.

21.692 wxSocketBase Class Reference

```
#include <wx/socket.h>
```

Inheritance diagram for wxSocketBase:



21.692.1 Detailed Description

wxSocketBase is the base class for all socket-related objects, and it defines all basic IO functionality.

Note

When using wxSocket from multiple threads, even implicitly (e.g. by using wxFTP or wxHTTP in another thread) you must initialize the sockets from the main thread by calling Initialize() before creating the other ones.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSocketEvent& event)

Event macros for events emitted by this class:

- EVT_SOCKET(id, func): Process a `wxEVT_SOCKET` event. See wxSocketEventFlags and wxSocketFlags for more info.

Library: wxNet

Category: Networking

See also

wxSocketEvent, wxSocketClient, wxSocketServer, Sockets Sample, wxSocketFlags, wxSocketEventFlags, wxSocketError

Construction and Destruction

- wxSocketBase ()

  Default constructor.
- virtual ~wxSocketBase ()

  Destructor.
- bool Destroy ()

  Destroys the socket safely.
- static bool Initialize ()

  Perform the initialization needed in order to use the sockets.
- static void Shutdown ()

  Shut down the sockets.

Public Member Functions

Socket State

- bool Error () const

  Returns true if an error occurred in the last IO operation.
- virtual bool GetLocal (wxSockAddress &addr) const

  Return the local address of the socket.
- virtual bool GetPeer (wxSockAddress &addr) const

  Return the peer address field of the socket.
- long GetTimeout () const

  Return the socket timeout in seconds.
- bool IsConnected () const

  Returns true if the socket is connected.
- bool IsData ()

  Check if the socket can be currently read or written.
- bool IsDisconnected () const

  Returns true if the socket is not connected.
- bool IsOk () const

  Returns true if the socket is initialized and ready and false in other cases.
- wxUint32 LastCount () const

  Returns the number of bytes read or written by the last IO call.
- wxUint32 LastReadCount () const

  Returns the number of bytes read by the last Read() or ReadMsg() call (receive direction only).
- wxUint32 LastWriteCount () const

  Returns the number of bytes written by the last Write() or WriteMsg() call (transmit direction only).
- wxSocketError LastError () const

  Returns the last wxSocket error.
- void RestoreState ()

  Restore the previous state of the socket, as saved with SaveState().
- void SaveState ()

  Save the current state of the socket in a stack.

Basic I/O

See also: wxSocketServer::WaitForAccept(), wxSocketClient::WaitOnConnect()

- virtual bool Close ()

  Shut down the socket, disabling further transmission and reception of data and disable events for the socket and frees the associated system resources.
- void ShutdownOutput ()

  Shuts down the writing end of the socket.
- wxSocketBase & Discard ()

  Delete all bytes in the incoming queue.
- wxSocketFlags GetFlags () const

Returns current IO flags, as set with SetFlags()

- void InterruptWait ()

  Use this function to interrupt any wait operation currently in progress.
- wxSocketBase & Peek (void *buffer, wxUint32 nbytes)

  Peek into the socket by copying the next bytes which would be read by Read() into the provided buffer.
- wxSocketBase & Read (void *buffer, wxUint32 nbytes)

  Read up to the given number of bytes from the socket.
- wxSocketBase & ReadMsg (void *buffer, wxUint32 nbytes)

  Receive a message sent by WriteMsg().
- void SetFlags (wxSocketFlags flags)

  Use SetFlags to customize IO operation for this socket.
- virtual bool SetLocal (const wxIPV4address &local)

  Set the local address and port to use.
- void SetTimeout (long seconds)

  Set the default socket timeout in seconds.
- wxSocketBase & Unread (const void *buffer, wxUint32 nbytes)

  Put the specified data into the input queue.
- bool Wait (long seconds=-1, long millisecond=0)

  Wait for any socket event.
- bool WaitForLost (long seconds=-1, long millisecond=0)

  Wait until the connection is lost.
- bool WaitForRead (long seconds=-1, long millisecond=0)

  Wait until the socket is readable.
- bool WaitForWrite (long seconds=-1, long millisecond=0)

  Wait until the socket becomes writable.
- wxSocketBase & Write (const void *buffer, wxUint32 nbytes)

  Write up to the given number of bytes to the socket.
- wxSocketBase & WriteMsg (const void *buffer, wxUint32 nbytes)

  Sends a buffer which can be read using ReadMsg().

Handling Socket Events

- void * GetClientData () const

  Returns a pointer of the client data for this socket, as set with SetClientData()
- void Notify (bool notify)

  According to the notify value, this function enables or disables socket events.
- void SetClientData (void *data)

  Sets user-supplied client data for this socket.
- void SetEventHandler (wxEvtHandler &handler, int id=-1)

  Sets an event handler to be called when a socket event occurs.
- void SetNotify (wxSocketEventFlags flags)

  Specifies which socket events are to be sent to the event handler.
- wxSOCKET_T GetSocket () const

  Returns the native socket descriptor.

Additional Inherited Members

21.692.2 Constructor & Destructor Documentation

wxSocketBase::wxSocketBase ( )

Default constructor.

Don't use it directly; instead, use wxSocketClient to construct a socket client, or wxSocketServer to construct a socket server.

virtual wxSocketBase::~wxSocketBase ( ) [virtual]

Destructor.

Do not destroy a socket using the delete operator directly; use Destroy() instead. Also, do not create socket objects in the stack.

21.692.3 Member Function Documentation

virtual bool wxSocketBase::Close ( ) [virtual]

Shut down the socket, disabling further transmission and reception of data and disable events for the socket and frees the associated system resources.

Upon socket destruction, Close() is automatically called, so in most cases you won't need to do it yourself, unless you explicitly want to shut down the socket, typically to notify the peer that you are closing the connection.

Remarks

Although Close() immediately disables events for the socket, it is possible that event messages may be waiting in the application's event queue. The application must therefore be prepared to handle socket event messages even after calling Close().

Reimplemented in wxFTP.

bool wxSocketBase::Destroy ( )

Destroys the socket safely.

Use this function instead of the delete operator, since otherwise socket events could reach the application even after the socket has been destroyed. To prevent this problem, this function appends the wxSocket to a list of object to be deleted on idle time, after all events have been processed. For the same reason, you should avoid creating socket objects in the stack.

Destroy() calls Close() automatically.

Returns

Always true.

wxSocketBase& wxSocketBase::Discard ( )

Delete all bytes in the incoming queue.

This function always returns immediately and its operation is not affected by IO flags.

Use LastCount() to verify the number of bytes actually discarded.

If you use Error(), it will always return false.

bool wxSocketBase::Error ( ) const

Returns true if an error occurred in the last IO operation.

Use this function to check for an error condition after one of the following calls: Discard(), Peek(), Read(), ReadMsg(), Unread(), Write(), WriteMsg().

void* wxSocketBase::GetClientData ( ) const

Returns a pointer of the client data for this socket, as set with SetClientData()

wxSocketFlags wxSocketBase::GetFlags ( ) const

Returns current IO flags, as set with SetFlags()

virtual bool wxSocketBase::GetLocal (wxSockAddress & *addr*) const [virtual]

Return the local address of the socket.

Returns

true if no error happened, false otherwise.

virtual bool wxSocketBase::GetPeer (wxSockAddress & *addr*) const [virtual]

Return the peer address field of the socket.

Returns

true if no error happened, false otherwise.

wxSOCKET_T wxSocketBase::GetSocket ( ) const

Returns the native socket descriptor.

This is intended to use with rarely used specific platform features that can only be accessed via the actual socket descriptor.

Do not use this for reading or writing data from or to the socket as this would almost surely interfere with wxSocket code logic and result in unexpected behaviour.

The socket must be successfully initialized, e.g. connected for client sockets, before this method can be called.

Returns

Returns the native socket descriptor.

Since

2.9.5

long wxSocketBase::GetTimeout ( ) const

Return the socket timeout in seconds.

The timeout can be set using SetTimeout() and is 10 minutes by default.

static bool wxSocketBase::Initialize ( ) [static]

Perform the initialization needed in order to use the sockets.

This function is called from wxSocket constructor implicitly and so normally doesn't need to be called explicitly. There is however one important exception: as this function must be called from the main (UI) thread, if you use wxSocket from multiple threads you must call Initialize() from the main thread before creating wxSocket objects in the other ones.

It is safe to call this function multiple times (only the first call does anything) but you must call Shutdown() exactly once for every call to Initialize().

This function should only be called from the main thread.

Returns

true if the sockets can be used, false if the initialization failed and sockets are not available at all.

void wxSocketBase::InterruptWait ( )

Use this function to interrupt any wait operation currently in progress.

Note that this is not intended as a regular way to interrupt a Wait call, but only as an escape mechanism for exceptional situations where it is absolutely necessary to use it, for example to abort an operation due to some exception or abnormal problem. InterruptWait is automatically called when you Close() a socket (and thus also upon socket destruction), so you don't need to use it in these cases.

See also

Wait(), WaitForLost(), WaitForRead(), WaitForWrite(), wxSocketServer::WaitForAccept(), wxSocketClient::↩ WaitOnConnect()

bool wxSocketBase::IsConnected ( ) const

Returns true if the socket is connected.

bool wxSocketBase::IsData ( )

Check if the socket can be currently read or written.

This might mean that queued data is available for reading or, for streamed sockets, that the connection has been closed, so that a read operation will complete immediately without blocking (unless the **wxSOCKET_WAITALL** flag is set, in which case the operation might still block).

bool wxSocketBase::IsDisconnected ( ) const

Returns true if the socket is not connected.

bool wxSocketBase::IsOk ( ) const

Returns true if the socket is initialized and ready and false in other cases.

Remarks

For wxSocketClient, IsOk() won't return true unless the client is connected to a server. For wxSocketServer, IsOk() will return true if the server could bind to the specified address and is already listening for new connections. IsOk() does not check for IO errors; use Error() instead for that purpose.

wxUint32 wxSocketBase::LastCount ( ) const

Returns the number of bytes read or written by the last IO call.

Use this function to get the number of bytes actually transferred after using one of the following IO calls: Discard(), Peek(), Read(), ReadMsg(), Unread(), Write(), WriteMsg().

Deprecated This function is kept mostly for backwards compatibility. Use LastReadCount() or LastWriteCount() instead. LastCount() is still needed for use with less commonly used functions: Discard(), Peek(), and Unread().

wxSocketError wxSocketBase::LastError ( ) const

Returns the last wxSocket error.

See wxSocketError .

Note

This function merely returns the last error code, but it should not be used to determine if an error has occurred (this is because successful operations do not change the LastError value). Use Error() first, in order to determine if the last IO call failed. If this returns true, use LastError() to discover the cause of the error.

wxUint32 wxSocketBase::LastReadCount ( ) const

Returns the number of bytes read by the last Read() or ReadMsg() call (receive direction only).

This function is thread-safe, in case Read() is executed in a different thread than Write(). Use LastReadCount() instead of LastCount() for this reason.

Unlike LastCount(), the functions Discard(), Peek(), and Unread() are currently not supported by LastReadCount().

Since

2.9.5

wxUint32 wxSocketBase::LastWriteCount ( ) const

Returns the number of bytes written by the last Write() or WriteMsg() call (transmit direction only).

This function is thread-safe, in case Write() is executed in a different thread than Read(). Use LastWriteCount() instead of LastCount() for this reason.

Since

2.9.5

void wxSocketBase::Notify (bool *notify*)

According to the *notify* value, this function enables or disables socket events.

If *notify* is true, the events configured with SetNotify() will be sent to the application. If *notify* is false; no events will be sent.

wxSocketBase& wxSocketBase::Peek (void * *buffer,* wxUint32 *nbytes*)

Peek into the socket by copying the next bytes which would be read by Read() into the provided buffer.

Peeking a buffer doesn't delete it from the socket input queue, i.e. calling Read() will return the same data.

Use LastCount() to verify the number of bytes actually peeked.

Use Error() to determine if the operation succeeded.

Parameters

buffer	Buffer where to put peeked data.
nbytes	Number of bytes.

Returns

Returns a reference to the current object.

Remarks

The exact behaviour of Peek() depends on the combination of flags being used. For a detailed explanation, see SetFlags()

See also

Error(), LastError(), LastCount(), SetFlags()

wxSocketBase& wxSocketBase::Read (void * *buffer*, wxUint32 *nbytes*)

Read up to the given number of bytes from the socket.

Use LastReadCount() to verify the number of bytes actually read. Use Error() to determine if the operation succeeded.

Parameters

buffer	Buffer where to put read data.
nbytes	Number of bytes.

Returns

Returns a reference to the current object.

Remarks

The exact behaviour of Read() depends on the combination of flags being used. For a detailed explanation, see SetFlags()

See also

Error(), LastError(), LastReadCount(), SetFlags()

wxSocketBase& wxSocketBase::ReadMsg (void * *buffer*, wxUint32 *nbytes*)

Receive a message sent by WriteMsg().

If the buffer passed to the function isn't big enough, the remaining bytes will be discarded. This function always waits for the buffer to be entirely filled, unless an error occurs.

Use LastReadCount() to verify the number of bytes actually read.

Use Error() to determine if the operation succeeded.

Parameters

buffer	Buffer where to put read data.
nbytes	Size of the buffer.

Returns

Returns a reference to the current object.

Remarks

ReadMsg() will behave as if the **wxSOCKET_WAITALL** flag was always set and it will always ignore the **wx↩ SOCKET_NOWAIT** flag. The exact behaviour of ReadMsg() depends on the **wxSOCKET_BLOCK** flag. For a detailed explanation, see SetFlags(). For thread safety, in case ReadMsg() and WriteMsg() are called in different threads, it is a good idea to call SetFlags(wxSOCKET_WAITALL|wx_SOCKET_BLOCK) before the first calls to ReadMsg() and WriteMsg() in different threads, as each of these functions will call SetFlags() which performs read/modify/write. By setting these flags before the multi-threading, it will ensure that they don't get reset by thread race conditions.

See also

Error(), LastError(), LastReadCount(), SetFlags(), WriteMsg()

void wxSocketBase::RestoreState ( )

Restore the previous state of the socket, as saved with SaveState().

Calls to SaveState() and RestoreState() can be nested.

See also

SaveState()

void wxSocketBase::SaveState ( )

Save the current state of the socket in a stack.

Socket state includes flags, as set with SetFlags(), event mask, as set with SetNotify() and Notify(), user data, as set with SetClientData(). Calls to SaveState and RestoreState can be nested.

See also

RestoreState()

void wxSocketBase::SetClientData (void * *data*)

Sets user-supplied client data for this socket.

All socket events will contain a pointer to this data, which can be retrieved with the wxSocketEvent::GetClientData() function.

void wxSocketBase::SetEventHandler (**wxEvtHandler** & *handler,* int *id* = -1)

Sets an event handler to be called when a socket event occurs.

The handler will be called for those events for which notification is enabled with SetNotify() and Notify().

Parameters

handler	Specifies the event handler you want to use.
id	The id of socket event.

See also

SetNotify(), Notify(), wxSocketEvent, wxEvtHandler

void wxSocketBase::SetFlags (wxSocketFlags *flags*)

Use SetFlags to customize IO operation for this socket.

The *flags* parameter may be a combination of flags ORed together. Notice that not all combinations of flags affecting the IO calls (Read() and Write()) make sense, e.g. **wxSOCKET_NOWAIT** can't be combined with **wxSOCKET_WAITALL** nor with **wxSOCKET_BLOCK**.

The following flags can be used:

- wxSOCKET_NONE: Default mode: the socket will read some data in the IO calls and will process events to avoid blocking UI while waiting for the data to become available.
- wxSOCKET_NOWAIT: Don't wait for the socket to become ready in IO calls, read as much data as is available – potentially 0 bytes – and return immediately.
- wxSOCKET_WAITALL: Don't return before the entire amount of data specified in IO calls is read or written unless an error occurs. If this flag is not specified, the IO calls return as soon as any amount of data, even less than the total number of bytes, is processed.
- wxSOCKET_BLOCK: Don't process the UI events while waiting for the socket to become ready. This means that UI will be unresponsive during socket IO.
- wxSOCKET_REUSEADDR: Allows the use of an in-use port (wxServerSocket only).
- wxSOCKET_BROADCAST: Switches the socket to broadcast mode.
- wxSOCKET_NOBIND: Stops the socket from being bound to a specific adapter (normally used in conjunction with **wxSOCKET_BROADCAST**).

For more information on socket events see wxSocketFlags .

virtual bool wxSocketBase::SetLocal (const wxIPV4address & *local*) [virtual]

Set the local address and port to use.

This function must always be called for the server sockets but may also be called for client sockets, if it is, **bind()** is called before **connect()**.

void wxSocketBase::SetNotify (wxSocketEventFlags *flags*)

Specifies which socket events are to be sent to the event handler.

The *flags* parameter may be combination of flags ORed together. The following flags can be used:

- wxSOCKET_INPUT_FLAG: to receive **wxSOCKET_INPUT**.
- wxSOCKET_OUTPUT_FLAG: to receive **wxSOCKET_OUTPUT**.
- wxSOCKET_CONNECTION_FLAG: to receive **wxSOCKET_CONNECTION**.
- wxSOCKET_LOST_FLAG: to receive **wxSOCKET_LOST**.

For example:

```
sock.SetNotify(wxSOCKET_INPUT_FLAG | wxSOCKET_LOST_FLAG);
sock.Notify(true);
```

In this example, the user will be notified about incoming socket data and whenever the connection is closed.

For more information on socket events see wxSocketEventFlags .

void wxSocketBase::SetTimeout (long *seconds*)

Set the default socket timeout in seconds.

This timeout applies to all IO calls, and also to the Wait() family of functions if you don't specify a wait interval. Initially, the default timeout is 10 minutes.

static void wxSocketBase::Shutdown ( ) [static]

Shut down the sockets.

This function undoes the call to Initialize() and must be called after every successful call to Initialize().

This function should only be called from the main thread, just as Initialize().

void wxSocketBase::ShutdownOutput ( )

Shuts down the writing end of the socket.

This function simply calls the standard shutdown() function on the underlying socket, indicating that nothing will be written to this socket any more.

wxSocketBase& wxSocketBase::Unread (const void * *buffer*, wxUint32 *nbytes*)

Put the specified data into the input queue.

The data in the buffer will be returned by the next call to Read().

This function is not affected by wxSocket flags.

If you use LastCount(), it will always return *nbytes*.

If you use Error(), it will always return false.

Parameters

buffer	Buffer to be unread.
nbytes	Number of bytes.

Returns

Returns a reference to the current object.

See also

Error(), LastCount(), LastError()

bool wxSocketBase::Wait (long *seconds* = -1, long *millisecond* = 0)

Wait for any socket event.

Possible socket events are:

- The socket becomes readable.
- The socket becomes writable.
- An ongoing connection request has completed (wxSocketClient only)
- An incoming connection request has arrived (wxSocketServer only)
- The connection has been closed.

Note that it is recommended to use the individual **WaitForXXX()** functions to wait for the required condition, instead of this one.

Parameters

seconds	Number of seconds to wait. If -1, it will wait for the default timeout, as set with SetTimeout().
millisecond	Number of milliseconds to wait.

Returns

true when any of the above conditions is satisfied or false if the timeout was reached.

See also

InterruptWait(), wxSocketServer::WaitForAccept(), WaitForLost(), WaitForRead(), WaitForWrite(), wxSocketClient::WaitOnConnect()

bool wxSocketBase::WaitForLost (long *seconds* = -1, long *millisecond* = 0)

Wait until the connection is lost.

This may happen if the peer gracefully closes the connection or if the connection breaks.

Parameters

seconds	Number of seconds to wait. If -1, it will wait for the default timeout, as set with SetTimeout().
millisecond	Number of milliseconds to wait.

Returns

Returns true if the connection was lost, false if the timeout was reached.

See also

InterruptWait(), Wait()

bool wxSocketBase::WaitForRead (long *seconds* = -1, long *millisecond* = 0)

Wait until the socket is readable.

This might mean that queued data is available for reading or, for streamed sockets, that the connection has been closed, so that a read operation will complete immediately without blocking (unless the **wxSOCKET_WAITALL** flag is set, in which case the operation might still block).

Notice that this function should not be called if there is already data available for reading on the socket.

Parameters

seconds	Number of seconds to wait. If -1, it will wait for the default timeout, as set with SetTimeout().
millisecond	Number of milliseconds to wait.

Returns

Returns true if the socket becomes readable, false on timeout.

See also

InterruptWait(), Wait()

bool wxSocketBase::WaitForWrite (long *seconds* = -1, long *millisecond* = 0)

Wait until the socket becomes writable.

This might mean that the socket is ready to send new data, or for streamed sockets, that the connection has been closed, so that a write operation is guaranteed to complete immediately (unless the **wxSOCKET_WAITALL** flag is set, in which case the operation might still block).

Notice that this function should not be called if the socket is already writable.

Parameters

seconds	Number of seconds to wait. If -1, it will wait for the default timeout, as set with SetTimeout().
millisecond	Number of milliseconds to wait.

Returns

Returns true if the socket becomes writable, false on timeout.

See also

InterruptWait(), Wait()

wxSocketBase& wxSocketBase::Write (const void * *buffer*, wxUint32 *nbytes*)

Write up to the given number of bytes to the socket.

Use LastWriteCount() to verify the number of bytes actually written.

Use Error() to determine if the operation succeeded.

Parameters

buffer	Buffer with the data to be sent.
nbytes	Number of bytes.

Returns

Returns a reference to the current object.

Remarks

The exact behaviour of Write() depends on the combination of flags being used. For a detailed explanation, see SetFlags().

See also

Error(), LastError(), LastWriteCount(), SetFlags()

wxSocketBase& wxSocketBase::WriteMsg (const void * *buffer*, wxUint32 *nbytes*)

Sends a buffer which can be read using ReadMsg().

WriteMsg() sends a short header before the data so that ReadMsg() knows how much data should be actually read.

This function always waits for the entire buffer to be sent, unless an error occurs.

Use LastWriteCount() to verify the number of bytes actually written.

Use Error() to determine if the operation succeeded.

Parameters

buffer	Buffer with the data to be sent.
nbytes	Number of bytes to send.

Returns

Returns a reference to the current object.

Remarks

WriteMsg() will behave as if the **wxSOCKET_WAITALL** flag was always set and it will always ignore the **wxSOCKET_NOWAIT** flag. The exact behaviour of WriteMsg() depends on the **wxSOCKET_BLOCK** flag. For a detailed explanation, see SetFlags(). For thread safety, in case ReadMsg() and WriteMsg() are called in different threads, it is a good idea to call

```
SetFlags(wxSOCKET_WAITALL|wx_SOCKET_BLOCK)
```

before the first calls to ReadMsg() and WriteMsg() in different threads, as each of these functions calls SetFlags() which performs read/modify/write. By setting these flags before the multi-threading, it will ensure that they don't get reset by thread race conditions.

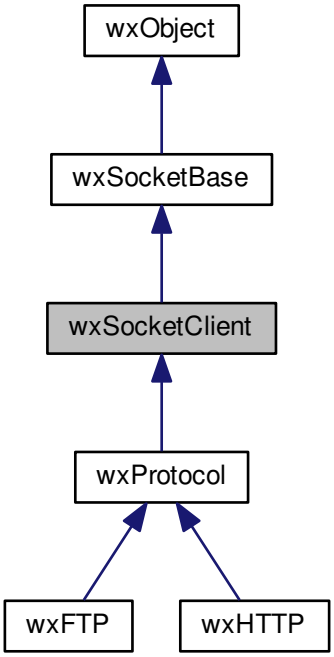
See also

Error(), LastError(), LastWriteCount(), SetFlags(), ReadMsg()

21.693 wxSocketClient Class Reference

```
#include <wx/socket.h>
```

Inheritance diagram for wxSocketClient:



21.693.1 Detailed Description

Todo describe me.

Library: wxNet

Category: Networking

Public Member Functions

- wxSocketClient (wxSocketFlags flags=wxSOCKET_NONE)

  Constructor.
- virtual ~wxSocketClient ()

  Destructor.
- virtual bool Connect (const wxSockAddress &address, bool wait=true)

  Connects to a server using the specified address.
- bool Connect (const wxSockAddress &address, const wxSockAddress &local, bool wait=true)

  Connects to a server using the specified address.
- bool WaitOnConnect (long seconds=-1, long milliseconds=0)

  Wait until a connection request completes, or until the specified timeout elapses.

Additional Inherited Members

21.693.2 Constructor & Destructor Documentation

wxSocketClient::wxSocketClient (wxSocketFlags *flags* = wxSOCKET_NONE)

Constructor.

Parameters

flags	Socket flags (See wxSocketBase::SetFlags())

virtual wxSocketClient::~wxSocketClient ( ) [virtual]

Destructor.

Please see wxSocketBase::Destroy().

21.693.3 Member Function Documentation

virtual bool wxSocketClient::Connect (const wxSockAddress & *address*, bool *wait* = true) [virtual]

Connects to a server using the specified address.

If *wait* is true, Connect() will wait until the connection completes.

Warning

This method will block the GUI.

If *wait* is false, Connect() will try to establish the connection and return immediately, without blocking the GUI. When used this way, even if Connect() returns false, the connection request can be completed later. To detect this, use WaitOnConnect(), or catch **wxSOCKET_CONNECTION** events (for successful establishment) and **wxSOCKET_LOST** events (for connection failure).

Parameters

address	Address of the server.
wait	If true, waits for the connection to complete.

Returns

true if the connection is established and no error occurs. If *wait* was true, and Connect() returns false, an error occurred and the connection failed. If *wait* was false, and Connect() returns false, you should still be prepared to handle the completion of this connection request, either with WaitOnConnect() or by watching wxSOCKET_CONNECTION and wxSOCKET_LOST events.

See also

WaitOnConnect(), wxSocketBase::SetNotify(), wxSocketBase::Notify()

Reimplemented in wxHTTP.

bool wxSocketClient::Connect (const wxSockAddress & *address*, const wxSockAddress & *local*, bool *wait* = true)

Connects to a server using the specified address.

If *wait* is true, Connect() will wait until the connection completes. **Warning:** This will block the GUI.

If *wait* is false, Connect() will try to establish the connection and return immediately, without blocking the GUI. When used this way, even if Connect() returns false, the connection request can be completed later. To detect this, use WaitOnConnect(), or catch **wxSOCKET_CONNECTION** events (for successful establishment) and **wxSOCKET_LOST** events (for connection failure).

Parameters

address	Address of the server.
local	Bind to the specified local address and port before connecting. The local address and port can also be set using SetLocal(), and then using the 2-parameter Connect() method.
wait	If true, waits for the connection to complete.

Returns

true if the connection is established and no error occurs. If *wait* was true, and Connect() returns false, an error occurred and the connection failed. If *wait* was false, and Connect() returns false, you should still be prepared to handle the completion of this connection request, either with WaitOnConnect() or by watching wxSOCKET_CONNECTION and wxSOCKET_LOST events.

See also

WaitOnConnect(), wxSocketBase::SetNotify(), wxSocketBase::Notify()

bool wxSocketClient::WaitOnConnect (long *seconds* = -1, long *milliseconds* = 0)

Wait until a connection request completes, or until the specified timeout elapses.

Use this function after issuing a call to Connect() with *wait* set to false.

Parameters

seconds	Number of seconds to wait. If -1, it will wait for the default timeout, as set with wxSocketBase::SetTimeout().
milliseconds	Number of milliseconds to wait.

Returns

WaitOnConnect() returns true if the connection request completes. This does not necessarily mean that the connection was successfully established; it might also happen that the connection was refused by the peer. Use wxSocketBase::IsConnected() to distinguish between these two situations.

If the timeout elapses, WaitOnConnect() returns false.

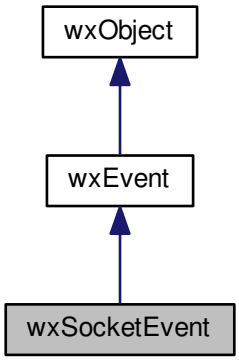
These semantics allow code like this:

```
// Issue the connection request
client->Connect(addr, false);

// Wait until the request completes or until we decide to give up
bool waitmore = true;
while ( !client->WaitOnConnect(seconds, millis) && waitmore )
{
    // possibly give some feedback to the user,
    // and update waitmore as needed.
}
bool success = client->IsConnected();
```

21.694 wxSocketEvent Class Reference

`#include <wx/socket.h>`

Inheritance diagram for wxSocketEvent:



21.694.1 Detailed Description

This event class contains information about socket events.

This kind of events are sent to the event handler specified with wxSocketBase::SetEventHandler.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSocketEvent& event)

Event macros:

- EVT_SOCKET(id, func): Process a socket event, supplying the member function.

Library: wxNet

Category: Networking

See also

wxSocketBase, wxSocketClient, wxSocketServer

Public Member Functions

- wxSocketEvent (int id=0)

  Constructor.
- void * GetClientData () const

  Gets the client data of the socket which generated this event, as set with wxSocketBase::SetClientData().
- wxSocketBase * GetSocket () const

  Returns the socket object to which this event refers to.
- wxSocketNotify GetSocketEvent () const

  Returns the socket event type.

Additional Inherited Members

21.694.2 Constructor & Destructor Documentation

wxSocketEvent::wxSocketEvent (int *id* = 0)

Constructor.

21.694.3 Member Function Documentation

void* wxSocketEvent::GetClientData ( ) const

Gets the client data of the socket which generated this event, as set with wxSocketBase::SetClientData().

wxSocketBase* wxSocketEvent::GetSocket ( ) const

Returns the socket object to which this event refers to.

This makes it possible to use the same event handler for different sockets.

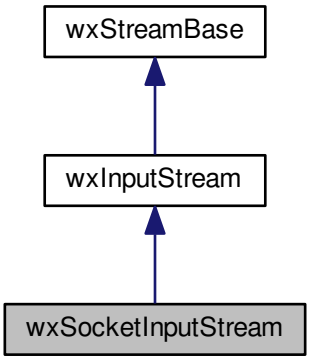
wxSocketNotify wxSocketEvent::GetSocketEvent ( ) const

Returns the socket event type.

21.695 wxSocketInputStream Class Reference

`#include <wx/sckstrm.h>`

Inheritance diagram for wxSocketInputStream:



21.695.1 Detailed Description

This class implements an input stream which reads data from a connected socket.

Note that this stream is purely sequential and it does not support seeking.

Library: wxNet

Category: Networking, Streams

See also

wxSocketBase

Public Member Functions

- wxSocketInputStream (wxSocketBase &s)

  Creates a new read-only socket stream using the specified initialized socket connection.

Additional Inherited Members

21.695.2 Constructor & Destructor Documentation

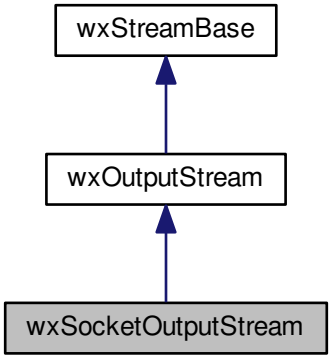
wxSocketInputStream::wxSocketInputStream (wxSocketBase & *s*)

Creates a new read-only socket stream using the specified initialized socket connection.

21.696 wxSocketOutputStream Class Reference

`#include <wx/sckstrm.h>`

Inheritance diagram for wxSocketOutputStream:



21.696.1 Detailed Description

This class implements an output stream which writes data from a connected socket.

Note that this stream is purely sequential and it does not support seeking.

Library: wxNet

Category: Networking, Streams

See also

wxSocketBase

Public Member Functions

- wxSocketOutputStream (wxSocketBase &s)

  Creates a new write-only socket stream using the specified initialized socket connection.

Additional Inherited Members

21.696.2 Constructor & Destructor Documentation

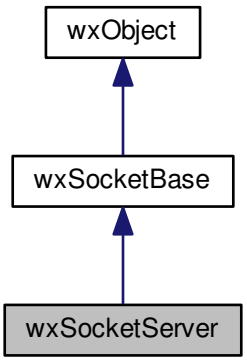
wxSocketOutputStream::wxSocketOutputStream (wxSocketBase & *s*)

Creates a new write-only socket stream using the specified initialized socket connection.

21.697 wxSocketServer Class Reference

```
#include <wx/socket.h>
```

Inheritance diagram for wxSocketServer:



21.697.1 Detailed Description

Todo describe me.

Library: wxNet

Category: Networking

Public Member Functions

- wxSocketServer (const wxSockAddress &address, wxSocketFlags flags=wxSOCKET_NONE)

  Constructs a new server and tries to bind to the specified address.
- virtual ~wxSocketServer ()

  Destructor (it doesn't close the accepted connections).
- wxSocketBase * Accept (bool wait=true)

  Accepts an incoming connection request, and creates a new wxSocketBase object which represents the server-side of the connection.
- bool AcceptWith (wxSocketBase &socket, bool wait=true)

  Accept an incoming connection using the specified socket object.
- bool WaitForAccept (long seconds=-1, long millisecond=0)

  Wait for an incoming connection.

Additional Inherited Members

21.697.2 Constructor & Destructor Documentation

wxSocketServer::wxSocketServer (const wxSockAddress & *address*, wxSocketFlags *flags* = wxSOCKET_NONE)

Constructs a new server and tries to bind to the specified *address*.

Before trying to accept new connections, remember to test whether it succeeded with wxSocketBase:IsOk().

Parameters

address	Specifies the local address for the server (e.g. port number).
flags	Socket flags (See wxSocketBase::SetFlags()).

virtual wxSocketServer::~wxSocketServer ( ) [virtual]

Destructor (it doesn't close the accepted connections).

21.697.3 Member Function Documentation

wxSocketBase* wxSocketServer::Accept (bool *wait* = true)

Accepts an incoming connection request, and creates a new wxSocketBase object which represents the server-side of the connection.

If *wait* is true and there are no pending connections to be accepted, it will wait for the next incoming connection to arrive.

Warning

This method will block the GUI.

If *wait* is false, it will try to accept a pending connection if there is one, but it will always return immediately without blocking the GUI. If you want to use Accept() in this way, you can either check for incoming connections with WaitForAccept() or catch **wxSOCKET_CONNECTION** events, then call Accept() once you know that there is an incoming connection waiting to be accepted.

Returns

Returns an opened socket connection, or NULL if an error occurred or if the wait parameter was false and there were no pending connections.

See also

WaitForAccept(), wxSocketBase::SetNotify(), wxSocketBase::Notify(), AcceptWith()

bool wxSocketServer::AcceptWith (wxSocketBase & *socket*, bool *wait* = `true`)

Accept an incoming connection using the specified socket object.

Parameters

socket	Socket to be initialized
wait	See Accept() for more info.

Returns

Returns true on success, or false if an error occurred or if the wait parameter was false and there were no pending connections.

See also

WaitForAccept(), wxSocketBase::SetNotify(), wxSocketBase::Notify(), Accept()

bool wxSocketServer::WaitForAccept (long *seconds* = `-1`, long *millisecond* = `0`)

Wait for an incoming connection.

Use it if you want to call Accept() or AcceptWith() with *wait* set to false, to detect when an incoming connection is waiting to be accepted.

Parameters

seconds	Number of seconds to wait. If -1, it will wait for the default timeout, as set with wxSocketBase::SetTimeout().
millisecond	Number of milliseconds to wait.

Returns

true if an incoming connection arrived, false if the timeout elapsed.

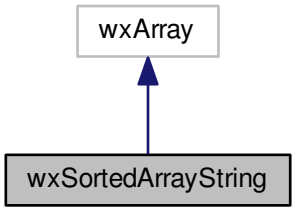
See also

Accept(), AcceptWith(), wxSocketBase::InterruptWait()

21.698 wxSortedArrayString Class Reference

`#include <wx/arrstr.h>`

Inheritance diagram for wxSortedArrayString:



21.698.1 Detailed Description

wxSortedArrayString is an efficient container for storing wxString objects which always keeps the string in alphabetical order.

wxSortedArrayString uses binary search in its wxSortedArrayString::Index() method (instead of linear search for wxArrayString::Index()) which makes it much more efficient if you add strings to the array rarely (because, of course, you have to pay for Index() efficiency by having Add() be slower) but search for them often. Several methods should not be used with sorted array (basically, all those which break the order of items) which is mentioned in their description.

Library: wxBase

Category: Containers

See also

wxArray, wxString, wxString Overview

Public Member Functions

- wxSortedArrayString ()

  Default constructor.
- wxSortedArrayString (CompareFunction compareFunction)

  Constructs a sorted array using the specified compareFunction for item comparison.
- wxSortedArrayString (const wxArrayString &array)

  Conversion constructor.
- size_t Add (const wxString &str, size_t copies=1)

  Appends the given number of copies of the new item str to the array and returns the index of the first new item in the array.
- int Index (const wxString &sz, bool bCase=true, bool bFromEnd=false) const

  Search the element in the array, starting from the beginning if bFromEnd is false or from end otherwise.
- void Insert (const wxString &str, size_t nIndex, size_t copies=1)

- void Sort (bool reverseOrder=false)
- void Sort (CompareFunction compareFunction)

21.698.2 Constructor & Destructor Documentation

wxSortedArrayString::wxSortedArrayString ( )

Default constructor.

The elements of the array are kept sorted in alphabetical order.

wxSortedArrayString::wxSortedArrayString (CompareFunction *compareFunction*)

Constructs a sorted array using the specified *compareFunction* for item comparison.

See also

wxStringSortAscending(), wxDictionaryStringSortAscending()

Since

3.1.0

wxSortedArrayString::wxSortedArrayString (const wxArrayString & *array*)

Conversion constructor.

Constructs a sorted array with the same contents as the (possibly unsorted) "array" argument.

21.698.3 Member Function Documentation

size_t wxSortedArrayString::Add (const wxString & *str*, size_t *copies* = `1`)

Appends the given number of *copies* of the new item *str* to the array and returns the index of the first new item in the array.

See also

Insert()

Warning

For sorted arrays, the index of the inserted item will not be, in general, equal to GetCount() - 1 because the item is inserted at the correct position to keep the array sorted and not appended.

int wxSortedArrayString::Index (const wxString & *sz*, bool *bCase* = `true`, bool *bFromEnd* = `false`) const

Search the element in the array, starting from the beginning if *bFromEnd* is false or from end otherwise.

If *bCase*, comparison is case sensitive (default), otherwise the case is ignored.

This function uses linear search for wxArrayString. Returns index of the first item matched or `wxNOT_FOUND` if there is no match.

This function uses binary search for wxSortedArrayString, but it ignores the *bCase* and *bFromEnd* parameters.

void wxSortedArrayString::Insert (const wxString & *str,* size_t *nIndex,* size_t *copies* = `1`)

Warning

This function should not be used with sorted arrays because it could break the order of items and, for example, subsequent calls to Index() would then not work!
In STL mode, Insert is private and simply invokes wxFAIL_MSG.

void wxSortedArrayString::Sort (bool *reverseOrder* = `false`)

Warning

This function should not be used with sorted array because it could break the order of items and, for example, subsequent calls to Index() would then not work! Also, sorting a wxSortedArrayString doesn't make sense because its elements are always already sorted.
In STL mode, Sort is private and simply invokes wxFAIL_MSG.

void wxSortedArrayString::Sort (CompareFunction *compareFunction*)

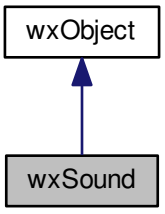
Warning

This function should not be used with sorted array because it could break the order of items and, for example, subsequent calls to Index() would then not work! Also, sorting a wxSortedArrayString doesn't make sense because its elements are always already sorted.
In STL mode, Sort is private and simply invokes wxFAIL_MSG.

21.699 wxSound Class Reference

`#include <wx/sound.h>`

Inheritance diagram for wxSound:



21.699.1 Detailed Description

This class represents a short sound (loaded from Windows WAV file), that can be stored in memory and played.

Currently this class is implemented on Windows and Unix (and uses either Open Sound System or Simple Direct↩ Media Layer).

Library: wxAdvanced

Category: Multimedia

- bool Play (unsigned flags=wxSOUND_ASYNC) const

  Plays the sound file.
- static bool Play (const wxString &filename, unsigned flags=wxSOUND_ASYNC)

  Plays the sound file.

Public Member Functions

- wxSound ()

  Default ctor.
- wxSound (const wxString &fileName, bool isResource=false)

  Constructs a wave object from a file or, under Windows, from a Windows resource.
- wxSound (size_t size, const void *data)

  Constructs a wave object from in-memory data.
- virtual ~wxSound ()

  Destroys the wxSound object.
- bool Create (const wxString &fileName, bool isResource=false)

  Constructs a wave object from a file or resource.
- bool Create (size_t size, const void *data)

  Constructs a wave object from in-memory data.
- bool IsOk () const

  Returns true if the object contains a successfully loaded file or resource, false otherwise.

Static Public Member Functions

- static bool IsPlaying ()

  Returns true if a sound is played at the moment.
- static void Stop ()

  If a sound is played, this function stops it.

Additional Inherited Members

21.699.2 Constructor & Destructor Documentation

wxSound::wxSound ( )

Default ctor.

wxSound::wxSound (const wxString & *fileName*, bool *isResource* = `false`)

Constructs a wave object from a file or, under Windows, from a Windows resource.

Call IsOk() to determine whether this succeeded.

Parameters

fileName	The filename or Windows resource.
isResource	true if fileName is a resource, false if it is a filename.

wxSound::wxSound (size_t *size*, const void * *data*)

Constructs a wave object from in-memory data.

Parameters

size	Size of the buffer pointer to by *data*.
data	The buffer containing the sound data in WAV format.

virtual wxSound::~wxSound ( ) [virtual]

Destroys the wxSound object.

21.699.3 Member Function Documentation

bool wxSound::Create (const wxString & *fileName*, bool *isResource* = false)

Constructs a wave object from a file or resource.

Parameters

fileName	The filename or Windows resource.
isResource	true if fileName is a resource, false if it is a filename.

Returns

true if the call was successful, false otherwise.

bool wxSound::Create (size_t *size*, const void * *data*)

Constructs a wave object from in-memory data.

Parameters

size	Size of the buffer pointer to by *data*.
data	The buffer containing the sound data in WAV format.

bool wxSound::IsOk ( ) const

Returns true if the object contains a successfully loaded file or resource, false otherwise.

static bool wxSound::IsPlaying ( ) [static]

Returns true if a sound is played at the moment.

This method is currently not available under Windows and may not be always implemented in Unix ports depending on the compilation options used (in this case it just always returns false).

Availability: only available for the wxGTK, wxOSX ports.

bool wxSound::Play (unsigned *flags* = wxSOUND_ASYNC) const

Plays the sound file.

If another sound is playing, it will be interrupted.

Returns true on success, false otherwise. Note that in general it is possible to delete the object which is being asynchronously played any time after calling this function and the sound would continue playing, however this currently doesn't work under Windows for sound objects loaded from memory data.

The possible values for *flags* are:

- wxSOUND_SYNC: `Play` will block and wait until the sound is replayed.
- wxSOUND_ASYNC: Sound is played asynchronously, `Play` returns immediately.
- wxSOUND_ASYNC|wxSOUND_LOOP: Sound is played asynchronously and loops until another sound is played, Stop() is called or the program terminates.

The static form is shorthand for this code:

```
wxSound(filename).Play(flags);
```

static bool wxSound::Play (const wxString & *filename*, unsigned *flags* = wxSOUND_ASYNC) [static]

Plays the sound file.

If another sound is playing, it will be interrupted.

Returns true on success, false otherwise. Note that in general it is possible to delete the object which is being asynchronously played any time after calling this function and the sound would continue playing, however this currently doesn't work under Windows for sound objects loaded from memory data.

The possible values for *flags* are:

- wxSOUND_SYNC: `Play` will block and wait until the sound is replayed.
- wxSOUND_ASYNC: Sound is played asynchronously, `Play` returns immediately.
- wxSOUND_ASYNC|wxSOUND_LOOP: Sound is played asynchronously and loops until another sound is played, Stop() is called or the program terminates.

The static form is shorthand for this code:

```
wxSound(filename).Play(flags);
```

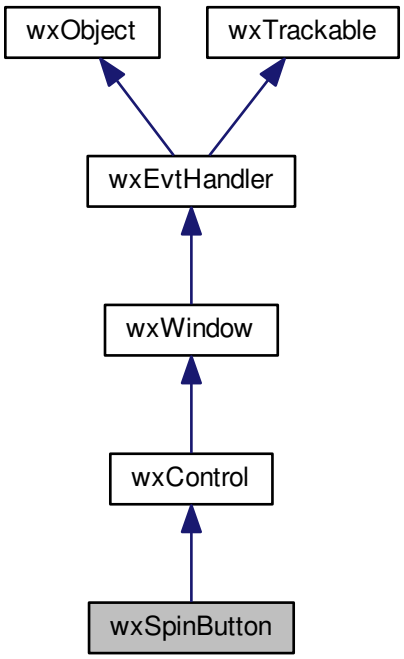
static void wxSound::Stop ( ) [static]

If a sound is played, this function stops it.

21.700 wxSpinButton Class Reference

`#include <wx/spinbutt.h>`

Inheritance diagram for wxSpinButton:



21.700.1 Detailed Description

A wxSpinButton has two small up and down (or left and right) arrow buttons.

It is often used next to a text control for increment and decrementing a value. Portable programs should try to use wxSpinCtrl instead as wxSpinButton is not implemented for all platforms but wxSpinCtrl is as it degenerates to a simple wxTextCtrl on such platforms.

Note

the range supported by this control (and wxSpinCtrl) depends on the platform but is at least `-0x8000` to `0x7fff`. Under GTK and Win32 with sufficiently new version of `comctrl32.dll` (at least 4.71 is required, 5.80 is recommended) the full 32 bit range is supported.

Styles

This class supports the following styles:

- wxSP_HORIZONTAL: Specifies a horizontal spin button (note that this style is not supported in wxGTK).
- wxSP_VERTICAL: Specifies a vertical spin button.
- wxSP_ARROW_KEYS: The user can use arrow keys to change the value.
- wxSP_WRAP: The value wraps at the minimum and maximum.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSpinEvent& event)

Event macros for events emitted by this class:

- EVT_SPIN(id, func): Generated whenever an arrow is pressed.
- EVT_SPIN_UP(id, func): Generated when left/up arrow is pressed.
- EVT_SPIN_DOWN(id, func): Generated when right/down arrow is pressed.

Note that if you handle both SPIN and UP or DOWN events, you will be notified about each of them twice: first the UP/DOWN event will be received and then, if it wasn't vetoed, the SPIN event will be sent.

Library: wxCore

Category: Controls

See also

wxSpinCtrl

Public Member Functions

- wxSpinButton ()

  Default constructor.
- wxSpinButton (wxWindow *parent, wxWindowID id=-1, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_VERTICAL, const wxString &name="spinButton")

  Constructor, creating and showing a spin button.
- virtual ~wxSpinButton ()

  Destructor, destroys the spin button control.
- bool Create (wxWindow *parent, wxWindowID id=-1, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_VERTICAL, const wxString &name="wxSpinButton")

  Scrollbar creation function called by the spin button constructor.
- virtual int GetMax () const

  Returns the maximum permissible value.
- virtual int GetMin () const

  Returns the minimum permissible value.
- virtual int GetValue () const

  Returns the current spin button value.
- virtual void SetRange (int min, int max)

  Sets the range of the spin button.
- virtual void SetValue (int value)

  Sets the value of the spin button.

Additional Inherited Members

21.700.2 Constructor & Destructor Documentation

wxSpinButton::wxSpinButton ( )

Default constructor.

wxSpinButton::wxSpinButton (wxWindow * *parent*, wxWindowID *id* = -1, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_VERTICAL, const wxString & *name* = "spinButton")

Constructor, creating and showing a spin button.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxSpinButton class description.
name	Window name.

See also

Create()

virtual wxSpinButton::~wxSpinButton ( ) [virtual]

Destructor, destroys the spin button control.

21.700.3 Member Function Documentation

bool wxSpinButton::Create (wxWindow * *parent*, wxWindowID *id* = -1, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_VERTICAL, const wxString & *name* = "wxSpinButton")

Scrollbar creation function called by the spin button constructor.

See wxSpinButton() for details.

virtual int wxSpinButton::GetMax ( ) const [virtual]

Returns the maximum permissible value.

See also

SetRange()

virtual int wxSpinButton::GetMin ( ) const [virtual]

Returns the minimum permissible value.

See also

SetRange()

virtual int wxSpinButton::GetValue ( ) const [virtual]

Returns the current spin button value.

See also

SetValue()

virtual void wxSpinButton::SetRange (int *min*, int *max*) [virtual]

Sets the range of the spin button.

Parameters

min	The minimum value for the spin button.
max	The maximum value for the spin button.

See also

GetMin(), GetMax()

virtual void wxSpinButton::SetValue (int *value*) [virtual]

Sets the value of the spin button.

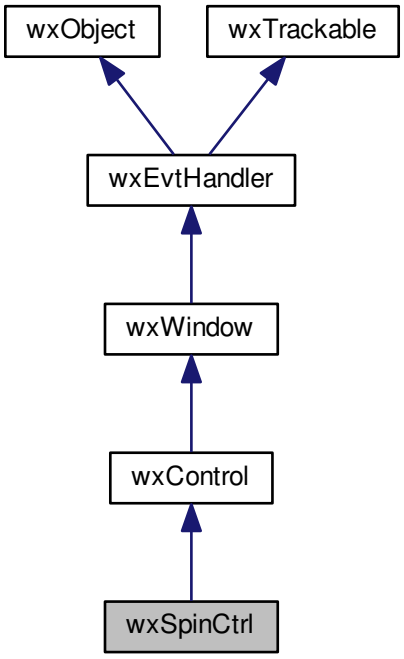
Parameters

value	The value for the spin button.

21.701 wxSpinCtrl Class Reference

`#include <wx/spinctrl.h>`

Inheritance diagram for wxSpinCtrl:



21.701.1 Detailed Description

wxSpinCtrl combines wxTextCtrl and wxSpinButton in one control.

Styles

This class supports the following styles:

- wxSP_ARROW_KEYS: The user can use arrow keys to change the value.
- wxSP_WRAP: The value wraps at the minimum and maximum.
- wxTE_PROCESS_ENTER: Indicates that the control should generate `wxEVT_TEXT_ENTER` events. Using this style will prevent the user from using the Enter key for dialog navigation (e.g. activating the default button in the dialog) under MSW.
- wxALIGN_LEFT: Same as wxTE_LEFT for wxTextCtrl: the text is left aligned.
- wxALIGN_CENTRE_HORIZONTAL: Same as wxTE_CENTRE for wxTextCtrl: the text is centered.
- wxALIGN_RIGHT: Same as wxTE_RIGHT for wxTextCtrl: the text is right aligned (this is the default).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSpinEvent& event)

Event macros for events emitted by this class:

- EVT_SPINCTRL(id, func): Process a wxEVT_SPINCTRL event, which is generated whenever the numeric value of the spin control is updated.

You may also use the wxSpinButton event macros, however the corresponding events will not be generated under all platforms. Finally, if the user modifies the text in the edit part of the spin control directly, the EVT_TEXT is generated, like for the wxTextCtrl. When the use enters text into the text area, the text is not validated until the control loses focus (e.g. by using the TAB key). The value is then adjusted to the range and a wxSpinEvent sent then if the value is different from the last value sent.

Library: wxCore

Category: Controls

See also

wxSpinButton, wxSpinCtrlDouble, wxControl

Public Member Functions

- wxSpinCtrl ()

  Default constructor.
- wxSpinCtrl (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_ARROW_KEYS, int min=0, int max=100, int initial=0, const wxString &name="wxSpinCtrl")

  Constructor, creating and showing a spin control.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_ARROW_KEYS, int min=0, int max=100, int initial=0, const wxString &name="wxSpinCtrl")

  Creation function called by the spin control constructor.
- int GetBase () const

  Returns the numerical base being currently used, 10 by default.

- int GetMax () const

  Gets maximal allowable value.
- int GetMin () const

  Gets minimal allowable value.
- int GetValue () const

  Gets the value of the spin control.
- bool SetBase (int base)

  Sets the base to use for the numbers in this control.
- void SetRange (int minVal, int maxVal)

  Sets range of allowable values.
- virtual void SetSelection (long from, long to)

  Select the text in the text part of the control between positions from (inclusive) and to (exclusive).
- virtual void SetValue (const wxString &text)

  Sets the value of the spin control.
- void SetValue (int value)

  Sets the value of the spin control.

Additional Inherited Members

21.701.2 Constructor & Destructor Documentation

wxSpinCtrl::wxSpinCtrl ( )

Default constructor.

wxSpinCtrl::wxSpinCtrl (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_ARROW_KEYS, int *min* = 0, int *max* = 100, int *initial* = 0, const wxString & *name* = "wxSpinCtrl")

Constructor, creating and showing a spin control.

If *value* is non-empty, it will be shown in the text entry part of the control and if it has numeric value, the initial numeric value of the control, as returned by GetValue() will also be determined by it instead of by *initial*. Hence, it only makes sense to specify *initial* if *value* is an empty string or is not convertible to a number, otherwise *initial* is simply ignored and the number specified by *value* is used.

Parameters

parent	Parent window. Must not be NULL.
value	Default value (as text).
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxSpinButton.
min	Minimal value.
max	Maximal value.
initial	Initial value.
name	Window name.

See also

Create()

21.701.3 Member Function Documentation

bool wxSpinCtrl::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_ARROW_KEYS, int *min* = 0, int *max* = 100, int *initial* = 0, const wxString & *name* = "wxSpinCtrl")

Creation function called by the spin control constructor.

See wxSpinCtrl() for details.

int wxSpinCtrl::GetBase ( ) const

Returns the numerical base being currently used, 10 by default.

See also

SetBase()

Since

2.9.5

int wxSpinCtrl::GetMax ( ) const

Gets maximal allowable value.

int wxSpinCtrl::GetMin ( ) const

Gets minimal allowable value.

int wxSpinCtrl::GetValue ( ) const

Gets the value of the spin control.

bool wxSpinCtrl::SetBase (int *base*)

Sets the base to use for the numbers in this control.

Currently the only supported values are 10 (which is the default) and 16.

Changing the base allows the user to enter the numbers in the specified base, e.g. with "0x" prefix for hexadecimal numbers, and also displays the numbers in the specified base when they are changed using the spin control arrows.

Parameters

base	Numeric base, currently only 10 and 16 are supported.

Returns

true if the base was successfully changed or false if it failed, usually meaning that either the base is not 10 or 16.

Since

2.9.5

void wxSpinCtrl::SetRange (int *minVal,* int *maxVal*)

Sets range of allowable values.

Notice that calling this method may change the value of the control if it's not inside the new valid range, e.g. it will become *minVal* if it is less than it now. However no `wxEVT_SPINCTRL` event is generated, even if it the value does change.

virtual void wxSpinCtrl::SetSelection (long *from,* long *to*) `[virtual]`

Select the text in the text part of the control between positions *from* (inclusive) and *to* (exclusive).

This is similar to wxTextCtrl::SetSelection().

Note

this is currently only implemented for Windows and generic versions of the control.

virtual void wxSpinCtrl::SetValue (const wxString & *text*) `[virtual]`

Sets the value of the spin control.

It is recommended to use the overload taking an integer value instead.

Notice that, unlike wxTextCtrl::SetValue(), but like most of the other setter methods in wxWidgets, calling this method does not generate any events as events are only generated for the user actions.

void wxSpinCtrl::SetValue (int *value*)

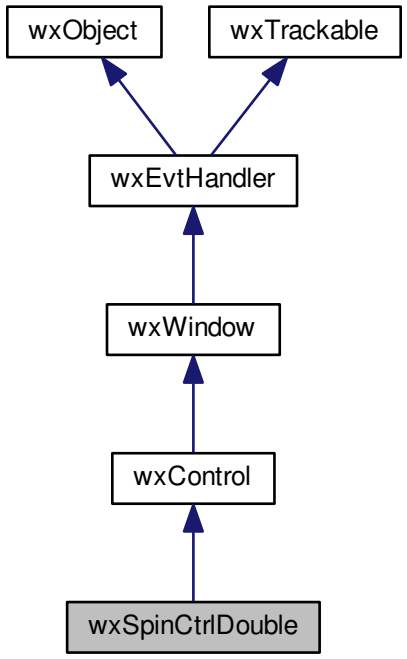
Sets the value of the spin control.

Calling this method doesn't generate any `wxEVT_SPINCTRL` events.

21.702 wxSpinCtrlDouble Class Reference

```
#include <wx/spinctrl.h>
```

Inheritance diagram for wxSpinCtrlDouble:



21.702.1 Detailed Description

wxSpinCtrlDouble combines wxTextCtrl and wxSpinButton in one control and displays a real number.

(wxSpinCtrl displays an integer.)

Styles

This class supports the following styles:

- wxSP_ARROW_KEYS: The user can use arrow keys to change the value.
- wxSP_WRAP: The value wraps at the minimum and maximum.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSpinDoubleEvent& event)

Event macros for events emitted by this class:

- EVT_SPINCTRLDOUBLE(id, func): Generated whenever the numeric value of the spin control is changed, that is, when the up/down spin button is clicked, when ENTER is pressed, or the control loses focus and the new value is different from the last. See wxSpinDoubleEvent.

Library: wxCore

Category: Controls

See also

wxSpinButton, wxSpinCtrl, wxControl

Public Member Functions

- wxSpinCtrlDouble ()

  Default constructor.
- wxSpinCtrlDouble (wxWindow *parent, wxWindowID id=-1, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_ARROW_KEYS, double min=0, double max=100, double initial=0, double inc=1, const wxString &name=wxT("wxSpinCtrlDouble"))

  Constructor, creating and showing a spin control.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_ARROW_KEYS, double min=0, double max=100, double initial=0, double inc=1, const wxString &name="wxSpinCtrlDouble")

  Creation function called by the spin control constructor.
- unsigned int GetDigits () const

  Gets the number of digits in the display.
- double GetIncrement () const

  Gets the increment value.
- double GetMax () const

  Gets maximal allowable value.
- double GetMin () const

  Gets minimal allowable value.
- double GetValue () const

  Gets the value of the spin control.
- void SetDigits (unsigned int digits)

  Sets the number of digits in the display.
- void SetIncrement (double inc)

  Sets the increment value.
- void SetRange (double minVal, double maxVal)

  Sets range of allowable values.
- virtual void SetValue (const wxString &text)

  Sets the value of the spin control.
- void SetValue (double value)

  Sets the value of the spin control.

Additional Inherited Members

21.702.2 Constructor & Destructor Documentation

wxSpinCtrlDouble::wxSpinCtrlDouble ( )

Default constructor.

wxSpinCtrlDouble::wxSpinCtrlDouble (wxWindow * *parent*, wxWindowID *id* = -1, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_ARROW_KEYS, double *min* = 0, double *max* = 100, double *initial* = 0, double *inc* = 1, const wxString & *name* = wxT("wxSpinCtrlDouble"))

Constructor, creating and showing a spin control.

Parameters

parent	Parent window. Must not be NULL.
value	Default value (as text).
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxSpinButton.
min	Minimal value.
max	Maximal value.
initial	Initial value.
inc	Increment value.
name	Window name.

See also

Create()

21.702.3 Member Function Documentation

bool wxSpinCtrlDouble::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_ARROW_KEYS, double *min* = 0, double *max* = 100, double *initial* = 0, double *inc* = 1, const wxString & *name* = "wxSpinCtrlDouble")

Creation function called by the spin control constructor.

See wxSpinCtrlDouble() for details.

unsigned int wxSpinCtrlDouble::GetDigits ( ) const

Gets the number of digits in the display.

double wxSpinCtrlDouble::GetIncrement ( ) const

Gets the increment value.

double wxSpinCtrlDouble::GetMax ( ) const

Gets maximal allowable value.

double wxSpinCtrlDouble::GetMin ( ) const

Gets minimal allowable value.

double wxSpinCtrlDouble::GetValue ( ) const

Gets the value of the spin control.

void wxSpinCtrlDouble::SetDigits (unsigned int *digits*)

Sets the number of digits in the display.

void wxSpinCtrlDouble::SetIncrement (double *inc*)

Sets the increment value.

Note

You may also need to increase the number of visible digits using SetDigits

void wxSpinCtrlDouble::SetRange (double *minVal,* double *maxVal*)

Sets range of allowable values.

virtual void wxSpinCtrlDouble::SetValue (const wxString & *text*) `[virtual]`

Sets the value of the spin control.

It is recommended to use the overload taking a double value instead.

Notice that, unlike wxTextCtrl::SetValue(), but like most of the other setter methods in wxWidgets, calling this method does not generate any events as events are only generated for the user actions.

void wxSpinCtrlDouble::SetValue (double *value*)

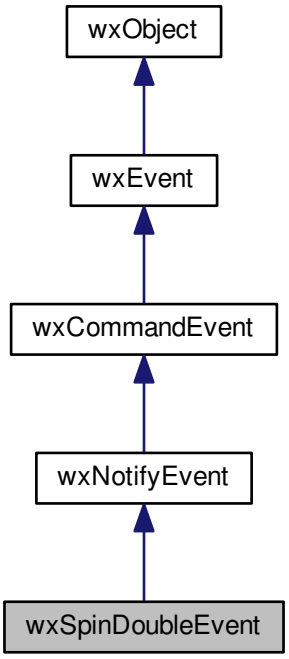
Sets the value of the spin control.

Calling this method doesn't generate any `wxEVT_SPINCTRLDOUBLE` events.

21.703 wxSpinDoubleEvent Class Reference

`#include <wx/spinctrl.h>`

Inheritance diagram for wxSpinDoubleEvent:



21.703.1 Detailed Description

This event class is used for the events generated by wxSpinCtrlDouble.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSpinDoubleEvent& event)

Event macros:

- EVT_SPINCTRLDOUBLE(id, func): Generated whenever the numeric value of the spin control is changed, that is, when the up/down spin button is clicked, when ENTER is pressed, or the control loses focus and the new value is different from the last. See wxSpinDoubleEvent.

Library: wxCore

Category: Events

See also

wxSpinCtrlDouble

Public Member Functions

- wxSpinDoubleEvent (wxEventType commandType=wxEVT_NULL, int winid=0, double value=0)

  The constructor.

- wxSpinDoubleEvent (const wxSpinDoubleEvent &event)

  The copy constructor.

- double GetValue () const

  Returns the value associated with this spin control event.

- void SetValue (double value)

  Set the value associated with the event.

Additional Inherited Members

21.703.2 Constructor & Destructor Documentation

wxSpinDoubleEvent::wxSpinDoubleEvent (wxEventType *commandType* = wxEVT_NULL, int *winid* = 0, double *value* = 0)

The constructor.

Not normally used by the user code.

wxSpinDoubleEvent::wxSpinDoubleEvent (const wxSpinDoubleEvent & *event*)

The copy constructor.

21.703.3 Member Function Documentation

double wxSpinDoubleEvent::GetValue ( ) const

Returns the value associated with this spin control event.

void wxSpinDoubleEvent::SetValue (double *value*)

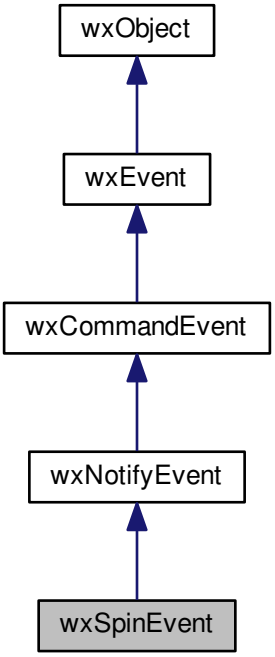
Set the value associated with the event.

(Not normally used by user code.)

21.704 wxSpinEvent Class Reference

`#include <wx/spinbutt.h>`

Inheritance diagram for wxSpinEvent:



21.704.1 Detailed Description

This event class is used for the events generated by wxSpinButton and wxSpinCtrl.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSpinEvent& event)

Event macros:

- EVT_SPIN(id, func): Generated whenever an arrow is pressed.
- EVT_SPIN_UP(id, func): Generated when left/up arrow is pressed.
- EVT_SPIN_DOWN(id, func): Generated when right/down arrow is pressed.

Note that if you handle both SPIN and UP or DOWN events, you will be notified about each of them twice: first the UP/DOWN event will be received and then, if it wasn't vetoed, the SPIN event will be sent.

Library: wxCore

Category: Events

See also

wxSpinButton and wxSpinCtrl

Public Member Functions

- wxSpinEvent (wxEventType commandType=wxEVT_NULL, int id=0)

  The constructor is not normally used by the user code.

- int GetPosition () const

  Retrieve the current spin button or control value.

- void SetPosition (int pos)

  Set the value associated with the event.

Additional Inherited Members

21.704.2 Constructor & Destructor Documentation

wxSpinEvent::wxSpinEvent (wxEventType *commandType* = wxEVT_NULL, int *id* = 0)

The constructor is not normally used by the user code.

21.704.3 Member Function Documentation

int wxSpinEvent::GetPosition ( ) const

Retrieve the current spin button or control value.

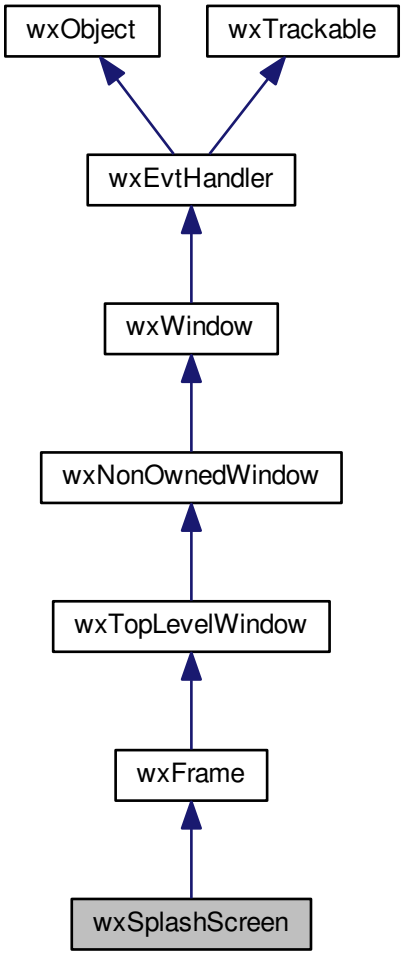
void wxSpinEvent::SetPosition (int *pos*)

Set the value associated with the event.

21.705 wxSplashScreen Class Reference

`#include <wx/splash.h>`

Inheritance diagram for wxSplashScreen:



21.705.1 Detailed Description

wxSplashScreen shows a window with a thin border, displaying a bitmap describing your application.

Show it in application initialisation, and then either explicitly destroy it or let it time-out.

Example usage:

```
wxBitmap bitmap;
if (bitmap.LoadFile("splash16.png", wxBITMAP_TYPE_PNG))
{
    wxSplashScreen* splash = new wxSplashScreen(bitmap,
        wxSPLASH_CENTRE_ON_SCREEN|wxSPLASH_TIMEOUT,
        6000, NULL, -1, wxDefaultPosition, wxDefaultSize,
        wxBORDER_SIMPLE|wxSTAY_ON_TOP);
}
wxYield();
```

Library: wxAdvanced

Category: Managed Windows

Public Member Functions

- wxSplashScreen (const wxBitmap &bitmap, long splashStyle, int milliseconds, wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxBORDER_SIMPLE|wxFRAME_NO_TASKBAR|wxSTAY_ON_TOP)

  Construct the splash screen passing a bitmap, a style, a timeout, a window id, optional position and size, and a window style.
- virtual ~wxSplashScreen ()

  Destroys the splash screen.
- long GetSplashStyle () const

  Returns the splash style (see wxSplashScreen() for details).
- wxSplashScreenWindow * GetSplashWindow () const

  Returns the window used to display the bitmap.
- int GetTimeout () const

  Returns the timeout in milliseconds.
- void OnCloseWindow (wxCloseEvent &event)

  Reimplement this event handler if you want to set an application variable on window destruction, for example.

Additional Inherited Members

21.705.2 Constructor & Destructor Documentation

wxSplashScreen::wxSplashScreen (const wxBitmap & *bitmap,* long *splashStyle,* int *milliseconds,* wxWindow * *parent,* wxWindowID *id,* const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxBORDER_SIMPLE|wxFRAME_NO_TASKBAR|wxSTAY_ON_TOP)

Construct the splash screen passing a bitmap, a style, a timeout, a window id, optional position and size, and a window style.

splashStyle is a bitlist of some of the following:

- wxSPLASH_CENTRE_ON_PARENT
- wxSPLASH_CENTRE_ON_SCREEN
- wxSPLASH_NO_CENTRE
- wxSPLASH_TIMEOUT
- wxSPLASH_NO_TIMEOUT

milliseconds is the timeout in milliseconds.

virtual wxSplashScreen::~wxSplashScreen ( ) [virtual]

Destroys the splash screen.

21.705.3 Member Function Documentation

long wxSplashScreen::GetSplashStyle ( ) const

Returns the splash style (see wxSplashScreen() for details).

wxSplashScreenWindow* wxSplashScreen::GetSplashWindow ( ) const

Returns the window used to display the bitmap.

int wxSplashScreen::GetTimeout ( ) const

Returns the timeout in milliseconds.

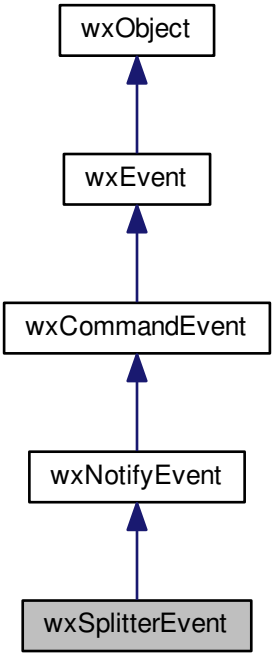
void wxSplashScreen::OnCloseWindow (wxCloseEvent & *event*)

Reimplement this event handler if you want to set an application variable on window destruction, for example.

21.706 wxSplitterEvent Class Reference

`#include <wx/splitter.h>`

Inheritance diagram for wxSplitterEvent:



21.706.1 Detailed Description

This class represents the events generated by a splitter control.

Also there is only one event class, the data associated to the different events is not the same and so not all accessor functions may be called for each event. The documentation mentions the kind of event(s) for which the given accessor function makes sense: calling it for other types of events will result in assert failure (in debug mode) and will return meaningless results.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSplitterEvent& event)

Event macros:

- EVT_SPLITTER_SASH_POS_CHANGING(id, func): The sash position is in the process of being changed. May be used to modify the position of the tracking bar to properly reflect the position that would be set if the drag were to be completed at this point. Processes a `wxEVT_SPLITTER_SASH_POS_CHANGING` event.
- EVT_SPLITTER_SASH_POS_CHANGED(id, func): The sash position was changed. May be used to modify the sash position before it is set, or to prevent the change from taking place. Processes a `wxEVT_SPLITTER_SASH_POS_CHANGED` event.
- EVT_SPLITTER_UNSPLIT(id, func): The splitter has been just unsplit. Processes a `wxEVT_SPLITTER_UNSPLIT` event.
- EVT_SPLITTER_DCLICK(id, func): The sash was double clicked. The default behaviour is to unsplit the window when this happens (unless the minimum pane size has been set to a value greater than zero). Processes a `wxEVT_SPLITTER_DOUBLECLICKED` event.

Library: wxCore

Category: Events

See also

wxSplitterWindow, Events and Event Handling

Public Member Functions

- wxSplitterEvent (wxEventType eventType=wxEVT_NULL, wxSplitterWindow *splitter=NULL)

  Constructor.
- int GetSashPosition () const

  Returns the new sash position.
- wxWindow * GetWindowBeingRemoved () const

  Returns a pointer to the window being removed when a splitter window is unsplit.
- int GetX () const

  Returns the x coordinate of the double-click point.
- int GetY () const

  Returns the y coordinate of the double-click point.
- void SetSashPosition (int pos)

  In the case of `wxEVT_SPLITTER_SASH_POS_CHANGED` events, sets the new sash position.

Additional Inherited Members

21.706.2 Constructor & Destructor Documentation

wxSplitterEvent::wxSplitterEvent (wxEventType *eventType* = wxEVT_NULL, wxSplitterWindow * *splitter* = `NULL`)

Constructor.

Used internally by wxWidgets only.

21.706.3 Member Function Documentation

int wxSplitterEvent::GetSashPosition ( ) const

Returns the new sash position.

May only be called while processing `wxEVT_SPLITTER_SASH_POS_CHANGING` and `wxEVT_SPLITTER_SASH_POS_CHANGED` events.

wxWindow* wxSplitterEvent::GetWindowBeingRemoved ( ) const

Returns a pointer to the window being removed when a splitter window is unsplit.

May only be called while processing `wxEVT_SPLITTER_UNSPLIT` events.

int wxSplitterEvent::GetX ( ) const

Returns the x coordinate of the double-click point.

May only be called while processing `wxEVT_SPLITTER_DOUBLECLICKED` events.

int wxSplitterEvent::GetY ( ) const

Returns the y coordinate of the double-click point.

May only be called while processing `wxEVT_SPLITTER_DOUBLECLICKED` events.

void wxSplitterEvent::SetSashPosition (int *pos*)

In the case of `wxEVT_SPLITTER_SASH_POS_CHANGED` events, sets the new sash position.

In the case of `wxEVT_SPLITTER_SASH_POS_CHANGING` events, sets the new tracking bar position so visual feedback during dragging will represent that change that will actually take place. Set to -1 from the event handler code to prevent repositioning.

May only be called while processing `wxEVT_SPLITTER_SASH_POS_CHANGING` and `wxEVT_SPLITTER_SASH_POS_CHANGED` events.

Parameters

pos	New sash position.

21.707 wxSplitterRenderParams Struct Reference

`#include <wx/renderer.h>`

21.707.1 Detailed Description

This is just a simple `struct` used as a return value of wxRendererNative::GetSplitterParams().

It doesn't have any methods and all of its fields are constant, so they can only be examined but not modified.

Library: wxCore

Category: Graphics Device Interface (GDI)

Public Member Functions

- wxSplitterRenderParams (wxCoord widthSash_, wxCoord border_, bool isSens_)

  The only way to initialize this struct is by using this ctor.

Public Attributes

- const wxCoord border

  The width of the border drawn by the splitter inside it, may be 0.
- const bool isHotSensitive

  true if the sash changes appearance when the mouse passes over it, false otherwise.
- const wxCoord widthSash

  The width of the splitter sash.

21.707.2 Constructor & Destructor Documentation

wxSplitterRenderParams::wxSplitterRenderParams (wxCoord *widthSash_*, wxCoord *border_*, bool *isSens_*)

The only way to initialize this struct is by using this ctor.

21.707.3 Member Data Documentation

const wxCoord wxSplitterRenderParams::border

The width of the border drawn by the splitter inside it, may be 0.

const bool wxSplitterRenderParams::isHotSensitive

true if the sash changes appearance when the mouse passes over it, false otherwise.

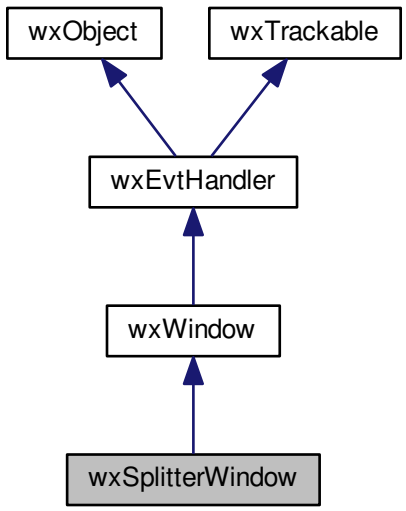
const wxCoord wxSplitterRenderParams::widthSash

The width of the splitter sash.

21.708 wxSplitterWindow Class Reference

`#include <wx/splitter.h>`

Inheritance diagram for wxSplitterWindow:



21.708.1 Detailed Description

This class manages up to two subwindows.

The current view can be split into two programmatically (perhaps from a menu command), and unsplit either programmatically or via the wxSplitterWindow user interface.

Styles

This class supports the following styles:

- wxSP_3D: Draws a 3D effect border and sash.
- wxSP_THIN_SASH: Draws a thin sash.
- wxSP_3DSASH: Draws a 3D effect sash (part of default style).
- wxSP_3DBORDER: Synonym for wxSP_BORDER.
- wxSP_BORDER: Draws a standard border.
- wxSP_NOBORDER: No border (default).
- wxSP_NO_XP_THEME: Under Windows XP, switches off the attempt to draw the splitter using Windows XP theming, so the borders and sash will take on the pre-XP look.
- wxSP_PERMIT_UNSPLIT: Always allow to unsplit, even with the minimum pane size other than zero.
- wxSP_LIVE_UPDATE: Don't draw XOR line but resize the child windows immediately.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSplitterEvent& event)

Event macros for events emitted by this class:

- EVT_SPLITTER_SASH_POS_CHANGING(id, func): The sash position is in the process of being changed. May be used to modify the position of the tracking bar to properly reflect the position that would be set if the drag were to be completed at this point. Processes a `wxEVT_SPLITTER_SASH_POS_CHANGING` event.
- EVT_SPLITTER_SASH_POS_CHANGED(id, func): The sash position was changed. May be used to modify the sash position before it is set, or to prevent the change from taking place. Processes a `wxEVT_SPLITTER_SASH_POS_CHANGED` event.
- EVT_SPLITTER_UNSPLIT(id, func): The splitter has been just unsplit. Processes a `wxEVT_SPLITTER_UNSPLIT` event.
- EVT_SPLITTER_DCLICK(id, func): The sash was double clicked. The default behaviour is to unsplit the window when this happens (unless the minimum pane size has been set to a value greater than zero). Processes a `wxEVT_SPLITTER_DOUBLECLICKED` event.

Library: wxCore

Category: Miscellaneous Windows

See also

wxSplitterEvent, wxSplitterWindow Overview

Public Member Functions

- wxSplitterWindow ()

  Default constructor.
- wxSplitterWindow (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_3D, const wxString &name="splitterWindow")

  Constructor for creating the window.
- virtual ~wxSplitterWindow ()

  Destroys the wxSplitterWindow and its children.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &point=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxSP_3D, const wxString &name="splitter")

  Creation function, for two-step construction.
- int GetMinimumPaneSize () const

  Returns the current minimum pane size (defaults to zero).
- double GetSashGravity () const

  Returns the current sash gravity.
- int GetSashPosition () const

  Returns the current sash position.
- int GetSashSize () const

  Returns the default sash size in pixels or 0 if it is invisible.
- int GetDefaultSashSize () const

  Returns the default sash size in pixels.
- wxSplitMode GetSplitMode () const

  Gets the split mode.

- wxWindow * GetWindow1 () const

  Returns the left/top or only pane.
- wxWindow * GetWindow2 () const

  Returns the right/bottom pane.
- void Initialize (wxWindow *window)

  Initializes the splitter window to have one pane.
- bool IsSashInvisible () const

  Returns true if the sash is invisible even when the window is split, false otherwise.
- bool IsSplit () const

  Returns true if the window is split, false otherwise.
- virtual void OnDoubleClickSash (int x, int y)

  Application-overridable function called when the sash is double-clicked with the left mouse button.
- virtual bool OnSashPositionChange (int newSashPosition)

  Application-overridable function called when the sash position is changed by user.
- virtual void OnUnsplit (wxWindow *removed)

  Application-overridable function called when the window is unsplit, either programmatically or using the wxSplitterWindow user interface.
- bool ReplaceWindow (wxWindow *winOld, wxWindow *winNew)

  This function replaces one of the windows managed by the wxSplitterWindow with another one.
- void SetMinimumPaneSize (int paneSize)

  Sets the minimum pane size.
- void SetSashGravity (double gravity)

  Sets the sash gravity.
- void SetSashPosition (int position, bool redraw=true)

  Sets the sash position.
- void SetSplitMode (int mode)

  Sets the split mode.
- void SetSashInvisible (bool invisible=true)

  Sets whether the sash should be invisible, even when the window is split.
- virtual bool SplitHorizontally (wxWindow *window1, wxWindow *window2, int sashPosition=0)

  Initializes the top and bottom panes of the splitter window.
- virtual bool SplitVertically (wxWindow *window1, wxWindow *window2, int sashPosition=0)

  Initializes the left and right panes of the splitter window.
- bool Unsplit (wxWindow *toRemove=NULL)

  Unsplits the window.
- void UpdateSize ()

  Causes any pending sizing of the sash and child panes to take place immediately.

Additional Inherited Members

21.708.2 Constructor & Destructor Documentation

wxSplitterWindow::wxSplitterWindow ( )

Default constructor.

wxSplitterWindow::wxSplitterWindow (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_3D, const wxString & *name* = "splitterWindow")

Constructor for creating the window.

Parameters

parent	The parent of the splitter window.
id	The window identifier.
pos	The window position.
size	The window size.
style	The window style. See wxSplitterWindow.
name	The window name.

Remarks

After using this constructor, you must create either one or two subwindows with the splitter window as parent, and then call one of Initialize(), SplitVertically() and SplitHorizontally() in order to set the pane(s). You can create two windows, with one hidden when not being shown; or you can create and delete the second pane on demand.

See also

Initialize(), SplitVertically(), SplitHorizontally(), Create()

virtual wxSplitterWindow::~wxSplitterWindow ( ) [virtual]

Destroys the wxSplitterWindow and its children.

21.708.3 Member Function Documentation

bool wxSplitterWindow::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *point* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxSP_3D, const wxString & *name* = "splitter")

Creation function, for two-step construction.

See wxSplitterWindow() for details.

int wxSplitterWindow::GetDefaultSashSize ( ) const

Returns the default sash size in pixels.

The size of the sash is its width for a vertically split window and its height for a horizontally split one. Its other direction is the same as the client size of the window in the corresponding direction.

The default sash size is platform-dependent because it conforms to the current platform look-and-feel and cannot be changed.

Since

2.9.4

int wxSplitterWindow::GetMinimumPaneSize ( ) const

Returns the current minimum pane size (defaults to zero).

See also

SetMinimumPaneSize()

double wxSplitterWindow::GetSashGravity ( ) const

Returns the current sash gravity.

See also

SetSashGravity()

int wxSplitterWindow::GetSashPosition ( ) const

Returns the current sash position.

See also

SetSashPosition()

int wxSplitterWindow::GetSashSize ( ) const

Returns the default sash size in pixels or 0 if it is invisible.

See also

GetDefaultSashSize(), IsSashInvisible()

wxSplitMode wxSplitterWindow::GetSplitMode ( ) const

Gets the split mode.

See also

SetSplitMode(), SplitVertically(), SplitHorizontally().

wxWindow* wxSplitterWindow::GetWindow1 ( ) const

Returns the left/top or only pane.

wxWindow* wxSplitterWindow::GetWindow2 ( ) const

Returns the right/bottom pane.

void wxSplitterWindow::Initialize (wxWindow * *window*)

Initializes the splitter window to have one pane.

The child window is shown if it is currently hidden.

Parameters

window	The pane for the unsplit window.

Remarks

This should be called if you wish to initially view only a single pane in the splitter window.

See also

SplitVertically(), SplitHorizontally()

bool wxSplitterWindow::IsSashInvisible ( ) const

Returns true if the sash is invisible even when the window is split, false otherwise.

Remarks

This is a shortcut for HasFlag(wxSP_NOSASH)

See also

SetSashInvisible()

Since

2.9.4

bool wxSplitterWindow::IsSplit ( ) const

Returns true if the window is split, false otherwise.

virtual void wxSplitterWindow::OnDoubleClickSash (int *x*, int *y*) [virtual]

Application-overridable function called when the sash is double-clicked with the left mouse button.

Parameters

x	The x position of the mouse cursor.
y	The y position of the mouse cursor.

Remarks

The default implementation of this function calls Unsplit if the minimum pane size is zero.

See also

Unsplit()

virtual bool wxSplitterWindow::OnSashPositionChange (int *newSashPosition*) [virtual]

Application-overridable function called when the sash position is changed by user.

It may return false to prevent the change or true to allow it.

Parameters

newSash↩ Position	The new sash position (always positive or zero)

Remarks

The default implementation of this function verifies that the sizes of both panes of the splitter are greater than minimum pane size.

virtual void wxSplitterWindow::OnUnsplit (wxWindow * *removed*) [virtual]

Application-overridable function called when the window is unsplit, either programmatically or using the wxSplitter↩ Window user interface.

Parameters

removed	The window being removed.

Remarks

The default implementation of this function simply hides removed. You may wish to delete the window.

bool wxSplitterWindow::ReplaceWindow (wxWindow * *winOld,* wxWindow * *winNew*)

This function replaces one of the windows managed by the wxSplitterWindow with another one.

It is in general better to use it instead of calling Unsplit() and then resplitting the window back because it will provoke much less flicker (if any). It is valid to call this function whether the splitter has two windows or only one.

Both parameters should be non-NULL and *winOld* must specify one of the windows managed by the splitter. If the parameters are incorrect or the window couldn't be replaced, false is returned. Otherwise the function will return true, but please notice that it will not delete the replaced window and you may wish to do it yourself.

See also

GetMinimumPaneSize()

void wxSplitterWindow::SetMinimumPaneSize (int *paneSize*)

Sets the minimum pane size.

Parameters

paneSize	Minimum pane size in pixels.

Remarks

The default minimum pane size is zero, which means that either pane can be reduced to zero by dragging the sash, thus removing one of the panes. To prevent this behaviour (and veto out-of-range sash dragging), set a minimum size, for example 20 pixels. If the wxSP_PERMIT_UNSPLIT style is used when a splitter window is created, the window may be unsplit even if minimum size is non-zero.

See also

GetMinimumPaneSize()

void wxSplitterWindow::SetSashGravity (double *gravity*)

Sets the sash gravity.

Parameters

gravity	The sash gravity. Value between 0.0 and 1.0.

Remarks

Gravity is real factor which controls position of sash while resizing wxSplitterWindow. Gravity tells wxSplitter↩ Window how much will left/top window grow while resizing. Example values:

- 0.0: only the bottom/right window is automatically resized
- 0.5: both windows grow by equal size

- 1.0: only left/top window grows Gravity should be a real value between 0.0 and 1.0. Default value of sash gravity is 0.0. That value is compatible with previous (before gravity was introduced) behaviour of wxSplitterWindow.

Notice that when sash gravity for a newly created splitter window, it is often necessary to explicitly set the splitter size using SetSize() to ensure that is big enough for its initial sash position. Otherwise, i.e. if the window is created with the default tiny size and only resized to its correct size later, the initial sash position will be affected by the gravity and typically result in sash being at the rightmost position for the gravity of 1. See the example code creating wxSplitterWindow in the splitter sample for more details.

See also

GetSashGravity()

void wxSplitterWindow::SetSashInvisible (bool *invisible* = `true`)

Sets whether the sash should be invisible, even when the window is split.

When the sash is invisible, it doesn't appear on the screen at all and, in particular, doesn't allow the user to resize the windows.

Remarks

Only sets the internal variable; does not update the display.

Parameters

invisible	If true, the sash is always invisible, else it is shown when the window is split.

See also

IsSashInvisible()

Since

2.9.4

void wxSplitterWindow::SetSashPosition (int *position*, bool *redraw* = `true`)

Sets the sash position.

Parameters

position	The sash position in pixels.
redraw	If true, resizes the panes and redraws the sash and border.

Remarks

Does not currently check for an out-of-range value.

See also

GetSashPosition()

void wxSplitterWindow::SetSplitMode (int *mode*)

Sets the split mode.

Parameters

mode	Can be wxSPLIT_VERTICAL or wxSPLIT_HORIZONTAL.

Remarks

Only sets the internal variable; does not update the display.

See also

GetSplitMode(), SplitVertically(), SplitHorizontally().

virtual bool wxSplitterWindow::SplitHorizontally (wxWindow * *window1*, wxWindow * *window2*, int *sashPosition* = 0) [virtual]

Initializes the top and bottom panes of the splitter window.

The child windows are shown if they are currently hidden.

Parameters

window1	The top pane.
window2	The bottom pane.
sashPosition	The initial position of the sash. If this value is positive, it specifies the size of the upper pane. If it is negative, its absolute value gives the size of the lower pane. Finally, specify 0 (default) to choose the default position (half of the total window height).

Returns

true if successful, false otherwise (the window was already split).

Remarks

This should be called if you wish to initially view two panes. It can also be called at any subsequent time, but the application should check that the window is not currently split using IsSplit().

See also

SplitVertically(), IsSplit(), Unsplit()

virtual bool wxSplitterWindow::SplitVertically (wxWindow * *window1*, wxWindow * *window2*, int *sashPosition* = 0) [virtual]

Initializes the left and right panes of the splitter window.

The child windows are shown if they are currently hidden.

Parameters

window1	The left pane.
window2	The right pane.
sashPosition	The initial position of the sash. If this value is positive, it specifies the size of the left pane. If it is negative, it is absolute value gives the size of the right pane. Finally, specify 0 (default) to choose the default position (half of the total window width).

Returns

true if successful, false otherwise (the window was already split).

Remarks

This should be called if you wish to initially view two panes. It can also be called at any subsequent time, but the application should check that the window is not currently split using IsSplit().

See also

SplitHorizontally(), IsSplit(), Unsplit().

bool wxSplitterWindow::Unsplit (wxWindow * *toRemove* = NULL)

Unsplits the window.

Parameters

toRemove	The pane to remove, or NULL to remove the right or bottom pane.

Returns

true if successful, false otherwise (the window was not split).

Remarks

This call will not actually delete the pane being removed; it calls OnUnsplit() which can be overridden for the desired behaviour. By default, the pane being removed is hidden.

See also

SplitHorizontally(), SplitVertically(), IsSplit(), OnUnsplit()

void wxSplitterWindow::UpdateSize ( )

Causes any pending sizing of the sash and child panes to take place immediately.

Such resizing normally takes place in idle time, in order to wait for layout to be completed. However, this can cause unacceptable flicker as the panes are resized after the window has been shown. To work around this, you can perform window layout (for example by sending a size event to the parent window), and then call this function, before showing the top-level window.

21.709 wxStack< T > Class Template Reference

`#include <wx/stack.h>`

21.709.1 Detailed Description

template<typename T>class wxStack< T >

wxStack<T> is similar to `std::stack` and can be used exactly like it.

If wxWidgets is compiled in STL mode, wxStack will just be a typedef to `std::stack` but the advantage of this class is that it is also available on the (rare) platforms where STL is not, so using it makes the code marginally more portable. If you only target the standard desktop platforms, please always use `std::stack` directly instead.

The main difference of this class compared to the standard version is that it always uses wxVector<T> as the underlying container and doesn't allow specifying an alternative container type. Another missing part is that the comparison operators between wxStacks are not currently implemented. Other than that, this class is exactly the same as `std::stack`, so please refer to the STL documentation for further information.

Library: None; this class implementation is entirely header-based.

Category: Containers

See also

Container Classes, wxVector<T>

Since

2.9.2

Public Types

- typedef wxVector< T > container_type
  Type of the underlying container used.
- typedef container_type::size_type size_type
  Type returned by size() method.
- typedef container_type::value_type value_type
  Type of the elements stored in the stack.

Public Member Functions

- bool empty () const
  Return whether the stack is currently empty.
- size_type size () const
  Return the number of elements in the stack.
- void push (const value_type &val)
  Adds an element to the stack.
- void pop ()
  Removes the element currently on top of the stack.

- wxStack ()
  Stack can be created either empty or initialized with the contents of an existing compatible container.
- wxStack (const container_type &cont)
  Stack can be created either empty or initialized with the contents of an existing compatible container.

- value_type & top ()
  Return the element on top of the stack.
- const value_type & top ()
  Return the element on top of the stack.

21.709.2 Member Typedef Documentation

template<typename T > typedef wxVector<T> wxStack< T >::container_type

Type of the underlying container used.

template<typename T > typedef container_type::size_type wxStack< T >::size_type

Type returned by size() method.

template<typename T > typedef container_type::value_type wxStack< T >::value_type

Type of the elements stored in the stack.

21.709.3 Constructor & Destructor Documentation

template<typename T > wxStack< T >::wxStack ( )

Stack can be created either empty or initialized with the contents of an existing compatible container.

template<typename T > wxStack< T >::wxStack (const container_type & *cont*) `[explicit]`

Stack can be created either empty or initialized with the contents of an existing compatible container.

21.709.4 Member Function Documentation

template<typename T > bool wxStack< T >::empty ( ) const

Return whether the stack is currently empty.

template<typename T > void wxStack< T >::pop ( )

Removes the element currently on top of the stack.

template<typename T > void wxStack< T >::push (const value_type & *val*)

Adds an element to the stack.

template<typename T > size_type wxStack< T >::size ( ) const

Return the number of elements in the stack.

template<typename T > value_type& wxStack< T >::top ( )

Return the element on top of the stack.

template<typename T > const value_type& wxStack< T >::top ( )

Return the element on top of the stack.

21.710 wxStackFrame Class Reference

`#include <wx/stackwalk.h>`

21.710.1 Detailed Description

wxStackFrame represents a single stack frame, or a single function in the call stack, and is used exclusively together with wxStackWalker, see there for a more detailed discussion.

Library: wxBase

Category: Debugging

See also

wxStackWalker

Public Member Functions

- void * GetAddress () const

  Return the address of this frame.
- wxString GetFileName () const

  Return the name of the file containing this frame, empty if unavailable (typically because debug info is missing).
- size_t GetLevel () const

  Get the level of this frame (deepest/innermost one is 0).
- size_t GetLine () const

  Return the line number of this frame, 0 if unavailable.
- wxString GetModule () const

  Get the module this function belongs to (empty if not available).
- wxString GetName () const

  Return the unmangled (if possible) name of the function containing this frame.
- size_t GetOffset () const

  Return the return address of this frame.
- virtual bool GetParam (size_t n, wxString *type, wxString *name, wxString *value) const

  Get the name, type and value (in text form) of the given parameter.
- virtual size_t GetParamCount () const

  Return the number of parameters of this function (may return 0 if we can't retrieve the parameters info even although the function does have parameters).
- bool HasSourceLocation () const

  Return true if we have the file name and line number for this frame.

21.710.2 Member Function Documentation

void* wxStackFrame::GetAddress ( ) const

Return the address of this frame.

wxString wxStackFrame::GetFileName ( ) const

Return the name of the file containing this frame, empty if unavailable (typically because debug info is missing).

Use HasSourceLocation() to check whether the file name is available.

size_t wxStackFrame::GetLevel ( ) const

Get the level of this frame (deepest/innermost one is 0).

size_t wxStackFrame::GetLine ( ) const

Return the line number of this frame, 0 if unavailable.

See also

GetFileName()

wxString wxStackFrame::GetModule ( ) const

Get the module this function belongs to (empty if not available).

wxString wxStackFrame::GetName ( ) const

Return the unmangled (if possible) name of the function containing this frame.

size_t wxStackFrame::GetOffset ( ) const

Return the return address of this frame.

virtual bool wxStackFrame::GetParam (size_t *n*, wxString * *type*, wxString * *name*, wxString * *value*) const `[virtual]`

Get the name, type and value (in text form) of the given parameter.

Any pointer may be NULL if you're not interested in the corresponding value.

Return true if at least some values could be retrieved. This function currently is only implemented under Win32 and requires a PDB file.

virtual size_t wxStackFrame::GetParamCount ( ) const `[virtual]`

Return the number of parameters of this function (may return 0 if we can't retrieve the parameters info even although the function does have parameters).

bool wxStackFrame::HasSourceLocation ( ) const

Return true if we have the file name and line number for this frame.

21.711 wxStackWalker Class Reference

`#include <wx/stackwalk.h>`

21.711.1 Detailed Description

wxStackWalker allows an application to enumerate, or walk, the stack frames (the function callstack).

It is mostly useful in only two situations: inside wxApp::OnFatalException function to programmatically get the location of the crash and, in debug builds, in wxApp::OnAssertFailure to report the caller of the failed assert.

wxStackWalker works by repeatedly calling the wxStackWalker::OnStackFrame method for each frame in the stack, so to use it you must derive your own class from it and override this method.

This class will not return anything except raw stack frame addresses if the debug information is not available. Under Win32 this means that the PDB file matching the program being executed should be present. Note that if you use Microsoft Visual C++ compiler, you can create PDB files even for the programs built in release mode and it doesn't affect the program size (at least if you don't forget to add `/opt`:ref option which is suppressed by using `/debug` linker option by default but should be always enabled for release builds). Under Unix, you need to compile your program with debugging information (usually using `-g` compiler and linker options) to get the file and line numbers information, however function names should be available even without it. Of course, all this is only true if you build using a recent enough version of GNU libc which provides the `backtrace()` function needed to walk the stack.

See Debugging for how to make it available.

Library: wxBase

Category: Debugging

See also

wxStackFrame

Public Member Functions

- wxStackWalker (const char *argv0=NULL)

  Constructor does nothing, use Walk() to walk the stack.
- virtual ~wxStackWalker ()

  Destructor does nothing neither but should be virtual as this class is used as a base one.
- virtual void Walk (size_t skip=1, size_t maxDepth=wxSTACKWALKER_MAX_DEPTH)

  Enumerate stack frames from the current location, skipping the initial number of them (this can be useful when Walk() is called from some known location and you don't want to see the first few frames anyhow; also notice that Walk() frame itself is not included if skip = 1).
- virtual void WalkFromException (size_t maxDepth=wxSTACKWALKER_MAX_DEPTH)

  Enumerate stack frames from the location of uncaught exception.

Protected Member Functions

- virtual void OnStackFrame (const wxStackFrame &frame)=0

  This function must be overidden to process the given frame.

21.711.2 Constructor & Destructor Documentation

wxStackWalker::wxStackWalker (const char * *argv0* = `NULL`)

Constructor does nothing, use Walk() to walk the stack.

virtual wxStackWalker::~wxStackWalker ( ) `[virtual]`

Destructor does nothing neither but should be virtual as this class is used as a base one.

21.711.3 Member Function Documentation

virtual void wxStackWalker::OnStackFrame (const wxStackFrame & *frame*) `[protected], [pure virtual]`

This function must be overidden to process the given frame.

virtual void wxStackWalker::Walk (size_t *skip* = 1, size_t *maxDepth* = wxSTACKWALKER_MAX_DEPTH) `[virtual]`

Enumerate stack frames from the current location, skipping the initial number of them (this can be useful when Walk() is called from some known location and you don't want to see the first few frames anyhow; also notice that Walk() frame itself is not included if skip = 1).

Up to *maxDepth* frames are walked from the innermost to the outermost one. It defaults to wxSTACKWALKER_MAX_DEPTH.

virtual void wxStackWalker::WalkFromException (size_t *maxDepth* = wxSTACKWALKER_MAX_DEPTH) `[virtual]`

Enumerate stack frames from the location of uncaught exception.

This method can only be called from wxApp::OnFatalException().

Up to *maxDepth* frames are walked from the innermost to the outermost one. It defaults to wxSTACKWALKER_MAX_DEPTH.

21.712 wxStandardPaths Class Reference

`#include <wx/stdpaths.h>`

21.712.1 Detailed Description

wxStandardPaths returns the standard locations in the file system and should be used by applications to find their data files in a portable way.

Note that you must not create objects of class wxStandardPaths directly, but use the global standard paths object returned by wxStandardPaths::Get() (which can be of a type derived from wxStandardPaths and not of exactly this type) and call the methods you need on it. The object returned by Get() may be customized by overriding wxAppTraits::GetStandardPaths() methods.

In the description of the methods below, the example return values are given for the Unix, Windows and Mac OS X systems, however please note that these are just the examples and the actual values may differ. For example, under Windows: the system administrator may change the standard directories locations, e.g. the Windows directory may be named `"W:\Win2003"` instead of the default `"C:\Windows"`.

Notice that in the examples below the string `appinfo` may be either just the application name (as returned by wxApp::GetAppName()) or a combination of the vendor name (wxApp::GetVendorName()) and the application name, with a path separator between them. By default, only the application name is used, use UseAppInfo() to change this.

The other placeholders should be self-explanatory: the string `username` should be replaced with the value the name of the currently logged in user. and `prefix` is only used under Unix and is `/usr/local` by default but may be changed using wxStandardPaths::SetInstallPrefix().

The directories returned by the methods of this class may or may not exist. If they don't exist, it's up to the caller to create them, wxStandardPaths doesn't do it.

Finally note that these functions only work with standardly packaged applications. I.e. under Unix you should follow the standard installation conventions and under Mac you should create your application bundle according to the Apple guidelines. Again, this class doesn't help you to do it.

This class is MT-safe: its methods may be called concurrently from different threads without additional locking.

Library: wxBase

Category: File Handling

See also

wxFileConfig

Public Types

- enum ResourceCat {
  ResourceCat_None,
  ResourceCat_Messages }

  Possible values for category parameter of GetLocalizedResourcesDir().

Public Member Functions

- void DontIgnoreAppSubDir ()

  MSW-specific function undoing the effect of IgnoreAppSubDir() calls.
- virtual wxString GetAppDocumentsDir () const

  Return the directory for the document files used by this application.
- virtual wxString GetConfigDir () const

  Return the directory containing the system config files.
- virtual wxString GetDataDir () const

  Return the location of the applications global, i.e. not user-specific, data files.
- virtual wxString GetDocumentsDir () const

  Return the directory containing the current user's documents.
- virtual wxString GetExecutablePath () const

  Return the directory and the filename for the current executable.
- wxString GetInstallPrefix () const

  Return the program installation prefix, e.g. `/usr`, `/opt` or `/home/zeitlin`.
- virtual wxString GetLocalDataDir () const

  Return the location for application data files which are host-specific and can't, or shouldn't, be shared with the other machines.
- virtual wxString GetLocalizedResourcesDir (const wxString &lang, ResourceCat category=ResourceCat_None) const

  Return the localized resources directory containing the resource files of the specified category for the given language.
- virtual wxString GetPluginsDir () const

  Return the directory where the loadable modules (plugins) live.
- virtual wxString GetResourcesDir () const

  Return the directory where the application resource files are located.
- virtual wxString GetTempDir () const

  Return the directory for storing temporary files.
- virtual wxString GetUserConfigDir () const

  Return the directory for the user config files:
- virtual wxString GetUserDataDir () const

  Return the directory for the user-dependent application data files:
- virtual wxString GetUserLocalDataDir () const

  Return the directory for user data files which shouldn't be shared with the other machines.
- void IgnoreAppSubDir (const wxString &subdirPattern)

  MSW-specific function to customize application directory detection.

- void IgnoreAppBuildSubDirs ()

  MSW-specific function to ignore all common build directories.
- void SetInstallPrefix (const wxString &prefix)

  Lets wxStandardPaths know about the real program installation prefix on a Unix system.
- void UseAppInfo (int info)

  Controls what application information is used when constructing paths that should be unique to this program, such as the application data directory, the plugins directory on Unix, etc.

Static Public Member Functions

- static wxStandardPaths & Get ()

  Returns reference to the unique global standard paths object.
- static wxString MSWGetShellDir (int csidl)

  Returns location of Windows shell special folder.

Protected Member Functions

- wxStandardPaths ()

  Protected default constructor.

21.712.2 Member Enumeration Documentation

enum wxStandardPaths::ResourceCat

Possible values for category parameter of GetLocalizedResourcesDir().

Enumerator

ResourceCat_None No special category, this is the default.

ResourceCat_Messages Message catalog resources category.

21.712.3 Constructor & Destructor Documentation

wxStandardPaths::wxStandardPaths ( ) [protected]

Protected default constructor.

This constructor is protected in order to prevent creation of objects of this class as Get() should be used instead to access the unique global wxStandardPaths object of the correct type.

21.712.4 Member Function Documentation

void wxStandardPaths::DontIgnoreAppSubDir ( )

MSW-specific function undoing the effect of IgnoreAppSubDir() calls.

After a call to this function the program directory will be exactly the directory containing the main application binary, i.e. it undoes the effect of any previous IgnoreAppSubDir() calls including the ones done indirectly by IgnoreApp↩ BuildSubDirs() called from the class constructor.

Since

2.9.1

static wxStandardPaths& wxStandardPaths::Get ( ) [static]

Returns reference to the unique global standard paths object.

virtual wxString wxStandardPaths::GetAppDocumentsDir ( ) const [virtual]

Return the directory for the document files used by this application.

If the application-specific directory doesn't exist, this function returns GetDocumentsDir().

Example return values:

- Unix: ~/appinfo
- Windows: "C:\Documents and Settings\username\My Documents\appinfo"
- Mac: ~/Documents/appinfo

Since

2.9.0

virtual wxString wxStandardPaths::GetConfigDir ( ) const [virtual]

Return the directory containing the system config files.

Example return values:

- Unix: /etc
- Windows: "C:\Documents and Settings\All Users\Application Data"
- Mac: /Library/Preferences

See also

wxFileConfig

virtual wxString wxStandardPaths::GetDataDir ( ) const [virtual]

Return the location of the applications global, i.e. not user-specific, data files.

Example return values:

- Unix: prefix/share/appinfo
- Windows: the directory where the executable file is located
- Mac: appinfo.app/Contents/SharedSupport bundle subdirectory

Under Unix (only) it is possible to override the default value returned from this function by setting the value of WX_APPNAME_DATA_DIR environment variable to the directory to use (where APPNAME is the upper-cased value of wxApp::GetAppName()). This is useful in order to be able to run applications using this function without installing them as you can simply set this environment variable to the source directory location to allow the application to find its files there.

See also

GetLocalDataDir()

virtual wxString wxStandardPaths::GetDocumentsDir ( ) const `[virtual]`

Return the directory containing the current user's documents.

Example return values:

- Unix: ~ (the home directory)
- Windows: `"C:\Documents and Settings\username\My Documents"`
- Mac: `~/Documents`

Since

2.7.0

See also

GetAppDocumentsDir()

virtual wxString wxStandardPaths::GetExecutablePath ( ) const `[virtual]`

Return the directory and the filename for the current executable.

Example return values:

- Unix: `/usr/local/bin/exename`
- Windows: `"C:\Programs\AppFolder\exename.exe"`
- Mac: `/Applications/exename`.app/Contents/MacOS/exename

wxString wxStandardPaths::GetInstallPrefix ( ) const

Return the program installation prefix, e.g. `/usr`, `/opt` or `/home/zeitlin`.

If the prefix had been previously by SetInstallPrefix(), returns that value, otherwise tries to determine it automatically (Linux only right now) and finally returns the default `/usr/local` value if it failed.

Note

This function is only available under Unix platforms (but not limited to wxGTK mentioned below).

Availability: only available for the wxGTK port.

virtual wxString wxStandardPaths::GetLocalDataDir ( ) const `[virtual]`

Return the location for application data files which are host-specific and can't, or shouldn't, be shared with the other machines.

This is the same as GetDataDir() except under Unix where it returns `/etc/appinfo`.

virtual wxString wxStandardPaths::GetLocalizedResourcesDir (const wxString & *lang*, ResourceCat *category* = ResourceCat_None) const `[virtual]`

Return the localized resources directory containing the resource files of the specified category for the given language.

In general this is just the same as *lang* subdirectory of GetResourcesDir() (or `lang.lproj` under Mac OS X) but is something quite different for message catalog category under Unix where it returns the standard `prefix/share/locale/lang/LC_MESSAGES` directory.

Since

2.7.0

virtual wxString wxStandardPaths::GetPluginsDir ( ) const `[virtual]`

Return the directory where the loadable modules (plugins) live.

Example return values:

- Unix: `prefix/lib/appinfo`
- Windows: the directory of the executable file
- Mac: `appinfo.app/Contents/PlugIns` bundle subdirectory

See also

wxDynamicLibrary

virtual wxString wxStandardPaths::GetResourcesDir ( ) const `[virtual]`

Return the directory where the application resource files are located.

The resources are the auxiliary data files needed for the application to run and include, for example, image and sound files it might use.

This function is the same as GetDataDir() for all platforms except Mac OS X. Example return values:

- Unix: `prefix/share/appinfo`
- Windows: the directory where the executable file is located
- Mac: `appinfo.app/Contents/Resources` bundle subdirectory

Since

2.7.0

See also

GetLocalizedResourcesDir()

virtual wxString wxStandardPaths::GetTempDir ( ) const `[virtual]`

Return the directory for storing temporary files.

To create unique temporary files, it is best to use wxFileName::CreateTempFileName for correct behaviour when multiple processes are attempting to create temporary files.

Since

2.7.2

virtual wxString wxStandardPaths::GetUserConfigDir ( ) const `[virtual]`

Return the directory for the user config files:

- Unix: ~ (the home directory)
- Windows: "C:\Documents and Settings\username\Application Data"
- Mac: ~/Library/Preferences

Only use this method if you have a single configuration file to put in this directory, otherwise GetUserDataDir() is more appropriate as the latter adds `appinfo` to the path, unlike this function.

virtual wxString wxStandardPaths::GetUserDataDir ( ) const `[virtual]`

Return the directory for the user-dependent application data files:

- Unix: ~/.appinfo
- Windows: "C:\Documents and Settings\username\Application Data\appinfo"
- Mac: "~/Library/Application Support/appinfo"

virtual wxString wxStandardPaths::GetUserLocalDataDir ( ) const `[virtual]`

Return the directory for user data files which shouldn't be shared with the other machines.

This is the same as GetUserDataDir() for all platforms except Windows where it returns "C:\Documents and Settings\username\Local Settings\Application Data\appinfo"

void wxStandardPaths::IgnoreAppBuildSubDirs ( )

MSW-specific function to ignore all common build directories.

This function calls IgnoreAppSubDir() with all common values for build directory, e.g. "debug" and "release".

It is called by the class constructor and so the build directories are always ignored by default. You may use DontIgnoreAppSubDir() to avoid ignoring them if this is inappropriate for your application.

Since

2.9.1

void wxStandardPaths::IgnoreAppSubDir (const wxString & *subdirPattern*)

MSW-specific function to customize application directory detection.

This class supposes that data, plugins &c files are located under the program directory which is the directory containing the application binary itself. But sometimes this binary may be in a subdirectory of the main program directory, e.g. this happens in at least the following common cases:

- The program is in "bin" subdirectory of the installation directory.
- The program is in "debug" subdirectory of the directory containing sources and data files during development

By calling this function you instruct the class to remove the last component of the path if it matches its argument. Notice that it may be called more than once, e.g. you can call both IgnoreAppSubDir("bin") and IgnoreAppSub↩ Dir("debug") to take care of both production and development cases above but that each call will only remove the last path component. Finally note that the argument can contain wild cards so you can also call IgnoreAppSub↩ Dir("vc∗msw∗") to ignore all build directories at once when using wxWidgets-inspired output directories names.

Since

2.9.1

See also

IgnoreAppBuildSubDirs()

Parameters

subdirPattern	The subdirectory containing the application binary which should be ignored when determining the top application directory. The pattern is case-insensitive and may contain wild card characters '?' and '∗'.

static wxString wxStandardPaths::MSWGetShellDir (int *csidl*) [static]

Returns location of Windows shell special folder.

This function is, by definition, MSW-specific. It can be used to access pre-defined shell directories not covered by the existing methods of this class, e.g.:

```
#ifdef __WXMSW__
    // get the location of files waiting to be burned on a CD
    wxString cdburnArea =
        wxStandardPaths::MSWGetShellDir(CSIDL_CDBURN_AREA);
#endif // __WXMSW__
```

Parameters

csidl	

Since

2.9.1

void wxStandardPaths::SetInstallPrefix (const wxString & *prefix*)

Lets wxStandardPaths know about the real program installation prefix on a Unix system.

By default, the value returned by GetInstallPrefix() is used.

Although under Linux systems the program prefix may usually be determined automatically, portable programs should call this function. Usually the prefix is set during program configuration if using GNU autotools and so it is enough to pass its value defined in config.h to this function.

Note

This function is only available under Unix platforms (but not limited to wxGTK mentioned below).

Availability: only available for the wxGTK port.

void wxStandardPaths::UseAppInfo (int *info*)

Controls what application information is used when constructing paths that should be unique to this program, such as the application data directory, the plugins directory on Unix, etc.

Valid values for *info* are:

- `AppInfo_None`: don't use neither application nor vendor name in the paths.
- `AppInfo_AppName`: use the application name in the paths.
- `AppInfo_VendorName`: use the vendor name in the paths, usually used combined with AppInfo_App↩ Name, i.e. as

```
AppInfo_AppName |
   AppInfo_VendorName
```

By default, only the application name is used.

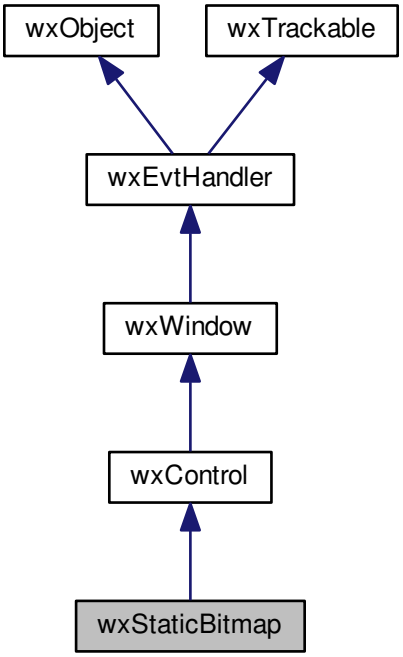
Since

2.9.0

21.713 wxStaticBitmap Class Reference

`#include <wx/statbmp.h>`

Inheritance diagram for wxStaticBitmap:



21.713.1 Detailed Description

A static bitmap control displays a bitmap.

Native implementations on some platforms are only meant for display of the small icons in the dialog boxes. In particular, under Windows 9x the size of bitmap is limited to 64∗64 pixels.

If you want to display larger images portably, you may use generic implementation wxGenericStaticBitmap declared in <wx/generic/statbmpg.h>.

Notice that for the best results, the size of the control should be the same as the size of the image displayed in it, as happens by default if it's not resized explicitly. Otherwise, behaviour depends on the platform: under MSW, the bitmap is drawn centred inside the control, while elsewhere it is drawn at the origin of the control.

Library: wxCore

Category: Controls

See also

wxBitmap

Public Member Functions

- wxStaticBitmap ()

Default constructor.

- wxStaticBitmap (wxWindow *parent, wxWindowID id, const wxBitmap &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxStaticBitmapNameStr)

  Constructor, creating and showing a static bitmap control.

- bool Create (wxWindow *parent, wxWindowID id, const wxBitmap &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxStaticBitmapNameStr)

  Creation function, for two-step construction.

- virtual wxBitmap GetBitmap () const

  Returns the bitmap currently used in the control.

- virtual wxIcon GetIcon () const

  Returns the icon currently used in the control.

- virtual void SetBitmap (const wxBitmap &label)

  Sets the bitmap label.

- virtual void SetIcon (const wxIcon &label)

  Sets the label to the given icon.

Additional Inherited Members

21.713.2 Constructor & Destructor Documentation

wxStaticBitmap::wxStaticBitmap ( )

Default constructor.

wxStaticBitmap::wxStaticBitmap (wxWindow * *parent*, wxWindowID *id*, const wxBitmap & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxStaticBitmapNameStr`)

Constructor, creating and showing a static bitmap control.

Parameters

parent	Parent window. Should not be NULL.
id	Control identifier. A value of -1 denotes a default value.
label	Bitmap label.
pos	Window position.
size	Window size.
style	Window style. See wxStaticBitmap.
name	Window name.

See also

Create()

21.713.3 Member Function Documentation

bool wxStaticBitmap::Create (wxWindow * *parent*, wxWindowID *id*, const wxBitmap & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxStaticBitmapNameStr`)

Creation function, for two-step construction.

For details see wxStaticBitmap().

virtual wxBitmap wxStaticBitmap::GetBitmap ( ) const `[virtual]`

Returns the bitmap currently used in the control.

Notice that this method can be called even if SetIcon() had been used.

See also

SetBitmap()

virtual wxIcon wxStaticBitmap::GetIcon ( ) const `[virtual]`

Returns the icon currently used in the control.

Notice that this method can only be called if SetIcon() had been used: an icon can't be retrieved from the control if a bitmap had been set (using wxStaticBitmap::SetBitmap).

See also

SetIcon()

virtual void wxStaticBitmap::SetBitmap (const wxBitmap & *label*) `[virtual]`

Sets the bitmap label.

Parameters

label	The new bitmap.

See also

GetBitmap()

virtual void wxStaticBitmap::SetIcon (const wxIcon & *label*) `[virtual]`

Sets the label to the given icon.

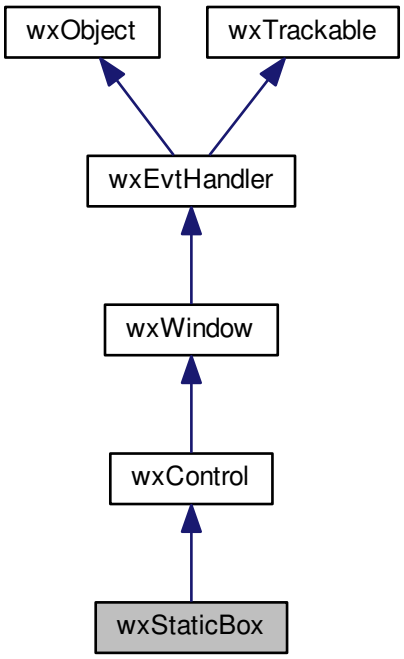
Parameters

label	The new icon.

21.714 wxStaticBox Class Reference

```
#include <wx/statbox.h>
```

Inheritance diagram for wxStaticBox:



21.714.1 Detailed Description

A static box is a rectangle drawn around other windows to denote a logical grouping of items.

Note that while the previous versions required that windows appearing inside a static box be created as its siblings (i.e. use the same parent as the static box itself), since wxWidgets 2.9.1 it is also possible to create them as children of wxStaticBox itself and you are actually encouraged to do it like this if compatibility with the previous versions is not important.

So the new recommended way to create static box is:

```
void MyFrame::CreateControls()
{
    wxPanel *panel = new wxPanel(this);
    wxStaticBox *box = new wxStaticBox(panel, wxID_ANY, "StaticBox");

    new wxStaticText(box, wxID_ANY "This window is a child of the staticbox");
    ...
}
```

While the compatible – and now deprecated – way is

```
wxStaticBox *box = new wxStaticBox(panel, wxID_ANY, "StaticBox");

new wxStaticText(panel, wxID_ANY "This window is a child of the panel");
...
```

Also note that there is a specialized wxSizer class (wxStaticBoxSizer) which can be used as an easier way to pack items into a static box.

Library: wxCore

Category: Controls

See also

wxStaticText, wxStaticBoxSizer

Public Member Functions

- wxStaticBox ()

  Default constructor.
- wxStaticBox (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxStaticBoxNameStr)

  Constructor, creating and showing a static box.
- virtual ~wxStaticBox ()

  Destructor, destroying the group box.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxStaticBoxNameStr)

  Creates the static box for two-step construction.

Additional Inherited Members

21.714.2 Constructor & Destructor Documentation

wxStaticBox::wxStaticBox ( )

Default constructor.

wxStaticBox::wxStaticBox (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxStaticBoxNameStr)

Constructor, creating and showing a static box.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
label	Text to be displayed in the static box, the empty string for no label.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Checkbox size. If wxDefaultSize is specified then a default size is chosen.
style	Window style. See wxStaticBox.
name	Window name.

See also

Create()

virtual wxStaticBox::~wxStaticBox ( ) [virtual]

Destructor, destroying the group box.

21.714.3 Member Function Documentation

bool wxStaticBox::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxStaticBoxNameStr`)

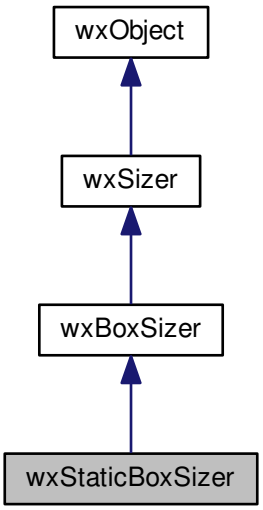
Creates the static box for two-step construction.

See wxStaticBox() for further details.

21.715 wxStaticBoxSizer Class Reference

`#include <wx/sizer.h>`

Inheritance diagram for wxStaticBoxSizer:



21.715.1 Detailed Description

wxStaticBoxSizer is a sizer derived from wxBoxSizer but adds a static box around the sizer.

The static box may be either created independently or the sizer may create it itself as a convenience. In any case, the sizer owns the wxStaticBox control and will delete it in the wxStaticBoxSizer destructor.

Note that since wxWidgets 2.9.1 you are encouraged to create the windows which are added to wxStaticBoxSizer as children of wxStaticBox itself, see this class documentation for more details.

Example of use of this class:

```
void MyFrame::CreateControls()
{
    wxPanel *panel = new wxPanel(this);
    ...
    wxStaticBoxSizer *sz = new wxStaticBoxSizer(
      wxVERTICAL, panel, "Box");
    sz->Add(new wxStaticText(sz->GetStaticBox(),
      wxID_ANY,
                             "This window is a child of the staticbox"));
```

```
    ...
}
```

Library: wxCore

Category: Window Layout

See also

wxSizer, wxStaticBox, wxBoxSizer, Sizers Overview

Public Member Functions

- wxStaticBoxSizer (wxStaticBox *box, int orient)

  This constructor uses an already existing static box.
- wxStaticBoxSizer (int orient, wxWindow *parent, const wxString &label=wxEmptyString)

  This constructor creates a new static box with the given label and parent window.
- wxStaticBox * GetStaticBox () const

  Returns the static box associated with the sizer.
- virtual wxSize CalcMin ()

  Implements the calculation of a box sizer's minimal.
- virtual void RecalcSizes ()

  Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

Additional Inherited Members

21.715.2 Constructor & Destructor Documentation

wxStaticBoxSizer::wxStaticBoxSizer (wxStaticBox * *box,* int *orient*)

This constructor uses an already existing static box.

Parameters

box	The static box to associate with the sizer (which will take its ownership).
orient	Can be either `wxVERTICAL` or `wxHORIZONTAL`.

wxStaticBoxSizer::wxStaticBoxSizer (int *orient,* wxWindow * *parent,* const wxString & *label* = wxEmptyString)

This constructor creates a new static box with the given label and parent window.

21.715.3 Member Function Documentation

virtual wxSize wxStaticBoxSizer::CalcMin ( ) `[virtual]`

Implements the calculation of a box sizer's minimal.

It is used internally only and must not be called by the user. Documented for information.

Reimplemented from wxBoxSizer.

wxStaticBox* wxStaticBoxSizer::GetStaticBox ( ) const

Returns the static box associated with the sizer.

virtual void wxStaticBoxSizer::RecalcSizes ( ) [virtual]

Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

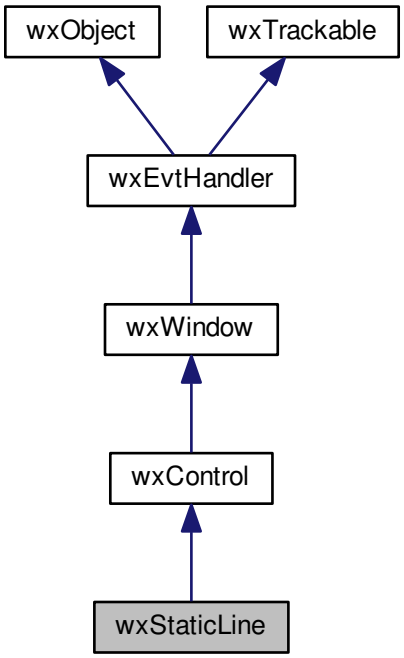
It is used internally only and must not be called by the user (call Layout() if you want to resize). Documented for information.

Reimplemented from wxBoxSizer.

21.716 wxStaticLine Class Reference

`#include <wx/statline.h>`

Inheritance diagram for wxStaticLine:



21.716.1 Detailed Description

A static line is just a line which may be used in a dialog to separate the groups of controls.

The line may be only vertical or horizontal. Moreover, not all ports (notably not wxGTK) support specifying the transversal direction of the line (e.g. height for a horizontal line) so for maximal portability you should specify it as wxDefaultCoord.

Styles

This class supports the following styles:

- wxLI_HORIZONTAL: Creates a horizontal line.
- wxLI_VERTICAL: Creates a vertical line.

Library: wxCore

Category: Controls

See also

wxStaticBox

Public Member Functions

- wxStaticLine ()

  Default constructor.
- wxStaticLine (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxLI_HORIZONTAL, const wxString &name=wxStaticLineNameStr)

  Constructor, creating and showing a static line.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxLI_HORIZONTAL, const wxString &name=wxStaticLineNameStr)

  Creates the static line for two-step construction.
- bool IsVertical () const

  Returns true if the line is vertical, false if horizontal.

Static Public Member Functions

- static int GetDefaultSize ()

  This static function returns the size which will be given to the smaller dimension of the static line, i.e.

Additional Inherited Members

21.716.2 Constructor & Destructor Documentation

wxStaticLine::wxStaticLine ( )

Default constructor.

wxStaticLine::wxStaticLine (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxLI_HORIZONTAL, const wxString & *name* = `wxStaticLineNameStr`)

Constructor, creating and showing a static line.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value wxID_ANY indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Size. Note that either the height or the width (depending on whether the line if horizontal or vertical) is ignored.
style	Window style (either wxLI_HORIZONTAL or wxLI_VERTICAL).
name	Window name.

See also

Create()

21.716.3 Member Function Documentation

bool wxStaticLine::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxLI_HORIZONTAL, const wxString & *name* = `wxStaticLineNameStr`)

Creates the static line for two-step construction.

See wxStaticLine() for further details.

static int wxStaticLine::GetDefaultSize ( ) `[static]`

This static function returns the size which will be given to the smaller dimension of the static line, i.e.

its height for a horizontal line or its width for a vertical one.

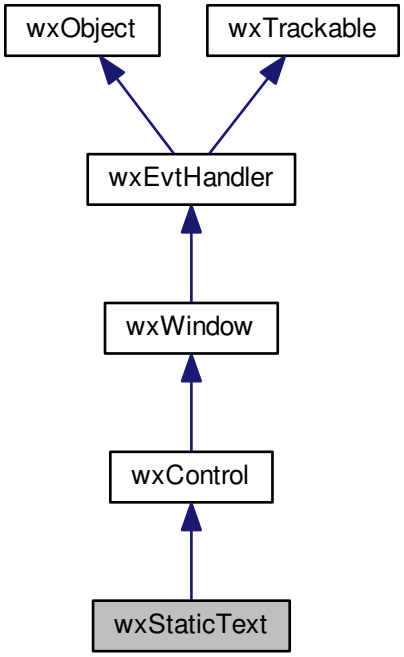
bool wxStaticLine::IsVertical ( ) const

Returns true if the line is vertical, false if horizontal.

21.717 wxStaticText Class Reference

`#include <wx/stattext.h>`

Inheritance diagram for wxStaticText:



21.717.1 Detailed Description

A static text control displays one or more lines of read-only text.

wxStaticText supports the three classic text alignments, label ellipsization i.e. replacing parts of the text with the ellipsis ("...") if the label doesn't fit into the provided space and also formatting markup with wxControl::SetLabel↩ Markup().

Styles

This class supports the following styles:

- wxALIGN_LEFT: Align the text to the left.
- wxALIGN_RIGHT: Align the text to the right.
- wxALIGN_CENTRE_HORIZONTAL: Center the text (horizontally).
- wxST_NO_AUTORESIZE: By default, the control will adjust its size to exactly fit to the size of the text when SetLabel() is called. If this style flag is given, the control will not change its size (this style is especially useful with controls which also have the `wxALIGN_RIGHT` or the `wxALIGN_CENTRE_HORIZONTAL` style because otherwise they won't make sense any longer after a call to SetLabel()).
- wxST_ELLIPSIZE_START: If the labeltext width exceeds the control width, replace the beginning of the label with an ellipsis; uses wxControl::Ellipsize.
- wxST_ELLIPSIZE_MIDDLE: If the label text width exceeds the control width, replace the middle of the label with an ellipsis; uses wxControl::Ellipsize.

- wxST_ELLIPSIZE_END: If the label text width exceeds the control width, replace the end of the label with an ellipsis; uses wxControl::Ellipsize.

Library: wxCore

Category: Controls

See also

wxStaticBitmap, wxStaticBox

Public Member Functions

- wxStaticText ()

  Default constructor.
- wxStaticText (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxStaticTextNameStr)

  Constructor, creating and showing a text control.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxStaticTextNameStr)

  Creation function, for two-step construction.
- bool IsEllipsized () const

  Returns true if the window styles for this control contains one of the `wxST_ELLIPSIZE_START`, `wxST_ELLIPSIZE_MIDDLE` or `wxST_ELLIPSIZE_END` styles.
- void Wrap (int width)

  This functions wraps the controls label so that each of its lines becomes at most width pixels wide if possible (the lines are broken at words boundaries so it might not be the case if words are too long).

Additional Inherited Members

21.717.2 Constructor & Destructor Documentation

wxStaticText::wxStaticText ( )

Default constructor.

wxStaticText::wxStaticText (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxStaticTextNameStr`)

Constructor, creating and showing a text control.

Parameters

parent	Parent window. Should not be NULL.
id	Control identifier. A value of -1 denotes a default value.
label	Text label.
pos	Window position.

size	Window size.
style	Window style. See wxStaticText.
name	Window name.

See also

Create()

21.717.3 Member Function Documentation

bool wxStaticText::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxStaticTextNameStr`)

Creation function, for two-step construction.

For details see wxStaticText().

bool wxStaticText::IsEllipsized ( ) const

Returns true if the window styles for this control contains one of the `wxST_ELLIPSIZE_START`, `wxST_ELLIPSIZE_MIDDLE` or `wxST_ELLIPSIZE_END` styles.

void wxStaticText::Wrap (int *width*)

This functions wraps the controls label so that each of its lines becomes at most *width* pixels wide if possible (the lines are broken at words boundaries so it might not be the case if words are too long).

If *width* is negative, no wrapping is done. Note that this width is not necessarily the total width of the control, since a few pixels for the border (depending on the controls border style) may be added.

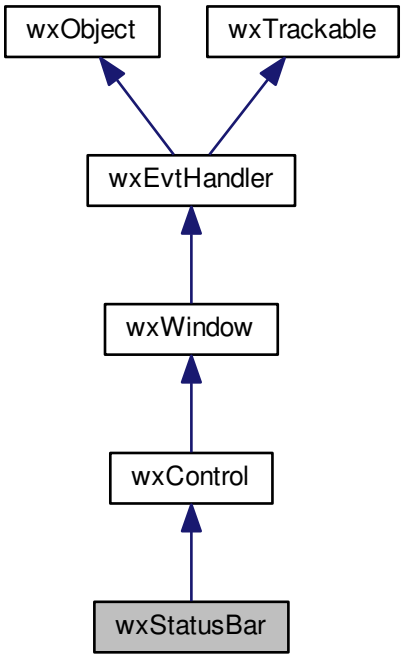
Since

2.6.2

21.718 wxStatusBar Class Reference

`#include <wx/statusbr.h>`

Inheritance diagram for wxStatusBar:



21.718.1 Detailed Description

A status bar is a narrow window that can be placed along the bottom of a frame to give small amounts of status information.

It can contain one or more fields, one or more of which can be variable length according to the size of the window.

wxStatusBar also maintains an independent stack of status texts for each field (see PushStatusText() and PopStatusText()).

Note that in wxStatusBar context, the terms *pane* and *field* are synonyms.

Styles

This class supports the following styles:

- wxSTB_SIZEGRIP: Displays a gripper at the right-hand side of the status bar which can be used to resize the parent window.
- wxSTB_SHOW_TIPS: Displays tooltips for those panes whose status text has been ellipsized/truncated because the status text doesn't fit the pane width. Note that this style has effect only on wxGTK (with GTK+ >= 2.12) currently.
- wxSTB_ELLIPSIZE_START: Replace the beginning of the status texts with an ellipsis when the status text widths exceed the status bar pane's widths (uses wxControl::Ellipsize).
- wxSTB_ELLIPSIZE_MIDDLE: Replace the middle of the status texts with an ellipsis when the status text widths exceed the status bar pane's widths (uses wxControl::Ellipsize).

- wxSTB_ELLIPSIZE_END: Replace the end of the status texts with an ellipsis when the status text widths exceed the status bar pane's widths (uses wxControl::Ellipsize).
- wxSTB_DEFAULT_STYLE: The default style: includes `wxSTB_SIZEGRIP|wxSTB_SHOW_TIPS|wxSTB_ELLIPSIZE_END|wxFULL_REPAINT_ON_RESIZE`.

Remarks

It is possible to create controls and other windows on the status bar. Position these windows from an OnSize() event handler.
Notice that only the first 127 characters of a string will be shown in status bar fields under pre-XP MSW systems (or even under later systems if a proper manifest indicating that the program uses version 6 of common controls library is not used). This is a limitation of the native control on these platforms.

Library: wxCore

Category: Miscellaneous Windows

See also

wxStatusBarPane, wxFrame, Status Bar Sample

Public Member Functions

- wxStatusBar ()

  Default ctor.
- wxStatusBar (wxWindow *parent, wxWindowID id=wxID_ANY, long style=wxSTB_DEFAULT_STYLE, const wxString &name=wxStatusBarNameStr)

  Constructor, creating the window.
- virtual ~wxStatusBar ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, long style=wxSTB_DEFAULT_STYLE, const wxString &name=wxStatusBarNameStr)

  Creates the window, for two-step construction.
- virtual bool GetFieldRect (int i, wxRect &rect) const

  Returns the size and position of a field's internal bounding rectangle.
- int GetFieldsCount () const

  Returns the number of fields in the status bar.
- const wxStatusBarPane & GetField (int n) const

  Returns the wxStatusBarPane representing the n-th field.
- wxSize GetBorders () const

  Returns the horizontal and vertical borders used when rendering the field text inside the field area.
- virtual wxString GetStatusText (int i=0) const

  Returns the string associated with a status bar field.
- int GetStatusWidth (int n) const

  Returns the width of the n-th field.
- int GetStatusStyle (int n) const

  Returns the style of the n-th field.
- void PopStatusText (int field=0)

  Restores the text to the value it had before the last call to PushStatusText().
- void PushStatusText (const wxString &string, int field=0)

  Saves the current field text in a per-field stack, and sets the field text to the string passed as argument.

- virtual void SetFieldsCount (int number=1, const int ∗widths=NULL)

  Sets the number of fields, and optionally the field widths.
- virtual void SetMinHeight (int height)

  Sets the minimal possible height for the status bar.
- virtual void SetStatusStyles (int n, const int ∗styles)

  Sets the styles of the fields in the status line which can make fields appear flat or raised instead of the standard sunken 3D border.
- virtual void SetStatusText (const wxString &text, int i=0)

  Sets the status text for the i-th field.
- virtual void SetStatusWidths (int n, const int ∗widths_field)

  Sets the widths of the fields in the status line.

Additional Inherited Members

21.718.2 Constructor & Destructor Documentation

wxStatusBar::wxStatusBar ( )

Default ctor.

wxStatusBar::wxStatusBar (wxWindow ∗ *parent*, wxWindowID *id* = wxID_ANY, long *style* = wxSTB_DEFAULT_STYLE, const wxString & *name* = `wxStatusBarNameStr`)

Constructor, creating the window.

Parameters

parent	The window parent, usually a frame.
id	The window identifier. It may take a value of -1 to indicate a default value.
style	The window style. See wxStatusBar.
name	The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

See also

Create()

virtual wxStatusBar::∼wxStatusBar ( ) `[virtual]`

Destructor.

21.718.3 Member Function Documentation

bool wxStatusBar::Create (wxWindow ∗ *parent*, wxWindowID *id* = wxID_ANY, long *style* = wxSTB_DEFAULT_STYLE, const wxString & *name* = `wxStatusBarNameStr`)

Creates the window, for two-step construction.

See wxStatusBar() for details.

wxSize wxStatusBar::GetBorders ( ) const

Returns the horizontal and vertical borders used when rendering the field text inside the field area.

Note that the rect returned by GetFieldRect() already accounts for the presence of horizontal and vertical border returned by this function.

const wxStatusBarPane& wxStatusBar::GetField (int *n*) const

Returns the wxStatusBarPane representing the *n-th* field.

virtual bool wxStatusBar::GetFieldRect (int *i*, wxRect & *rect*) const [virtual]

Returns the size and position of a field's internal bounding rectangle.

Parameters

i	The field in question.
rect	The rectangle values are placed in this variable.

Returns

true if the field index is valid, false otherwise.

wxPerl Note: In wxPerl this function returns a `Wx::Rect` if the field index is valid, `undef` otherwise.

See also

wxRect

int wxStatusBar::GetFieldsCount ( ) const

Returns the number of fields in the status bar.

int wxStatusBar::GetStatusStyle (int *n*) const

Returns the style of the *n-th* field.

See wxStatusBarPane::GetStyle() for more info.

virtual wxString wxStatusBar::GetStatusText (int *i* = 0) const [virtual]

Returns the string associated with a status bar field.

Parameters

i	The number of the status field to retrieve, starting from zero.

Returns

The status field string if the field is valid, otherwise the empty string.

See also

SetStatusText()

int wxStatusBar::GetStatusWidth (int *n*) const

Returns the width of the *n-th* field.

See wxStatusBarPane::GetWidth() for more info.

void wxStatusBar::PopStatusText (int *field* = 0)

Restores the text to the value it had before the last call to PushStatusText().

Notice that if SetStatusText() had been called in the meanwhile, PopStatusText() will not change the text, i.e. it does not override explicit changes to status text but only restores the saved text if it hadn't been changed since.

See also

PushStatusText()

void wxStatusBar::PushStatusText (const wxString & *string,* int *field* = 0)

Saves the current field text in a per-field stack, and sets the field text to the string passed as argument.

See also

PopStatusText()

virtual void wxStatusBar::SetFieldsCount (int *number* = 1, const int * *widths* = NULL) [virtual]

Sets the number of fields, and optionally the field widths.

Parameters

number	The number of fields. If this is greater than the previous number, then new fields with empty strings will be added to the status bar.
widths	An array of n integers interpreted in the same way as in SetStatusWidths().

wxPerl Note: In wxPerl this function accepts only the *number* parameter. Use SetStatusWidths to set the field widths.

virtual void wxStatusBar::SetMinHeight (int *height*) [virtual]

Sets the minimal possible height for the status bar.

The real height may be bigger than the height specified here depending on the size of the font used by the status bar.

virtual void wxStatusBar::SetStatusStyles (int *n,* const int * *styles*) [virtual]

Sets the styles of the fields in the status line which can make fields appear flat or raised instead of the standard sunken 3D border.

Parameters

n	The number of fields in the status bar. Must be equal to the number passed to SetFields↩ Count() the last time it was called.

styles	Contains an array of *n* integers with the styles for each field. There are four possible styles: • `wxSB_NORMAL` (default): The field appears with the default native border. • `wxSB_FLAT`: No border is painted around the field so that it appears flat. • `wxSB_RAISED`: A raised 3D border is painted around the field. • `wxSB_SUNKEN`: A sunken 3D border is painted around the field (this style is new since wxWidgets 2.9.5).

virtual void wxStatusBar::SetStatusText (const wxString & *text*, int *i* = 0) [virtual]

Sets the status text for the *i-th* field.

The given text will replace the current text.

Note that if PushStatusText() had been called before the new text will also replace the last saved value to make sure that the next call to PopStatusText() doesn't restore the old value, which was overwritten by the call to this function.

Parameters

text	The text to be set. Use an empty string ("") to clear the field.
i	The field to set, starting from zero.

See also

GetStatusText(), wxFrame::SetStatusText

virtual void wxStatusBar::SetStatusWidths (int *n*, const int * *widths_field*) [virtual]

Sets the widths of the fields in the status line.

There are two types of fields: **fixed** widths and **variable** width fields. For the fixed width fields you should specify their (constant) width in pixels. For the variable width fields, specify a negative number which indicates how the field should expand: the space left for all variable width fields is divided between them according to the absolute value of this number. A variable width field with width of -2 gets twice as much of it as a field with width -1 and so on.

For example, to create one fixed width field of width 100 in the right part of the status bar and two more fields which get 66% and 33% of the remaining space correspondingly, you should use an array containing -2, -1 and 100.

Parameters

n	The number of fields in the status bar. Must be equal to the number passed to SetFields↩ Count() the last time it was called.
widths_field	Contains an array of n integers, each of which is either an absolute status field width in pixels if positive or indicates a variable width field if negative. The special value NULL means that all fields should get the same width.

Remarks

The widths of the variable fields are calculated from the total width of all fields, minus the sum of widths of the non-variable fields, divided by the number of variable fields.

wxPerl Note: In wxPerl this method takes as parameters the field widths.

See also

SetFieldsCount(), wxFrame::SetStatusWidths()

21.719 wxStatusBarPane Class Reference

```
#include <wx/statusbr.h>
```

21.719.1 Detailed Description

A status bar pane data container used by wxStatusBar.

Library: wxCore

Category: Data Structures

See also

wxStatusBar

Public Member Functions

- wxStatusBarPane (int style=wxSB_NORMAL, int width=0)

  Constructs the pane with the given style and width.
- int GetWidth () const

  Returns the pane width; it maybe negative, indicating a variable-width field.
- int GetStyle () const

  Returns the pane style.
- wxString GetText () const

  Returns the text currently shown in this pane.

21.719.2 Constructor & Destructor Documentation

wxStatusBarPane::wxStatusBarPane (int *style* = wxSB_NORMAL, int *width* = 0)

Constructs the pane with the given *style* and *width*.

21.719.3 Member Function Documentation

int wxStatusBarPane::GetStyle ( ) const

Returns the pane style.

wxString wxStatusBarPane::GetText ( ) const

Returns the text currently shown in this pane.

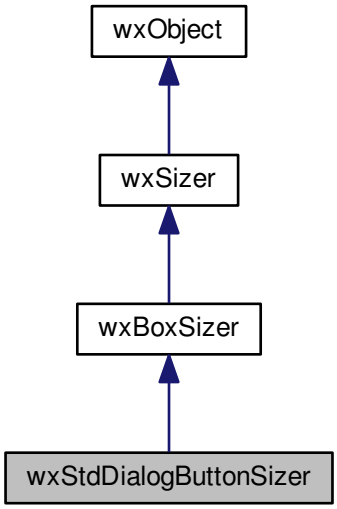
int wxStatusBarPane::GetWidth ( ) const

Returns the pane width; it maybe negative, indicating a variable-width field.

21.720 wxStdDialogButtonSizer Class Reference

`#include <wx/sizer.h>`

Inheritance diagram for wxStdDialogButtonSizer:



21.720.1 Detailed Description

This class creates button layouts which conform to the standard button spacing and ordering defined by the platform or toolkit's user interface guidelines (if such things exist).

By using this class, you can ensure that all your standard dialogs look correct on all major platforms. Currently it conforms to the Windows, GTK+ and Mac OS X human interface guidelines.

When there aren't interface guidelines defined for a particular platform or toolkit, wxStdDialogButtonSizer reverts to the Windows implementation.

To use this class, first add buttons to the sizer by calling wxStdDialogButtonSizer::AddButton (or wxStdDialogButtonSizer::SetAffirmativeButton, wxStdDialogButtonSizer::SetNegativeButton or wxStdDialogButtonSizer::SetCancelButton) and then call Realize in order to create the actual button layout used. Other than these special operations, this sizer works like any other sizer.

If you add a button with wxID_SAVE, on Mac OS X the button will be renamed to "Save" and the wxID_NO button will be renamed to "Don't Save" in accordance with the Mac OS X Human Interface Guidelines.

Library: wxCore

Category: Window Layout

See also

wxSizer, Sizers Overview, wxDialog::CreateButtonSizer

Public Member Functions

- wxStdDialogButtonSizer ()

  Constructor for a wxStdDialogButtonSizer.
- void AddButton (wxButton ∗button)

  Adds a button to the wxStdDialogButtonSizer.
- void Realize ()

  Rearranges the buttons and applies proper spacing between buttons to make them match the platform or toolkit's interface guidelines.
- void SetAffirmativeButton (wxButton ∗button)

  Sets the affirmative button for the sizer.
- void SetCancelButton (wxButton ∗button)

  Sets the cancel button for the sizer.
- void SetNegativeButton (wxButton ∗button)

  Sets the negative button for the sizer.
- virtual void RecalcSizes ()

  Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).
- virtual wxSize CalcMin ()

  Implements the calculation of a box sizer's minimal.

Additional Inherited Members

21.720.2 Constructor & Destructor Documentation

wxStdDialogButtonSizer::wxStdDialogButtonSizer ( )

Constructor for a wxStdDialogButtonSizer.

21.720.3 Member Function Documentation

void wxStdDialogButtonSizer::AddButton (wxButton ∗ *button*)

Adds a button to the wxStdDialogButtonSizer.

The *button* must have one of the following identifiers:

- wxID_OK
- wxID_YES
- wxID_SAVE
- wxID_APPLY
- wxID_CLOSE
- wxID_NO
- wxID_CANCEL
- wxID_HELP
- wxID_CONTEXT_HELP

virtual wxSize wxStdDialogButtonSizer::CalcMin ( ) [virtual]

Implements the calculation of a box sizer's minimal.

It is used internally only and must not be called by the user. Documented for information.

Reimplemented from wxBoxSizer.

void wxStdDialogButtonSizer::Realize ( )

Rearranges the buttons and applies proper spacing between buttons to make them match the platform or toolkit's interface guidelines.

virtual void wxStdDialogButtonSizer::RecalcSizes ( ) [virtual]

Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

It is used internally only and must not be called by the user (call Layout() if you want to resize). Documented for information.

Reimplemented from wxBoxSizer.

void wxStdDialogButtonSizer::SetAffirmativeButton (wxButton * *button*)

Sets the affirmative button for the sizer.

This allows you to use identifiers other than the standard identifiers outlined above.

void wxStdDialogButtonSizer::SetCancelButton (wxButton * *button*)

Sets the cancel button for the sizer.

This allows you to use identifiers other than the standard identifiers outlined above.

void wxStdDialogButtonSizer::SetNegativeButton (wxButton * *button*)

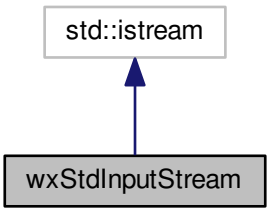
Sets the negative button for the sizer.

This allows you to use identifiers other than the standard identifiers outlined above.

21.721 wxStdInputStream Class Reference

```
#include <wx/stdstream.h>
```

Inheritance diagram for wxStdInputStream:



21.721.1 Detailed Description

wxStdInputStream is a std::istream derived stream which reads from a wxInputStream.

Example:

```
wxFFileInputStream file("words.txt");
wxStdInputStream in(file);
std::vector<std::string> words;

// read words from words.txt
std::copy(std::istream_iterator<std::string>(in),
          std::istream_iterator<std::string>(),
          std::back_inserter(words));

// sort and remove duplicates
std::sort(words.begin(), words.end());
words.resize(std::unique(words.begin(), words.end()) - words.begin());

// print words
std::copy(words.begin(), words.end(),
          std::ostream_iterator<std::string>(std::cout, "\n"));
```

Library: wxBase

Category: Streams

See also

wxInputStream, wxStdInputStreamBuffer

Public Member Functions

- wxStdInputStream (wxInputStream &stream)

  Creates a std::istream derived stream which reads from a wxInputStream.
- virtual ~wxStdInputStream ()

  Destructor.

21.721.2 Constructor & Destructor Documentation

wxStdInputStream::wxStdInputStream (wxInputStream & *stream*)

Creates a std::istream derived stream which reads from a wxInputStream.

Parameters

stream	Stream to read from.

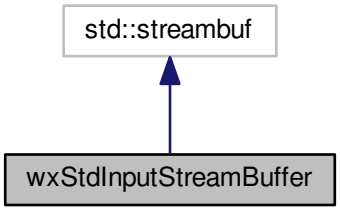
virtual wxStdInputStream::~wxStdInputStream ( ) [inline], [virtual]

Destructor.

21.722 wxStdInputStreamBuffer Class Reference

`#include <wx/stdstream.h>`

Inheritance diagram for wxStdInputStreamBuffer:



21.722.1 Detailed Description

wxStdInputStreamBuffer is a std::streambuf derived stream buffer which reads from a wxInputStream.

Example:

```
wxFFileInputStream file("input.txt.gz");
wxZlibInputStream gzipInput(file, wxZLIB_GZIP);
wxStdInputStreamBuffer gzipStreamBuffer(gzipInput);

// redirect std::cin to read from compressed file
std::streambuf* streamBufferOld = std::cin.rdbuf(&gzipStreamBuffer);

// prompt for integer
int number;
std::cout << "Enter an integer: " << std::flush;
std::cin >> number;
std::cout << std::endl;
std::cout << "You entered the integer " << number << "." << std::endl;

// restore std::cin
std::cin.rdbuf(streamBufferOld);
```

Library: wxBase

Category: Streams

See also

wxInputStream, wxStdInputStream

Public Member Functions

- wxStdInputStreamBuffer (wxInputStream &stream)

  Creates a std::steambuf derived stream buffer which reads from a wxInputStream.
- virtual ~wxStdInputStreamBuffer ()

  Destructor.

21.722.2 Constructor & Destructor Documentation

wxStdInputStreamBuffer::wxStdInputStreamBuffer (wxInputStream & *stream*)

Creates a std::steambuf derived stream buffer which reads from a wxInputStream.

Parameters

stream	Stream to read from.

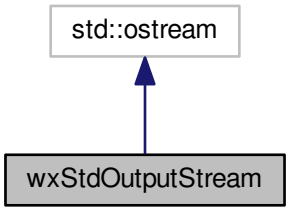
virtual wxStdInputStreamBuffer::~wxStdInputStreamBuffer ( ) [inline], [virtual]

Destructor.

21.723 wxStdOutputStream Class Reference

`#include <wx/stdstream.h>`

Inheritance diagram for wxStdOutputStream:



21.723.1 Detailed Description

wxStdOutputStream is a std::ostream derived stream which writes to a wxOutputStream.

Example:

```
wxFFileOutputStream file("out.txt.gz");
wxZlibOutputStream gzipOutput(file, -1, wxZLIB_GZIP);
wxStdOutputStream out(gzipOutput);

out << "Hello world!" << std::endl;
```

Library: wxBase

Category: Streams

See also

wxOutputStream, wxStdOutputStreamBuffer

Public Member Functions

- wxStdOutputStream (wxOutputStream &stream)

  Creates a std::ostream derived stream which writes to a wxOutputStream.
- virtual ~wxStdOutputStream ()

  Destructor.

21.723.2 Constructor & Destructor Documentation

wxStdOutputStream::wxStdOutputStream (wxOutputStream & *stream*)

Creates a std::ostream derived stream which writes to a wxOutputStream.

Parameters

stream	Stream to write to.

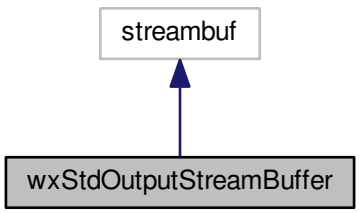
virtual wxStdOutputStream::~wxStdOutputStream ( ) [inline], [virtual]

Destructor.

21.724 wxStdOutputStreamBuffer Class Reference

`#include <wx/stdstream.h>`

Inheritance diagram for wxStdOutputStreamBuffer:



21.724.1 Detailed Description

wxStdOutputStreamBuffer is a std::streambuf derived stream buffer which writes to a wxOutputStream.

Example:

```
wxFFileOutputStream file("cout.txt.gz");
wxZlibOutputStream gzipOutput(file, -1, wxZLIB_GZIP);
wxStdOutputStreamBuffer gzipStreamBuffer(gzipOutput);

// redirect std::cout to cout.txt.gz using GZIP compression
std::streambuf* streamBufferOld = std::cout.rdbuf(&gzipStreamBuffer);

// write to std::cout
std::cout << "Hello world!" << std::endl;

// restore std::cout
std::cout.rdbuf(streamBufferOld);
```

Library: wxBase

Category: Streams

See also

wxOutputStream, wxStdOutputStream

Public Member Functions

- wxStdOutputStreamBuffer (wxOutputStream &stream)

  Creates a std::steambuf derived stream buffer which writes to a wxOutputStream.
- virtual ~wxStdOutputStreamBuffer ()

  Destructor.

21.724.2 Constructor & Destructor Documentation

wxStdOutputStreamBuffer::wxStdOutputStreamBuffer (wxOutputStream & *stream*)

Creates a std::steambuf derived stream buffer which writes to a wxOutputStream.

Parameters

stream	Stream to write to.

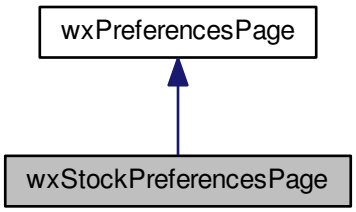
virtual wxStdOutputStreamBuffer::~wxStdOutputStreamBuffer ( ) [inline], [virtual]

Destructor.

21.725 wxStockPreferencesPage Class Reference

```
#include <wx/preferences.h>
```

Inheritance diagram for wxStockPreferencesPage:



21.725.1 Detailed Description

Specialization of wxPreferencesPage useful for certain commonly used preferences page.

On OS X, preferences pages named "General" and "Advanced" are commonly used in apps and the OS provides stock icons for them that should be used. Instead of reimplementing this behavior yourself, you can inherit from wxStockPreferencesPage and get correct title and icon.

Notice that this class only implements GetName() and GetLargeIcon(), you still have to provide the rest of wxPreferencesPage implementation.

Library: wxCore

Since

2.9.5

Public Types

- enum Kind {
  Kind_General,
  Kind_Advanced }

  Kinds of stock pages.

Public Member Functions

- wxStockPreferencesPage (Kind kind)

  Constructor.
- Kind GetKind () const

  Returns the page's kind.
- virtual wxString GetName () const

  Reimplemented to return suitable name for the page's kind.
- virtual wxBitmap GetLargeIcon () const

  Reimplemented to return stock icon on OS X.

21.725.2 Member Enumeration Documentation

enum wxStockPreferencesPage::Kind

Kinds of stock pages.

Enumerator

Kind_General The "General" page.

Kind_Advanced The "Advanced" page.

21.725.3 Constructor & Destructor Documentation

wxStockPreferencesPage::wxStockPreferencesPage (Kind *kind*)

Constructor.

21.725.4 Member Function Documentation

Kind wxStockPreferencesPage::GetKind ( ) const

Returns the page's kind.

virtual wxBitmap wxStockPreferencesPage::GetLargeIcon ( ) const `[virtual]`

Reimplemented to return stock icon on OS X.

Implements wxPreferencesPage.

virtual wxString wxStockPreferencesPage::GetName ( ) const `[virtual]`

Reimplemented to return suitable name for the page's kind.

Implements wxPreferencesPage.

21.726 wxStopWatch Class Reference

`#include <wx/stopwatch.h>`

21.726.1 Detailed Description

The wxStopWatch class allow you to measure time intervals.

For example, you may use it to measure the time elapsed by some function:

```
wxStopWatch sw;
CallLongRunningFunction();
wxLogMessage("The long running function took %ldms to execute",
             sw.Time());
sw.Pause();
... stopwatch is stopped now ...
sw.Resume();
CallLongRunningFunction();
wxLogMessage("And calling it twice took %ldms in all", sw.Time());
```

Since wxWidgets 2.9.3 this class uses `QueryPerformanceCounter()` function under MSW to measure the elapsed time. It provides higher precision than the usual timer functions but can suffer from bugs in its implementation in some Windows XP versions. If you encounter such problems, installing a Microsoft hot fix from `http://support.microsoft.com/?id=896256` could be necessary.

Library: wxBase

Category: Miscellaneous

See also

wxTimer

Public Member Functions

- wxStopWatch ()

  Constructor.
- void Pause ()

  Pauses the stop watch.
- void Resume ()

  Resumes the stop watch which had been paused with Pause().
- void Start (long milliseconds=0)

  (Re)starts the stop watch with a given initial value.
- long Time () const

  Returns the time in milliseconds since the start (or restart) or the last call of Pause().
- wxLongLong TimeInMicro () const

  Returns elapsed time in microseconds.

21.726.2 Constructor & Destructor Documentation

wxStopWatch::wxStopWatch ( )

Constructor.

This starts the stop watch.

21.726.3 Member Function Documentation

void wxStopWatch::Pause ( )

Pauses the stop watch.

Call Resume() to resume time measuring again.

If this method is called several times, `Resume()` must be called the same number of times to really resume the stop watch. You may, however, call Start() to resume it unconditionally.

void wxStopWatch::Resume ( )

Resumes the stop watch which had been paused with Pause().

void wxStopWatch::Start (long *milliseconds* = 0)

(Re)starts the stop watch with a given initial value.

The stopwatch will always be running after calling Start(), even if Pause() had been called before and even if it had been called multiple times.

long wxStopWatch::Time ( ) const

Returns the time in milliseconds since the start (or restart) or the last call of Pause().

See also

TimeInMicro()

wxLongLong wxStopWatch::TimeInMicro ( ) const

Returns elapsed time in microseconds.

This method is similar to Time() but returns the elapsed time in microseconds and not milliseconds. Notice that not all platforms really can measure times with this precision.

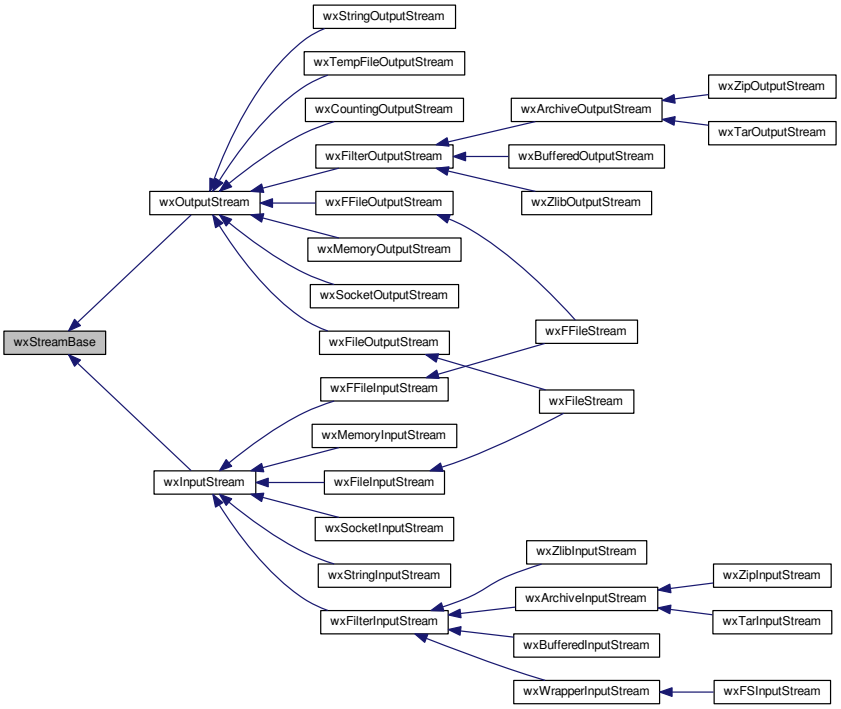
Since

2.9.3

21.727 wxStreamBase Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxStreamBase:



21.727.1 Detailed Description

This class is the base class of most stream related classes in wxWidgets.

It must not be used directly.

Library: wxBase

Category: Streams

See also

wxStreamBuffer

Public Member Functions

- wxStreamBase ()

  Creates a dummy stream object.
- virtual ~wxStreamBase ()

  Destructor.
- wxStreamError GetLastError () const

  This function returns the last error.
- virtual wxFileOffset GetLength () const

  Returns the length of the stream in bytes.
- virtual size_t GetSize () const

  This function returns the size of the stream.
- virtual bool IsOk () const

  Returns true if no error occurred on the stream.
- virtual bool IsSeekable () const

  Returns true if the stream supports seeking to arbitrary offsets.
- void Reset (wxStreamError error=wxSTREAM_NO_ERROR)

  Resets the stream state.
- bool operator! () const

  Returns the opposite of IsOk().

Protected Member Functions

- virtual wxFileOffset OnSysSeek (wxFileOffset pos, wxSeekMode mode)

  Internal function.
- virtual wxFileOffset OnSysTell () const

  Internal function.

21.727.2 Constructor & Destructor Documentation

wxStreamBase::wxStreamBase ( )

Creates a dummy stream object.

It doesn't do anything.

virtual wxStreamBase::~wxStreamBase ( ) [virtual]

Destructor.

21.727.3 Member Function Documentation

wxStreamError wxStreamBase::GetLastError ( ) const

This function returns the last error.

virtual wxFileOffset wxStreamBase::GetLength ( ) const [virtual]

Returns the length of the stream in bytes.

If the length cannot be determined (this is always the case for socket streams for example), returns wxInvalidOffset.

Since

2.5.4

Reimplemented in wxCountingOutputStream.

virtual size_t wxStreamBase::GetSize ( ) const [virtual]

This function returns the size of the stream.

For example, for a file it is the size of the file.

Warning

There are streams which do not have size by definition, such as socket streams. In that cases, GetSize() returns 0 so you should always test its return value.

virtual bool wxStreamBase::IsOk ( ) const [virtual]

Returns true if no error occurred on the stream.

See also

GetLastError()

Reimplemented in wxFSInputStream, wxFileStream, wxFFileStream, wxFFileInputStream, wxFileInputStream, wxFileOutputStream, and wxFFileOutputStream.

virtual bool wxStreamBase::IsSeekable ( ) const [virtual]

Returns true if the stream supports seeking to arbitrary offsets.

virtual wxFileOffset wxStreamBase::OnSysSeek (wxFileOffset *pos*, wxSeekMode *mode*) [protected], [virtual]

Internal function.

It is called when the stream needs to change the current position.

Parameters

pos	Offset to seek to.
mode	One of the wxSeekMode enumeration values.

Returns

The new stream position or wxInvalidOffset on error.

virtual wxFileOffset wxStreamBase::OnSysTell ( ) const `[protected], [virtual]`

Internal function.

It is called when the stream needs to know the real position.

Returns

The current stream position.

bool wxStreamBase::operator! ( ) const

Returns the opposite of IsOk().

You can use this function to test the validity of the stream as if it was a pointer:

```
bool DoSomething(wxInputStream& stream)
{
    wxInt32 data;
    if (!stream.Read(&data, 4))
        return false;
    ...
}
```

void wxStreamBase::Reset (wxStreamError *error* = wxSTREAM_NO_ERROR)

Resets the stream state.

By default, resets the stream to good state, i.e. clears any errors. Since wxWidgets 2.9.3 can be also used to explicitly set the state to the specified error (the *error* argument didn't exist in the previous versions).

See also

GetLastError()

21.728 wxStreamBuffer Class Reference

`#include <wx/stream.h>`

21.728.1 Detailed Description

wxStreamBuffer is a cache manager for wxStreamBase: it manages a stream buffer linked to a stream.

Each stream always has one autoinitialized stream buffer, but you may attach more of them to the same stream.

Library: wxBase

Category: Streams

See also

wxStreamBase, Stream Classes Overview

Public Types

- enum BufMode {
  read,
  write,
  read_write }

  BufMode flags.

Public Member Functions

- wxStreamBuffer (wxStreamBase &stream, BufMode mode)

  Constructor, creates a new stream buffer using stream as a parent stream and mode as the IO mode.
- wxStreamBuffer (size_t bufsize, wxInputStream &stream)

  Constructor for an input buffer of the specified size.
- wxStreamBuffer (size_t bufsize, wxOutputStream &stream)

  Constructor for an output buffer of the specified size.
- wxStreamBuffer (BufMode mode)

  Constructor; creates a new empty stream buffer which won't flush any data to a stream.
- wxStreamBuffer (const wxStreamBuffer &buffer)

  Copy constructor.
- ~wxStreamBuffer ()

  Destructor.
- bool FillBuffer ()

  Fill the IO buffer.
- void Fixed (bool fixed)

  Toggles the fixed flag.
- bool FlushBuffer ()

  Flushes the IO buffer.
- void Flushable (bool flushable)

  Toggles the flushable flag.
- void * GetBufferEnd () const

  Returns a pointer on the end of the stream buffer.
- void * GetBufferPos () const

  Returns a pointer on the current position of the stream buffer.
- size_t GetBufferSize () const

  Returns the size of the buffer.
- void * GetBufferStart () const

  Returns a pointer on the start of the stream buffer.
- virtual char GetChar ()

  Gets a single char from the stream buffer.
- size_t GetDataLeft ()

  Returns the amount of available data in the buffer.

- size_t GetIntPosition () const

  Returns the current position (counted in bytes) in the stream buffer.
- size_t GetLastAccess () const

  Returns the amount of bytes read during the last IO call to the parent stream.
- virtual void PutChar (char c)

  Puts a single char to the stream buffer.
- virtual size_t Read (void ∗buffer, size_t size)

  Reads a block of the specified size and stores the data in buffer.
- size_t Read (wxStreamBuffer ∗buffer)

  Copies data to buffer.
- void ResetBuffer ()

  Resets to the initial state variables concerning the buffer.
- virtual wxFileOffset Seek (wxFileOffset pos, wxSeekMode mode)

  Changes the current position.
- void SetBufferIO (void ∗start, void ∗end, bool takeOwnership=false)

  Specifies which pointers to use for stream buffering.
- void SetBufferIO (size_t bufsize)

  Destroys or invalidates the previous IO buffer and allocates a new one of the specified size.
- void SetIntPosition (size_t pos)

  Sets the current position (in bytes) in the stream buffer.
- wxStreamBase ∗ Stream ()

  Returns the parent stream of the stream buffer.
- virtual wxFileOffset Tell () const

  Gets the current position in the stream.
- void Truncate ()

  Truncates the buffer to the current position.
- virtual size_t Write (const void ∗buffer, size_t size)

  Writes a block of the specified size using data of buffer.
- size_t Write (wxStreamBuffer ∗buffer)

  See Read().

21.728.2 Member Enumeration Documentation

enum wxStreamBuffer::BufMode

BufMode flags.

Enumerator

read

write

read_write

21.728.3 Constructor & Destructor Documentation

wxStreamBuffer::wxStreamBuffer (wxStreamBase & *stream*, BufMode *mode*)

Constructor, creates a new stream buffer using *stream* as a parent stream and mode as the IO mode.

Parameters

stream	The parent stream.
mode	Can be: wxStreamBuffer::read, wxStreamBuffer::write, wxStreamBuffer::read_write.

One stream can have many stream buffers but only one is used internally to pass IO call (e.g. wxInputStream::Read() -> wxStreamBuffer::Read()), but you can call directly wxStreamBuffer::Read without any problems. Note that all errors and messages linked to the stream are stored in the stream, not the stream buffers:

```
streambuffer.Read(...);
streambuffer2.Read(...);
    // This call erases previous error messages set by 'streambuffer'
    // assuming that both instances are stream buffers for the same stream
```

See also

SetBufferIO()

wxStreamBuffer::wxStreamBuffer (size_t *bufsize,* **wxInputStream** & *stream*)

Constructor for an input buffer of the specified size.

Using it is equivalent to using the constructor above with read mode and calling SetBufferIO() but is more convenient.

Since

2.9.0

Parameters

bufsize	The size of buffer in bytes.
stream	The associated input stream, the buffer will be used in read mode.

wxStreamBuffer::wxStreamBuffer (size_t *bufsize,* **wxOutputStream** & *stream*)

Constructor for an output buffer of the specified size.

Using it is equivalent to using the constructor above with write mode and calling SetBufferIO() but is more convenient.

Since

2.9.0

Parameters

bufsize	The size of buffer in bytes.
stream	The associated output stream, the buffer will be used in write mode.

wxStreamBuffer::wxStreamBuffer (**BufMode** *mode*)

Constructor; creates a new empty stream buffer which won't flush any data to a stream.

mode specifies the type of the buffer (read, write, read_write).

This stream buffer has the advantage to be stream independent and to work only on memory buffers but it is still compatible with the rest of the wxStream classes. You can write, read to this special stream and it will grow (if it is allowed by the user) its internal buffer. Briefly, it has all functionality of a "normal" stream.

Warning

The "read_write" mode doesn't currently work for standalone stream buffers.

See also

SetBufferIO()

wxStreamBuffer::wxStreamBuffer (const wxStreamBuffer & *buffer*)

Copy constructor.

This method initializes the stream buffer with the data of the specified stream buffer. The new stream buffer has the same attributes, size, position and they share the same buffer. This will cause problems if the stream to which the stream buffer belong is destroyed and the newly cloned stream buffer continues to be used, trying to call functions in the (destroyed) stream. It is advised to use this feature only in very local area of the program.

wxStreamBuffer::~wxStreamBuffer ( )

Destructor.

It finalizes all IO calls and frees all internal buffers if necessary.

21.728.4 Member Function Documentation

bool wxStreamBuffer::FillBuffer ( )

Fill the IO buffer.

void wxStreamBuffer::Fixed (bool *fixed*)

Toggles the fixed flag.

Usually this flag is toggled at the same time as *flushable*. This flag allows (when it has the false value) or forbids (when it has the true value) the stream buffer to resize dynamically the IO buffer.

See also

SetBufferIO()

void wxStreamBuffer::Flushable (bool *flushable*)

Toggles the flushable flag.

If *flushable* is disabled, no data are sent to the parent stream.

bool wxStreamBuffer::FlushBuffer ( )

Flushes the IO buffer.

void* wxStreamBuffer::GetBufferEnd ( ) const

Returns a pointer on the end of the stream buffer.

void* wxStreamBuffer::GetBufferPos ( ) const

Returns a pointer on the current position of the stream buffer.

size_t wxStreamBuffer::GetBufferSize ( ) const

Returns the size of the buffer.

void* wxStreamBuffer::GetBufferStart ( ) const

Returns a pointer on the start of the stream buffer.

virtual char wxStreamBuffer::GetChar ( ) [virtual]

Gets a single char from the stream buffer.

It acts like the Read() call.

Warning

You aren't directly notified if an error occurred during the IO call.

See also

Read()

size_t wxStreamBuffer::GetDataLeft ( )

Returns the amount of available data in the buffer.

size_t wxStreamBuffer::GetIntPosition ( ) const

Returns the current position (counted in bytes) in the stream buffer.

size_t wxStreamBuffer::GetLastAccess ( ) const

Returns the amount of bytes read during the last IO call to the parent stream.

virtual void wxStreamBuffer::PutChar (char *c*) [virtual]

Puts a single char to the stream buffer.

Warning

You aren't directly notified if an error occurred during the IO call.

See also

Read()

virtual size_t wxStreamBuffer::Read (void * *buffer,* size_t *size*) [virtual]

Reads a block of the specified size and stores the data in buffer.

This function tries to read from the buffer first and if more data has been requested, reads more data from the associated stream and updates the buffer accordingly until all requested data is read.

Returns

It returns the size of the data read. If the returned size is different of the specified size, an error has occurred and should be tested using GetLastError().

size_t wxStreamBuffer::Read (wxStreamBuffer * *buffer*)

Copies data to *buffer*.

The function returns when *buffer* is full or when there isn't any more data in the current buffer.

See also

Write()

void wxStreamBuffer::ResetBuffer ( )

Resets to the initial state variables concerning the buffer.

virtual wxFileOffset wxStreamBuffer::Seek (wxFileOffset *pos,* wxSeekMode *mode*) [virtual]

Changes the current position.

Parameter *mode* may be one of the following:

- **wxFromStart:** The position is counted from the start of the stream.
- **wxFromCurrent:** The position is counted from the current position of the stream.
- **wxFromEnd:** The position is counted from the end of the stream.

Returns

Upon successful completion, it returns the new offset as measured in bytes from the beginning of the stream. Otherwise, it returns wxInvalidOffset.

void wxStreamBuffer::SetBufferIO (void * *start,* void * *end,* bool *takeOwnership* = false)

Specifies which pointers to use for stream buffering.

You need to pass a pointer on the start of the buffer end and another on the end. The object will use this buffer to cache stream data. It may be used also as a source/destination buffer when you create an empty stream buffer (See wxStreamBuffer::wxStreamBuffer).

Remarks

When you use this function, you will have to destroy the IO buffers yourself after the stream buffer is destroyed or don't use it anymore. In the case you use it with an empty buffer, the stream buffer will not resize it when it is full.

See also

wxStreamBuffer(), Fixed(), Flushable()

void wxStreamBuffer::SetBufferIO (size_t *bufsize*)

Destroys or invalidates the previous IO buffer and allocates a new one of the specified size.

Warning

All previous pointers aren't valid anymore.

Remarks

The created IO buffer is growable by the object.

See also

Fixed(), Flushable()

void wxStreamBuffer::SetIntPosition (size_t *pos*)

Sets the current position (in bytes) in the stream buffer.

Warning

Since it is a very low-level function, there is no check on the position: specifying an invalid position can induce unexpected results.

wxStreamBase* wxStreamBuffer::Stream ( )

Returns the parent stream of the stream buffer.

Deprecated use GetStream() instead

virtual wxFileOffset wxStreamBuffer::Tell ( ) const `[virtual]`

Gets the current position in the stream.

This position is calculated from the *real* position in the stream and from the internal buffer position: so it gives you the position in the *real* stream counted from the start of the stream.

Returns

Returns the current position in the stream if possible, wxInvalidOffset in the other case.

void wxStreamBuffer::Truncate ( )

Truncates the buffer to the current position.

Note

Truncate() cannot be used to enlarge the buffer. This is usually not needed since the buffer expands automatically.

virtual size_t wxStreamBuffer::Write (const void * *buffer*, size_t *size*) `[virtual]`

Writes a block of the specified size using data of buffer.

The data are cached in a buffer before being sent in one block to the stream.

size_t wxStreamBuffer::Write (wxStreamBuffer * *buffer*)

See Read().

21.729 wxStreamToTextRedirector Class Reference

`#include <wx/textctrl.h>`

21.729.1 Detailed Description

This class can be used to (temporarily) redirect all output sent to a C++ ostream object to a wxTextCtrl instead.

Note

Some compilers and/or build configurations don't support multiply inheriting wxTextCtrl from `std::streambuf` in which case this class is not compiled in. You also must have `wxUSE_STD_IOSTREAM` option on (i.e. set to 1) in your `setup.h` to be able to use it. Under Unix, specify `-enable-std_iostreams` switch when running configure for this.

Example of usage:

```
using namespace std;
wxTextCtrl* text = new wxTextCtrl(...);
{
    wxStreamToTextRedirector redirect(text);

    // this goes to the text control
    cout << "Hello, text!" << endl;
}
// this goes somewhere else, presumably to stdout
cout << "Hello, console!" << endl;
```

Library: wxCore

Category: Logging

See also

wxTextCtrl

Public Member Functions

- wxStreamToTextRedirector (wxTextCtrl *text, ostream *ostr)

  The constructor starts redirecting output sent to ostr or cout for the default parameter value to the text control text.
- ~wxStreamToTextRedirector ()

  When a wxStreamToTextRedirector object is destroyed, the redirection is ended and any output sent to the C++ ostream which had been specified at the time of the object construction will go to its original destination.

21.729.2 Constructor & Destructor Documentation

wxStreamToTextRedirector::wxStreamToTextRedirector (wxTextCtrl * *text,* ostream * *ostr*)

The constructor starts redirecting output sent to *ostr* or *cout* for the default parameter value to the text control *text*.

Parameters

text	The text control to append output too, must be non-NULL
ostr	The C++ stream to redirect, cout is used if it is NULL

wxStreamToTextRedirector::~wxStreamToTextRedirector ( )

When a wxStreamToTextRedirector object is destroyed, the redirection is ended and any output sent to the C++ ostream which had been specified at the time of the object construction will go to its original destination.

21.730 wxString Class Reference

`#include <wx/string.h>`

21.730.1 Detailed Description

String class for passing textual data to or receiving it from wxWidgets.

Note

> While the use of wxString is unavoidable in wxWidgets program, you are encouraged to use the standard string classes `std::string` or `std::wstring` in your applications and convert them to and from wxString only when interacting with wxWidgets.

wxString is a class representing a Unicode character string but with methods taking or returning both `wchar_t` wide characters and `wchar_t*` wide strings and traditional `char` characters and `char*` strings. The dual nature of wxString API makes it simple to use in all cases and, importantly, allows the code written for either ANSI or Unicode builds of the previous wxWidgets versions to compile and work correctly with the single unified Unicode build of wxWidgets 3.0. It is also mostly transparent when using wxString with the few exceptions described below.

21.730.2 API overview

wxString tries to be similar to both `std::string` and `std::wstring` and can mostly be used as either class. It provides practically all of the methods of these classes, which behave exactly the same as in the standard C++, and so are not documented here (please see any standard library documentation, for example `http://en.cppreference.com/w/cpp/string` for more details).

In addition to these standard methods, wxString adds functions dealing with the conversions between different string encodings, described below, as well as many extra helpers such as functions for formatted output (Printf(), Format(), ...), case conversion (MakeUpper(), Capitalize(), ...) and various others (Trim(), StartsWith(), Matches(), ...). All of the non-standard methods follow wxWidgets "CamelCase" naming convention and are documented here.

Notice that some wxString methods exist in several versions for compatibility reasons. For example all of length(), Length() and Len() are provided. In such cases it is recommended to use the standard string-like method, i.e. length() in this case.

21.730.3 Converting to and from wxString

wxString can be created from:

- ASCII string guaranteed to contain only 7 bit characters using wxString::FromAscii().
- Narrow `char*` string in the current locale encoding using implicit wxString::wxString(const char*) constructor.

- Narrow `char*` string in UTF-8 encoding using wxString::FromUTF8().
- Narrow `char*` string in the given encoding using wxString::wxString(const char*, const wxMBConv&) constructor passing a wxCSConv corresponding to the encoding as the second argument.
- Standard `std::string` using implicit wxString::wxString(const std::string&) constructor. Notice that this constructor supposes that the string contains data in the current locale encoding, use FromUTF8() or the constructor taking wxMBConv if this is not the case.
- Wide `wchar_t*` string using implicit wxString::wxString(const wchar_t*) constructor.
- Standard `std::wstring` using implicit wxString::wxString(const std::wstring&) constructor.

Notice that many of the constructors are implicit, meaning that you don't even need to write them at all to pass the existing string to some wxWidgets function taking a wxString.

Similarly, wxString can be converted to:

- ASCII string using wxString::ToAscii(). This is a potentially destructive operation as all non-ASCII string characters are replaced with a placeholder character.
- String in the current locale encoding implicitly or using c_str() or mb_str() methods. This is a potentially destructive operation as an *empty* string is returned if the conversion fails.
- String in UTF-8 encoding using wxString::utf8_str().
- String in any given encoding using mb_str() with the appropriate wxMBConv object. This is also a potentially destructive operation.
- Standard `std::string` using wxString::ToStdString(). The contents of the returned string use the current locale encoding, so this conversion is potentially destructive as well.
- Wide C string using wxString::wc_str().
- Standard `std::wstring` using wxString::ToStdWstring().

Note

If you built wxWidgets with `wxUSE_STL` set to 1, the implicit conversions to both narrow and wide C strings are disabled and replaced with implicit conversions to `std::string` and `std::wstring`.

Please notice that the conversions marked as "potentially destructive" above can result in loss of data if their result is not checked, so you need to verify that converting the contents of a non-empty Unicode string to a non-UTF-8 multibyte encoding results in non-empty string. The simplest and best way to ensure that the conversion never fails is to always use UTF-8.

21.730.4 Traps for the unwary

As mentioned above, wxString tries to be compatible with both narrow and wide standard string classes and mostly does it transparently, but there are some exceptions.

String element access

Some problems are caused by wxString::operator[]() which returns an object of a special proxy class allowing to assign either a simple `char` or a `wchar_t` to the given index. Because of this, the return type of this operator is neither `char` nor `wchar_t` nor a reference to one of these types but wxUniCharRef which is not a primitive type and hence can't be used in the `switch` statement. So the following code does *not* compile

```
wxString s(...);
switch ( s[n] ) {
    case 'A':
        ...
        break;
}
```

and you need to use

```
switch ( s[n].GetValue() ) {
    ...
}
```

instead. Alternatively, you can use an explicit cast:

```
switch ( static_cast<char>(s[n]) ) {
    ...
}
```

but notice that this will result in an assert failure if the character at the given position is not representable as a single `char` in the current encoding, so you may want to cast to `int` instead if non-ASCII values can be used.

Another consequence of this unusual return type arises when it is used with template deduction or C++11 `auto` keyword. Unlike with the normal references which are deduced to be of the referenced type, the deduced type for wxUniCharRef is wxUniCharRef itself. This results in potentially unexpected behaviour, for example:

```
wxString s("abc");
auto c = s[0];
c = 'x';             // Modifies the string!
wxASSERT( s == "xbc" );
```

Due to this, either explicitly specify the variable type:

```
int c = s[0];
c = 'x';             // Doesn't modify the string any more.
wxASSERT( s == "abc" );
```

or explicitly convert the return value:

```
auto c = s[0].GetValue();
c = 'x';             // Doesn't modify the string neither.
wxASSERT( s == "abc" );
```

Conversion to C string

A different class of problems happens due to the dual nature of the return value of wxString::c_str() method, which is also used for implicit conversions. The result of calls to this method is convertible to either narrow `char*` string or wide `wchar_t*` string and so, again, has neither the former nor the latter type. Usually, the correct type will be chosen depending on how you use the result but sometimes the compiler can't choose it because of an ambiguity, e.g.:

```
// Some non-wxWidgets functions existing for both narrow and wide
// strings:
void dump_text(const char* text);       // Version (1)
void dump_text(const wchar_t* text);    // Version (2)

wxString s(...);
dump_text(s);           // ERROR: ambiguity.
dump_text(s.c_str());   // ERROR: still ambiguous.
```

In this case you need to explicitly convert to the type that you need to use or use a different, non-ambiguous, conversion function (which is usually the best choice):

```
dump_text(static_cast<const char*>(s));            // OK, calls (1)
dump_text(static_cast<const wchar_t*>(s.c_str())); // OK, calls (2)
dump_text(s.mb_str());                             // OK, calls (1)
dump_text(s.wc_str());                             // OK, calls (2)
dump_text(s.wx_str());                             // OK, calls ???
```

Using wxString with vararg functions

A special subclass of the problems arising due to the polymorphic nature of wxString::c_str() result type happens when using functions taking an arbitrary number of arguments, such as the standard `printf()`. Due to the rules of the C++ language, the types for the "variable" arguments of such functions are not specified and hence the compiler cannot convert wxString objects, or the objects returned by wxString::c_str(), to these unknown types automatically. Hence neither wxString objects nor the results of most of the conversion functions can be passed as vararg arguments:

```
// ALL EXAMPLES HERE DO NOT WORK, DO NOT USE THEM!
printf("Don't do this: %s", s);
printf("Don't do that: %s", s.c_str());
printf("Nor even this: %s", s.mb_str());
wprintf("And even not always this: %s", s.wc_str());
```

Instead you need to either explicitly cast to the needed type:

```
// These examples work but are not the best solution, see below.
printf("You can do this: %s", static_cast<const char*>(s));
printf("Or this: %s", static_cast<const char*>(s.c_str()));
printf("And this: %s", static_cast<const char*>(s.mb_str()));
wprintf("Or this: %s", static_cast<const wchar_t*>(s.wc_str()));
```

But a better solution is to use wxWidgets-provided functions, if possible, as is the case for `printf` family of functions:

```
// This is the recommended way.
wxPrintf("You can do just this: %s", s);
wxPrintf("And this (but it is redundant): %s", s.c_str());
wxPrintf("And this (not using Unicode): %s", s.mb_str());
wxPrintf("And this (always Unicode): %s", s.wc_str());
```

Notice that wxPrintf() replaces both `printf()` and `wprintf()` and accepts wxString objects, results of c_str() calls but also `char*` and `wchar_t*` strings directly.

wxWidgets provides wx-prefixed equivalents to all the standard vararg functions and a few more, notably wxString::Format(), wxLogMessage(), wxLogError() and other log functions. But if you can't use one of those functions and need to pass wxString objects to non-wx vararg functions, you need to use the explicit casts as explained above.

21.730.5 Performance characteristics

wxString uses `std::basic_string` internally to store its content (unless this is not supported by the compiler or disabled specifically when building wxWidgets) and it therefore inherits many features from `std::basic_string`. In particular, most modern implementations of `std::basic_string` are thread-safe and don't use reference counting (making copying large strings potentially expensive) and so wxString has the same characteristics.

By default, wxString uses `std::basic_string` specialized for the platform-dependent `wchar_t` type, meaning that it is not memory-efficient for ASCII strings, especially under Unix platforms where every ASCII character, normally fitting in a byte, is represented by a 4 byte `wchar_t`.

It is possible to build wxWidgets with `wxUSE_UNICODE_UTF8` set to 1 in which case an UTF-8-encoded string representation is stored in `std::basic_string` specialized for `char`, i.e. the usual `std::string`. In this case the memory efficiency problem mentioned above doesn't arise but run-time performance of many wxString methods changes dramatically, in particular accessing the N-th character of the string becomes an operation taking O(N) time instead of O(1), i.e. constant, time by default. Thus, if you do use this so called UTF-8 build, you should avoid using indices to access the strings whenever possible and use the iterators instead. As an example, traversing the string using iterators is an O(N), where N is the string length, operation in both the normal ("wchar_t") and UTF-8 builds but doing it using indices becomes O(N^2) in UTF-8 case meaning that simply checking every character of a reasonably long (e.g. a couple of millions elements) string can take an unreasonably long time.

However, if you do use iterators, UTF-8 build can be a better choice than the default build, especially for the memory-constrained embedded systems. Notice also that GTK+ and DirectFB use UTF-8 internally, so using this build not only saves memory for ASCII strings but also avoids conversions between wxWidgets and the underlying toolkit.

21.730.6 Index of the member groups

Links for quick access to the various categories of wxString functions:

Library: wxBase

Category: Data Structures

Predefined objects/pointers: wxEmptyString

See also

wxString Overview, Unicode Support in wxWidgets, String-related functions, wxUString, wxCharBuffer, wxUniChar, wxStringTokenizer, wxStringBuffer, wxStringBufferLength

Public Types

Standard types

Types used with wxString.

- typedef wxUniChar value_type
- typedef wxUniChar char_type
- typedef wxUniCharRef reference
- typedef wxChar * pointer
- typedef const wxChar * const_pointer
- typedef size_t size_type
- typedef wxUniChar const_reference

Public Member Functions

Constructors and assignment operators

A string may be constructed either from a C string, (some number of copies of) a single character or a wide (Unicode) string.

For all constructors (except the default which creates an empty string) there is also a corresponding assignment operator.

See also the assign() STL-like function.

- wxString ()
  Default constructor.
- wxString (const wxString &stringSrc)
  Creates a string from another string.
- wxString (wxUniChar ch, size_t nRepeat=1)
  Construct a string consisting of nRepeat copies of ch.
- wxString (wxUniCharRef ch, size_t nRepeat=1)
  Construct a string consisting of nRepeat copies of ch.
- wxString (char ch, size_t nRepeat=1)
  Construct a string consisting of nRepeat copies of ch converted to Unicode using the current locale encoding.
- wxString (wchar_t ch, size_t nRepeat=1)
  Construct a string consisting of nRepeat copies of ch.
- wxString (const char *psz)
  Constructs a string from the string literal psz using the current locale encoding to convert it to Unicode (wxConv↩ Libc).
- wxString (const char *psz, const wxMBConv &conv)
  Constructs a string from the string literal psz using conv to convert it Unicode.
- wxString (const char *psz, size_t nLength)
  Constructs a string from the first nLength character of the string literal psz using the current locale encoding to convert it to Unicode (wxConvLibc).
- wxString (const char *psz, const wxMBConv &conv, size_t nLength)
  Constructs a string from the first nLength character of the string literal psz using conv to convert it Unicode.
- wxString (const wchar_t *pwz)
  Constructs a string from the string literal pwz.
- wxString (const wchar_t *pwz, size_t nLength)
  Constructs a string from the first nLength characters of the string literal pwz.
- wxString (const wxCharBuffer &buf)
  Constructs a string from buf using the using the current locale encoding to convert it to Unicode.
- wxString (const wxWCharBuffer &buf)
  Constructs a string from buf.
- wxString (const std::string &str)
  Constructs a string from str using the using the current locale encoding to convert it to Unicode (wxConvLibc).
- wxString (const std::wstring &str)
  Constructs a string from str.
- ~wxString ()
  String destructor.
- wxString operator= (const wxString &str)
  Assignment: see the relative wxString constructor.
- wxString operator= (wxUniChar c)
  Assignment: see the relative wxString constructor.

String length

These functions return the string length and/or check whether the string is empty.

See also the length(), size() or empty() STL-like functions.

- size_t Len () const
  Returns the length of the string.
- size_t Length () const
  Returns the length of the string (same as Len).

- bool IsEmpty () const
  Returns true if the string is empty.
- bool IsNull () const
  Returns true if the string is empty (same as wxString::IsEmpty).
- bool operator! () const
  Empty string is false, so !string will only return true if the string is empty.

Character access

Many functions below take a character index in the string.

As with C strings and arrays, the indices start from 0, so the first character of a string is string[0]. An attempt to access a character beyond the end of the string (which may even be 0 if the string is empty) will provoke an assert failure in debug builds, but no checks are done in release builds.

- wxUniChar GetChar (size_t n) const
  Returns the character at position n (read-only).
- const wxCStrData GetData () const
  wxWidgets compatibility conversion.
- wxUniCharRef GetWritableChar (size_t n)
  Returns a reference to the character at position n.
- wxStringCharType * GetWriteBuf (size_t len)
  Returns a writable buffer of at least len bytes.
- void UngetWriteBuf ()
  Puts the string back into a reasonable state (in which it can be used normally), after GetWriteBuf() was called.
- void UngetWriteBuf (size_t len)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetChar (size_t n, wxUniChar ch)
  Sets the character at position n.
- wxUniChar Last () const
  Returns the last character.
- wxUniCharRef Last ()
  Returns a reference to the last character (writable).
- wxUniChar operator[] (size_t i) const
  Returns the i-th character of the string.
- wxUniCharRef operator[] (size_t i)
  Returns a writable reference to the i-th character of the string.

Conversions

This section contains both implicit and explicit conversions to C style strings.

Although implicit conversion is quite convenient, you are advised to use wc_str() for the sake of clarity.

- wxCStrData c_str () const
  Returns a lightweight intermediate class which is in turn implicitly convertible to both `const char*` and to `const wchar_t*`.
- wxWritableCharBuffer char_str (const wxMBConv &conv=wxConvLibc) const
  Returns an object with string data that is implicitly convertible to `char*` pointer.
- template<typename T >
  wxCharTypeBuffer< T > tchar_str (size_t *len=NULL) const
  Returns buffer of the specified type containing the string data.
- const wchar_t * fn_str () const
  Returns a string representation suitable for passing to OS' functions for file handling.
- const char * fn_str () const
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- const wxCharBuffer fn_str () const
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- const wxCharBuffer mb_str (const wxMBConv &conv=wxConvLibc) const

Returns the multibyte (C string) representation of the string using conv's wxMBConv::cWC2MB method and returns wxCharBuffer.

- const wxScopedCharBuffer utf8_str () const

  Converts the strings contents to UTF-8 and returns it either as a temporary wxCharBuffer object or as a pointer to the internal string contents in UTF-8 build.
- const wchar_t * wc_str () const

  Converts the strings contents to the wide character representation and returns it as a temporary wxWCharBuffer object (Unix and OS X) or returns a pointer to the internal string contents in wide character mode (Windows).
- const wxWCharBuffer wc_str () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxWritableWCharBuffer wchar_str () const

  Returns an object with string data that is implicitly convertible to `char*` pointer.
- const wxStringCharType * wx_str () const

  Explicit conversion to C string in the internal representation (either wchar_t or UTF-8-encoded char*, depending on the build).*
- const wxScopedCharBuffer To8BitData () const

  Converts the string to an 8-bit string in ISO-8859-1 encoding in the form of a wxCharBuffer (Unicode builds only).
- const char * ToAscii () const

  Converts the string to an ASCII, 7-bit string in the form of a wxCharBuffer (Unicode builds only) or a C string (ANSI builds).
- const wxCharBuffer ToAscii () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- std::string ToStdString () const

  Return the string as an std::string in current locale encoding.
- std::wstring ToStdWstring () const

  Return the string as an std::wstring.
- const wxScopedCharBuffer ToUTF8 () const

  Same as utf8_str().

Concatenation

Almost anything may be concatenated (appended to) with a string!

Note that the various operator<<() overloads work as C++ stream insertion operators. They insert the given value into the string. Precision and format cannot be set using them. Use Printf() instead.

See also the insert() and append() STL-like functions.

- wxString & Append (const char *psz)

  Appends the string literal psz.
- wxString & Append (const wchar_t *pwz)

  Appends the wide string literal pwz.
- wxString & Append (const char *psz, size_t nLen)

  Appends the string literal psz with max length nLen.
- wxString & Append (const wchar_t *pwz, size_t nLen)

  Appends the wide string literal psz with max length nLen.
- wxString & Append (const wxString &s)

  Appends the string s.
- wxString & Append (wxUniChar ch, size_t count=1u)

  Appends the character ch count times.
- wxString & Prepend (const wxString &str)

  Prepends str to this string, returning a reference to this string.
- wxString operator+ (const wxString &x, const wxString &y)

  Concatenation: returns a new string equal to the concatenation of the operands.
- wxString operator+ (const wxString &x, wxUniChar y)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxString & operator<< (const wxString &s)

  Appends the string literal psz.
- wxString & operator<< (const char *psz)

Appends the string literal psz.

- wxString & operator<< (const wchar_t *pwz)

  Appends the string literal psz.
- wxString & operator<< (const wxCStrData &psz)

  Appends the string literal psz.
- wxString & operator<< (char ch)

  Appends the string literal psz.
- wxString & operator<< (unsigned char ch)

  Appends the string literal psz.
- wxString & operator<< (wchar_t ch)

  Appends the string literal psz.
- wxString & operator<< (const wxCharBuffer &s)

  Appends the string literal psz.
- wxString & operator<< (const wxWCharBuffer &s)

  Appends the string literal psz.
- wxString & operator<< (wxUniChar ch)

  Appends the string literal psz.
- wxString & operator<< (wxUniCharRef ch)

  Appends the string literal psz.
- wxString & operator<< (unsigned int ui)

  Appends the string literal psz.
- wxString & operator<< (long l)

  Appends the string literal psz.
- wxString & operator<< (unsigned long ul)

  Appends the string literal psz.
- wxString & operator<< (wxLongLong_t ll)

  Appends the string literal psz.
- wxString & operator<< (wxULongLong_t ul)

  Appends the string literal psz.
- wxString & operator<< (float f)

  Appends the string literal psz.
- wxString & operator<< (double d)

  Appends the string literal psz.
- void operator+= (const wxString &str)

  Concatenation in place: the argument is appended to the string.
- void operator+= (wxUniChar c)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Comparison

The default comparison function Cmp() is case-sensitive and so is the default version of IsSameAs().

For case insensitive comparisons you should use CmpNoCase() or give a second parameter to IsSameAs(). This last function is maybe more convenient if only equality of the strings matters because it returns a boolean true value if the strings are the same and not 0 (which is usually false in C) as Cmp() does.

Matches() is a poor man's regular expression matcher: it only understands '' and '?' metacharacters in the sense of DOS command line interpreter.*

StartsWith() is helpful when parsing a line of text which should start with some predefined prefix and is more efficient than doing direct string comparison as you would also have to precalculate the length of the prefix.

See also the compare() STL-like function.

- int Cmp (const wxString &s) const

  Case-sensitive comparison.
- int CmpNoCase (const wxString &s) const

  Case-insensitive comparison.
- bool IsSameAs (const wxString &s, bool caseSensitive=true) const

  Test whether the string is equal to another string s.
- bool IsSameAs (wxUniChar ch, bool caseSensitive=true) const

Test whether the string is equal to the single character ch.

- bool Matches (const wxString &mask) const

  Returns true if the string contents matches a mask containing '' and '?'.*
- bool StartsWith (const wxString &prefix, wxString *rest=NULL) const

  This function can be used to test if the string starts with the specified prefix.
- bool EndsWith (const wxString &suffix, wxString *rest=NULL) const

  This function can be used to test if the string ends with the specified suffix.

Substring extraction

These functions allow you to extract a substring from the string.

The original string is not modified and the function returns the extracted substring.

See also the at() and the substr() STL-like functions.

- wxString Mid (size_t first, size_t nCount=wxString::npos) const

  Returns a substring starting at first, with length count, or the rest of the string if count is the default value.
- wxString SubString (size_t from, size_t to) const

  Returns the part of the string between the indices from and to inclusive.
- wxString operator() (size_t start, size_t len) const

  Same as Mid() (substring extraction).
- wxString Left (size_t count) const

  Returns the first count characters of the string.
- wxString Right (size_t count) const

  Returns the last count characters.
- wxString AfterFirst (wxUniChar ch) const

  Gets all the characters after the first occurrence of ch.
- wxString AfterLast (wxUniChar ch) const

  Gets all the characters after the last occurrence of ch.
- wxString BeforeFirst (wxUniChar ch, wxString *rest=NULL) const

  Gets all characters before the first occurrence of ch.
- wxString BeforeLast (wxUniChar ch, wxString *rest=NULL) const

  Gets all characters before the last occurrence of ch.

Case conversion

The MakeXXX() variants modify the string in place, while the other functions return a new string which contains the original text converted to the upper or lower case and leave the original string unchanged.

- wxString Capitalize () const

  Return the copy of the string with the first string character in the upper case and the subsequent ones in the lower case.
- wxString Lower () const

  Returns this string converted to the lower case.
- void LowerCase ()

  Same as MakeLower.
- wxString & MakeCapitalized ()

  Converts the first characters of the string to the upper case and all the subsequent ones to the lower case and returns the result.
- wxString & MakeLower ()

  Converts all characters to lower case and returns the reference to the modified string.
- wxString & MakeUpper ()

  Converts all characters to upper case and returns the reference to the modified string.
- wxString Upper () const

  Returns this string converted to upper case.
- void UpperCase ()

  The same as MakeUpper().

Searching and replacing

These functions replace the standard `strchr()` and `strstr()` functions.

See also the find(), rfind(), replace() STL-like functions.

- int Find (wxUniChar ch, bool fromEnd=false) const
  Searches for the given character ch.
- int Find (const wxString &sub) const
  Searches for the given string sub.
- int First (wxUniChar ch) const
  Same as Find().
- int First (const wxString &str) const
  Same as Find().
- size_t Replace (const wxString &strOld, const wxString &strNew, bool replaceAll=true)
  Replace first (or all) occurrences of substring with another one.

Conversion to numbers

The string provides functions for conversion to signed and unsigned integer and floating point numbers.

*All functions take a pointer to the variable to put the numeric value in and return true if the **entire** string could be converted to a number. Notice if there is a valid number in the beginning of the string, it is returned in the output parameter even if the function returns false because there is more text following it.*

- bool ToDouble (double ∗val) const
  Attempts to convert the string to a floating point number.
- bool ToCDouble (double ∗val) const
  Variant of ToDouble() always working in "C" locale.
- bool ToLong (long ∗val, int base=10) const
  Attempts to convert the string to a signed integer in base base.
- bool ToCLong (long ∗val, int base=10) const
  Variant of ToLong() always working in "C" locale.
- bool ToLongLong (wxLongLong_t ∗val, int base=10) const
  This is exactly the same as ToLong() but works with 64 bit integer numbers.
- bool ToULong (unsigned long ∗val, int base=10) const
  Attempts to convert the string to an unsigned integer in base base.
- bool ToCULong (unsigned long ∗val, int base=10) const
  Variant of ToULong() always working in "C" locale.
- bool ToULongLong (wxULongLong_t ∗val, int base=10) const
  This is exactly the same as ToULong() but works with 64 bit integer numbers.

Formatting and printing

Both formatted versions (Printf/() and stream-like insertion operators exist (for basic types only).

See also the static Format() and FormatV() functions.

- int Printf (const wxString &pszFormat,...)
  Similar to the standard function sprintf().
- int PrintfV (const wxString &pszFormat, va_list argPtr)
  Similar to vprintf.

Memory management

The following are "advanced" functions and they will be needed rarely.

Alloc() and Shrink() are only interesting for optimization purposes. wxStringBuffer and wxStringBufferLength classes may be very useful when working with some external API which requires the caller to provide a writable buffer.

See also the reserve() and resize() STL-like functions.

- bool Alloc (size_t nLen)
  Preallocate enough space for wxString to store nLen characters.
- bool Shrink ()
  Minimizes the string's memory.
- wxString Clone () const
  Returns a deep copy of the string.
- void Clear ()

Empties the string and frees memory occupied by it.

Miscellaneous

Miscellaneous other string functions.

- bool Contains (const wxString &str) const
  Returns true if target appears anywhere in wxString; else false.
- void Empty ()
  Makes the string empty, but doesn't free memory occupied by the string.
- int Freq (wxUniChar ch) const
  Returns the number of occurrences of ch in the string.
- bool IsAscii () const
  Returns true if the string contains only ASCII characters.
- bool IsNumber () const
  Returns true if the string is an integer (with possible sign).
- bool IsWord () const
  Returns true if the string is a word.
- wxString & Pad (size_t count, wxUniChar chPad= ' ', bool fromRight=true)
  Adds count copies of chPad to the beginning, or to the end of the string (the default).
- wxString & Remove (size_t pos)
  Removes all characters from the string starting at pos.
- wxString & Remove (size_t pos, size_t len)
  Removes len characters from the string, starting at pos.
- wxString & RemoveLast (size_t n=1)
  Removes the last character.
- wxString Strip (stripType s=trailing) const
  Strip characters at the front and/or end.
- wxString & Trim (bool fromRight=true)
  Removes white-space (space, tabs, form feed, newline and carriage return) from the left or from the right end of the string (right is default).
- wxString & Truncate (size_t len)
  Truncate the string to the given length.

Iterator interface

These methods return iterators to the beginning or end of the string.

Please see any STL reference (e.g. http://www.cppreference.com/wiki/string/start*) for their documentation.*

- const_iterator begin () const
- iterator begin ()
- const_iterator end () const
- iterator end ()
- const_reverse_iterator rbegin () const
- reverse_iterator rbegin ()
- const_reverse_iterator rend () const
- reverse_iterator rend ()

STL interface

The supported STL functions are listed here.

Please see any STL reference (e.g. http://www.cppreference.com/wiki/string/start*) for their documentation.*

- wxString & append (const wxString &str, size_t pos, size_t n)
- wxString & append (const wxString &str)
- wxString & append (const char *sz, size_t n)
- wxString & append (const wchar_t *sz, size_t n)
- wxString & append (size_t n, wxUniChar ch)
- wxString & append (const_iterator first, const_iterator last)
- wxString & assign (const wxString &str, size_t pos, size_t n)

- wxString & assign (const wxString &str)
- wxString & assign (const char *sz, size_t n)
- wxString & assign (const wchar_t *sz, size_t n)
- wxString & assign (size_t n, wxUniChar ch)
- wxString & assign (const_iterator first, const_iterator last)
- wxUniChar at (size_t n) const
- wxUniCharRef at (size_t n)
- void clear ()
- size_type capacity () const
- int compare (const wxString &str) const
- int compare (size_t nStart, size_t nLen, const wxString &str) const
- int compare (size_t nStart, size_t nLen, const wxString &str, size_t nStart2, size_t nLen2) const
- int compare (size_t nStart, size_t nLen, const char *sz, size_t nCount=npos) const
- int compare (size_t nStart, size_t nLen, const wchar_t *sz, size_t nCount=npos) const
- wxCStrData data () const
- bool empty () const
- wxString & erase (size_type pos=0, size_type n=npos)
- iterator erase (iterator first, iterator last)
- iterator erase (iterator first)
- size_t find (const wxString &str, size_t nStart=0) const
- size_t find (const char *sz, size_t nStart=0, size_t n=npos) const
- size_t find (const wchar_t *sz, size_t nStart=0, size_t n=npos) const
- size_t find (wxUniChar ch, size_t nStart=0) const
- size_t find_first_of (const char *sz, size_t nStart=0) const
- size_t find_first_of (const wchar_t *sz, size_t nStart=0) const
- size_t find_first_of (const char *sz, size_t nStart, size_t n) const
- size_t find_first_of (const wchar_t *sz, size_t nStart, size_t n) const
- size_t find_first_of (wxUniChar c, size_t nStart=0) const
- size_t find_last_of (const wxString &str, size_t nStart=npos) const
- size_t find_last_of (const char *sz, size_t nStart=npos) const
- size_t find_last_of (const wchar_t *sz, size_t nStart=npos) const
- size_t find_last_of (const char *sz, size_t nStart, size_t n) const
- size_t find_last_of (const wchar_t *sz, size_t nStart, size_t n) const
- size_t find_last_of (wxUniChar c, size_t nStart=npos) const
- size_t find_first_not_of (const wxString &str, size_t nStart=0) const
- size_t find_first_not_of (const char *sz, size_t nStart=0) const
- size_t find_first_not_of (const wchar_t *sz, size_t nStart=0) const
- size_t find_first_not_of (const char *sz, size_t nStart, size_t n) const
- size_t find_first_not_of (const wchar_t *sz, size_t nStart, size_t n) const
- size_t find_first_not_of (wxUniChar ch, size_t nStart=0) const
- size_t find_last_not_of (const wxString &str, size_t nStart=npos) const
- size_t find_last_not_of (const char *sz, size_t nStart=npos) const
- size_t find_last_not_of (const wchar_t *sz, size_t nStart=npos) const
- size_t find_last_not_of (const char *sz, size_t nStart, size_t n) const
- size_t find_last_not_of (const wchar_t *sz, size_t nStart, size_t n) const
- wxString & insert (size_t nPos, const wxString &str)
- wxString & insert (size_t nPos, const wxString &str, size_t nStart, size_t n)
- wxString & insert (size_t nPos, const char *sz, size_t n)
- wxString & insert (size_t nPos, const wchar_t *sz, size_t n)
- wxString & insert (size_t nPos, size_t n, wxUniChar ch)
- iterator insert (iterator it, wxUniChar ch)
- void insert (iterator it, const_iterator first, const_iterator last)
- void insert (iterator it, size_type n, wxUniChar ch)
- size_t length () const
- size_type max_size () const
- void reserve (size_t sz)
- void resize (size_t nSize, wxUniChar ch= '\0')
- wxString & replace (size_t nStart, size_t nLen, const wxString &str)
- wxString & replace (size_t nStart, size_t nLen, size_t nCount, wxUniChar ch)
- wxString & replace (size_t nStart, size_t nLen, const wxString &str, size_t nStart2, size_t nLen2)
- wxString & replace (size_t nStart, size_t nLen, const char *sz, size_t nCount)
- wxString & replace (size_t nStart, size_t nLen, const wchar_t *sz, size_t nCount)
- wxString & replace (size_t nStart, size_t nLen, const wxString &s, size_t nCount)
- wxString & replace (iterator first, iterator last, const wxString &s)

- wxString & replace (iterator first, iterator last, const char *s, size_type n)
- wxString & replace (iterator first, iterator last, const wchar_t *s, size_type n)
- wxString & replace (iterator first, iterator last, size_type n, wxUniChar ch)
- wxString & replace (iterator first, iterator last, const_iterator first1, const_iterator last1)
- wxString & replace (iterator first, iterator last, const char *first1, const char *last1)
- wxString & replace (iterator first, iterator last, const wchar_t *first1, const wchar_t *last1)
- size_t rfind (const wxString &str, size_t nStart=npos) const
- size_t rfind (const char *sz, size_t nStart=npos, size_t n=npos) const
- size_t rfind (const wchar_t *sz, size_t nStart=npos, size_t n=npos) const
- size_t rfind (wxUniChar ch, size_t nStart=npos) const
- size_type size () const
- wxString substr (size_t nStart=0, size_t nLen=npos) const
- void swap (wxString &str)

Static Public Member Functions

- static wxString Format (const wxString &format,...)

  This static function returns the string containing the result of calling Printf() with the passed parameters on it.
- static wxString FormatV (const wxString &format, va_list argptr)

  This static function returns the string containing the result of calling PrintfV() with the passed parameters on it.
- static wxString FromCDouble (double val, int precision=-1)

  Returns a string with the textual representation of the number in C locale.
- static wxString FromDouble (double val, int precision=-1)

  Returns a string with the textual representation of the number.

- static wxString From8BitData (const char *buf, size_t len)

  Converts given buffer of binary data from 8-bit string to wxString.
- static wxString From8BitData (const char *buf)

  Converts given buffer of binary data from 8-bit string to wxString.

- static wxString FromAscii (const char *s)

  Converts the string or character from an ASCII, 7-bit form to the native wxString representation.
- static wxString FromAscii (const unsigned char *s)

  Converts the string or character from an ASCII, 7-bit form to the native wxString representation.
- static wxString FromAscii (const char *s, size_t len)

  Converts the string or character from an ASCII, 7-bit form to the native wxString representation.
- static wxString FromAscii (const unsigned char *s, size_t len)

  Converts the string or character from an ASCII, 7-bit form to the native wxString representation.
- static wxString FromAscii (char c)

  Converts the string or character from an ASCII, 7-bit form to the native wxString representation.

- static wxString FromUTF8 (const char *s)

  Converts C string encoded in UTF-8 to wxString.
- static wxString FromUTF8 (const char *s, size_t len)

  Converts C string encoded in UTF-8 to wxString.

- static wxString FromUTF8Unchecked (const char *s)

  Converts C string encoded in UTF-8 to wxString without checking its validity.
- static wxString FromUTF8Unchecked (const char *s, size_t len)

  Converts C string encoded in UTF-8 to wxString without checking its validity.

Static Public Attributes

- static const size_t npos

  An 'invalid' value for string index.

21.730.7 Member Typedef Documentation

typedef wxUniChar wxString::char_type

typedef const wxChar* wxString::const_pointer

typedef wxUniChar wxString::const_reference

typedef wxChar* wxString::pointer

typedef wxUniCharRef wxString::reference

typedef size_t wxString::size_type

typedef wxUniChar wxString::value_type

21.730.8 Constructor & Destructor Documentation

wxString::wxString ( )

Default constructor.

wxString::wxString (const wxString & *stringSrc*)

Creates a string from another string.

Just increases the ref count by 1.

wxString::wxString (wxUniChar *ch*, size_t *nRepeat* = 1)

Construct a string consisting of *nRepeat* copies of ch.

wxString::wxString (wxUniCharRef *ch*, size_t *nRepeat* = 1)

Construct a string consisting of *nRepeat* copies of ch.

wxString::wxString (char *ch*, size_t *nRepeat* = 1)

Construct a string consisting of *nRepeat* copies of ch converted to Unicode using the current locale encoding.

wxString::wxString (wchar_t *ch*, size_t *nRepeat* = 1)

Construct a string consisting of *nRepeat* copies of ch.

wxString::wxString (const char * *psz*)

Constructs a string from the string literal *psz* using the current locale encoding to convert it to Unicode (wxConvLibc).

wxString::wxString (const char * *psz*, const wxMBConv & *conv*)

Constructs a string from the string literal *psz* using *conv* to convert it Unicode.

wxString::wxString (const char * *psz*, size_t *nLength*)

Constructs a string from the first *nLength* character of the string literal *psz* using the current locale encoding to convert it to Unicode (wxConvLibc).

wxString::wxString (const char * *psz*, const wxMBConv & *conv*, size_t *nLength*)

Constructs a string from the first *nLength* character of the string literal *psz* using *conv* to convert it Unicode.

wxString::wxString (const wchar_t * *pwz*)

Constructs a string from the string literal *pwz*.

wxString::wxString (const wchar_t * *pwz*, size_t *nLength*)

Constructs a string from the first *nLength* characters of the string literal *pwz*.

wxString::wxString (const wxCharBuffer & *buf*)

Constructs a string from *buf* using the using the current locale encoding to convert it to Unicode.

wxString::wxString (const wxWCharBuffer & *buf*)

Constructs a string from *buf*.

wxString::wxString (const std::string & *str*)

Constructs a string from *str* using the using the current locale encoding to convert it to Unicode (wxConvLibc).

See also

ToStdString()

wxString::wxString (const std::wstring & *str*)

Constructs a string from *str*.

See also

ToStdWstring()

wxString::~wxString ( )

String destructor.

Note that this is not virtual, so wxString must not be inherited from.

21.730.9 Member Function Documentation

wxString wxString::AfterFirst (wxUniChar *ch*) const

Gets all the characters after the first occurrence of *ch*.

Returns the empty string if *ch* is not found.

wxString wxString::AfterLast (wxUniChar *ch*) const

Gets all the characters after the last occurrence of *ch*.

Returns the whole string if *ch* is not found.

bool wxString::Alloc (size_t *nLen*)

Preallocate enough space for wxString to store *nLen* characters.

Please note that this method does the same thing as the standard reserve() one and shouldn't be used in new code.

This function may be used to increase speed when the string is constructed by repeated concatenation as in

```
// delete all vowels from the string
wxString DeleteAllVowels(const wxString& original)
{
    wxString result;

    size_t len = original.length();

    result.Alloc(len);

    for ( size_t n = 0; n < len; n++ )
    {
        if ( strchr("aeuio", tolower(original[n])) == NULL )
            result += original[n];
    }

    return result;
}
```

because it will avoid the need to reallocate string memory many times (in case of long strings). Note that it does not set the maximal length of a string – it will still expand if more than *nLen* characters are stored in it. Also, it does not truncate the existing string (use Truncate() for this) even if its current length is greater than *nLen*.

Returns

true if memory was successfully allocated, false otherwise.

wxString& wxString::Append (const char * *psz*)

Appends the string literal *psz*.

wxString& wxString::Append (const wchar_t * *pwz*)

Appends the wide string literal *pwz*.

wxString& wxString::Append (const char * *psz*, size_t *nLen*)

Appends the string literal *psz* with max length *nLen*.

wxString& wxString::Append (const wchar_t * *pwz*, size_t *nLen*)

Appends the wide string literal *psz* with max length *nLen*.

wxString& wxString::Append (const wxString & *s*)

Appends the string *s*.

wxString& wxString::Append (wxUniChar *ch*, size_t *count* = `1u`)

Appends the character *ch count* times.

wxString& wxString::append (const wxString & *str*, size_t *pos*, size_t *n*)

wxString& wxString::append (const wxString & *str*)

wxString& wxString::append (const char * *sz*, size_t *n*)

wxString& wxString::append (const wchar_t * *sz*, size_t *n*)

wxString& wxString::append (size_t *n*, wxUniChar *ch*)

wxString& wxString::append (const_iterator *first*, const_iterator *last*)

wxString& wxString::assign (const wxString & *str*, size_t *pos*, size_t *n*)

wxString& wxString::assign (const wxString & *str*)

wxString& wxString::assign (const char * *sz*, size_t *n*)

wxString& wxString::assign (const wchar_t * *sz*, size_t *n*)

wxString& wxString::assign (size_t *n*, wxUniChar *ch*)

wxString& wxString::assign (const_iterator *first*, const_iterator *last*)

wxUniChar wxString::at (size_t *n*) const

wxUniCharRef wxString::at (size_t *n*)

wxString wxString::BeforeFirst (wxUniChar *ch*, wxString * *rest* = `NULL`) const

Gets all characters before the first occurrence of *ch*.

Returns the whole string if *ch* is not found.

Parameters

ch	The character to look for.
rest	Filled with the part of the string following the first occurrence of *ch* or cleared if it was not found. The same string is returned by AfterFirst() but it is more efficient to use this output parameter if both the "before" and "after" parts are needed than calling both functions one after the other. This parameter is available in wxWidgets version 2.9.2 and later only.

Returns

Part of the string before the first occurrence of *ch*.

wxString wxString::BeforeLast (wxUniChar *ch*, wxString * *rest* = NULL) const

Gets all characters before the last occurrence of *ch*.

Returns the empty string if *ch* is not found.

Parameters

ch	The character to look for.
rest	Filled with the part of the string following the last occurrence of *ch* or the copy of this string if it was not found. The same string is returned by AfterLast() but it is more efficient to use this output parameter if both the "before" and "after" parts are needed than calling both functions one after the other. This parameter is available in wxWidgets version 2.9.2 and later only.

Returns

Part of the string before the last occurrence of *ch*.

const_iterator wxString::begin ( ) const

iterator wxString::begin ( )

wxCStrData wxString::c_str ( ) const

Returns a lightweight intermediate class which is in turn implicitly convertible to both `const char*` and to `const wchar_t*`.

Given this ambiguity it is mostly better to use wc_str(), mb_str() or utf8_str() instead.

Please see the Unicode Support in wxWidgets for more information about it.

Note that the returned value is not convertible to `char*` or `wchar_t*`, use char_str() or wchar_str() if you need to pass string value to a function expecting non-const pointer.

See also

wc_str(), utf8_str(), c_str(), mb_str(), fn_str()

size_type wxString::capacity ( ) const

wxString wxString::Capitalize ( ) const

Return the copy of the string with the first string character in the upper case and the subsequent ones in the lower case.

Since

2.9.0

See also

MakeCapitalized()

wxWritableCharBuffer wxString::char_str (const wxMBConv & *conv* = `wxConvLibc`) const

Returns an object with string data that is implicitly convertible to `char*` pointer.

Note that any change to the returned buffer is lost and so this function is only usable for passing strings to legacy libraries that don't have const-correct API. Use wxStringBuffer if you want to modify the string.

See also

c_str()

void wxString::Clear ( )

Empties the string and frees memory occupied by it.

See also

Empty()

void wxString::clear ( )

wxString wxString::Clone ( ) const

Returns a deep copy of the string.

That is, the returned string is guaranteed to not share data with this string when using reference-counted wxString implementation.

This method is primarily useful for passing strings between threads (because wxString is not thread-safe). Unlike creating a copy using `wxString(c_str())`, Clone() handles embedded NULs correctly.

Since

2.9.0

int wxString::Cmp (const wxString & *s*) const

Case-sensitive comparison.

Returns a positive value if the string is greater than the argument, zero if it is equal to it or a negative value if it is less than the argument (same semantics as the standard `strcmp()` function).

See also

CmpNoCase(), IsSameAs().

int wxString::CmpNoCase (const wxString & *s*) const

Case-insensitive comparison.

Returns a positive value if the string is greater than the argument, zero if it is equal to it or a negative value if it is less than the argument (same semantics as the standard `strcmp()` function).

See also

Cmp(), IsSameAs().

int wxString::compare (const wxString & *str*) const

int wxString::compare (size_t *nStart*, size_t *nLen*, const wxString & *str*) const

int wxString::compare (size_t *nStart*, size_t *nLen*, const wxString & *str*, size_t *nStart2*, size_t *nLen2*) const

int wxString::compare (size_t *nStart*, size_t *nLen*, const char * *sz*, size_t *nCount* = npos) const

int wxString::compare (size_t *nStart*, size_t *nLen*, const wchar_t * *sz*, size_t *nCount* = npos) const

bool wxString::Contains (const wxString & *str*) const

Returns true if target appears anywhere in wxString; else false.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxCStrData wxString::data ( ) const

void wxString::Empty ( )

Makes the string empty, but doesn't free memory occupied by the string.

See also

Clear().

bool wxString::empty ( ) const

const_iterator wxString::end ( ) const

iterator wxString::end ( )

bool wxString::EndsWith (const wxString & *suffix*, wxString * *rest* = NULL) const

This function can be used to test if the string ends with the specified *suffix*.

If it does, the function will return true and put the beginning of the string before the suffix into *rest* string if it is not NULL. Otherwise, the function returns false and doesn't modify the *rest*.

wxString& wxString::erase (size_type *pos* = 0, size_type *n* = npos)

iterator wxString::erase (iterator *first*, iterator *last*)

iterator wxString::erase (iterator *first*)

int wxString::Find (wxUniChar *ch*, bool *fromEnd* = false) const

Searches for the given character *ch*.

Returns the position or wxNOT_FOUND if not found.

int wxString::Find (const wxString & *sub*) const

Searches for the given string *sub*.

Returns the starting position or wxNOT_FOUND if not found.

size_t wxString::find (const wxString & *str,* size_t *nStart* = 0) const

size_t wxString::find (const char * *sz,* size_t *nStart* = 0, size_t *n* = npos) const

size_t wxString::find (const wchar_t * *sz,* size_t *nStart* = 0, size_t *n* = npos) const

size_t wxString::find (wxUniChar *ch,* size_t *nStart* = 0) const

size_t wxString::find_first_not_of (const wxString & *str,* size_t *nStart* = 0) const

size_t wxString::find_first_not_of (const char * *sz,* size_t *nStart* = 0) const

size_t wxString::find_first_not_of (const wchar_t * *sz,* size_t *nStart* = 0) const

size_t wxString::find_first_not_of (const char * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_first_not_of (const wchar_t * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_first_not_of (wxUniChar *ch,* size_t *nStart* = 0) const

size_t wxString::find_first_of (const char * *sz,* size_t *nStart* = 0) const

size_t wxString::find_first_of (const wchar_t * *sz,* size_t *nStart* = 0) const

size_t wxString::find_first_of (const char * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_first_of (const wchar_t * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_first_of (wxUniChar *c,* size_t *nStart* = 0) const

size_t wxString::find_last_not_of (const wxString & *str,* size_t *nStart* = npos) const

size_t wxString::find_last_not_of (const char * *sz,* size_t *nStart* = npos) const

size_t wxString::find_last_not_of (const wchar_t * *sz,* size_t *nStart* = npos) const

size_t wxString::find_last_not_of (const char * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_last_not_of (const wchar_t * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_last_of (const wxString & *str,* size_t *nStart* = npos) const

size_t wxString::find_last_of (const char * *sz,* size_t *nStart* = npos) const

size_t wxString::find_last_of (const wchar_t * *sz,* size_t *nStart* = npos) const

size_t wxString::find_last_of (const char * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_last_of (const wchar_t * *sz,* size_t *nStart,* size_t *n*) const

size_t wxString::find_last_of (wxUniChar *c,* size_t *nStart* = npos) const

int wxString::First (wxUniChar *ch*) const

Same as Find().

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

int wxString::First (const wxString & *str*) const

Same as Find().

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

const wchar_t* wxString::fn_str ( ) const

Returns a string representation suitable for passing to OS' functions for file handling.

const char* wxString::fn_str ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

const wxCharBuffer wxString::fn_str ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

static wxString wxString::Format (const wxString & *format, ...*) [static]

This static function returns the string containing the result of calling Printf() with the passed parameters on it.

See also

FormatV(), Printf()

static wxString wxString::FormatV (const wxString & *format,* va_list *argptr*) [static]

This static function returns the string containing the result of calling PrintfV() with the passed parameters on it.

See also

Format(), PrintfV()

int wxString::Freq (wxUniChar *ch*) const

Returns the number of occurrences of *ch* in the string.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

static wxString wxString::From8BitData (const char * *buf,* size_t *len*) [static]

Converts given buffer of binary data from 8-bit string to wxString.

In Unicode build, the string is interpreted as being in ISO-8859-1 encoding. The version without *len* parameter takes NUL-terminated data.

This is a convenience method useful when storing binary data in wxString. It should be used *only* for that purpose and only in conjunction with To8BitData(). Use mb_str() for conversion of character data to known encoding.

Since

2.8.4

See also

wxString::To8BitData()

static wxString wxString::From8BitData (const char * *buf*) [static]

Converts given buffer of binary data from 8-bit string to wxString.

In Unicode build, the string is interpreted as being in ISO-8859-1 encoding. The version without *len* parameter takes NUL-terminated data.

This is a convenience method useful when storing binary data in wxString. It should be used *only* for that purpose and only in conjunction with To8BitData(). Use mb_str() for conversion of character data to known encoding.

Since

2.8.4

See also

wxString::To8BitData()

static wxString wxString::FromAscii (const char * *s*) [static]

Converts the string or character from an ASCII, 7-bit form to the native wxString representation.

static wxString wxString::FromAscii (const unsigned char * *s*) [static]

Converts the string or character from an ASCII, 7-bit form to the native wxString representation.

static wxString wxString::FromAscii (const char * *s*, size_t *len*) [static]

Converts the string or character from an ASCII, 7-bit form to the native wxString representation.

static wxString wxString::FromAscii (const unsigned char * *s*, size_t *len*) [static]

Converts the string or character from an ASCII, 7-bit form to the native wxString representation.

static wxString wxString::FromAscii (char *c*) [static]

Converts the string or character from an ASCII, 7-bit form to the native wxString representation.

static wxString wxString::FromCDouble (double *val*, int *precision* = -1) [static]

Returns a string with the textual representation of the number in C locale.

Unlike FromDouble() the string returned by this function always uses the period character as decimal separator, independently of the current locale. Otherwise its behaviour is identical to the other function.

Since

2.9.1

See also

ToCDouble()

static wxString wxString::FromDouble (double *val,* int *precision* = -1) [static]

Returns a string with the textual representation of the number.

For the default value of *precision*, this function behaves as a simple wrapper for

```
wxString::Format("%g", val)
```

. If *precision* is positive (or zero), the %.Nf format is used with the given precision value.

Notice that the string returned by this function uses the decimal separator appropriate for the current locale, e.g. "," and not a period in French locale. Use FromCDouble() if this is unwanted.

Parameters

val	The value to format.
precision	The number of fractional digits to use in or -1 to use the most appropriate format. This parameter is new in wxWidgets 2.9.2.

Since

2.9.1

See also

ToDouble()

static wxString wxString::FromUTF8 (const char * *s*) [static]

Converts C string encoded in UTF-8 to wxString.

If *s* is not a valid UTF-8 string, an empty string is returned.

Notice that when using UTF-8 wxWidgets build there is a more efficient alternative to this function called FromUTF8Unchecked() which, unlike this one, doesn't check that the input string is valid.

Since

2.8.4

static wxString wxString::FromUTF8 (const char * *s,* size_t *len*) [static]

Converts C string encoded in UTF-8 to wxString.

If *s* is not a valid UTF-8 string, an empty string is returned.

Notice that when using UTF-8 wxWidgets build there is a more efficient alternative to this function called FromUTF8Unchecked() which, unlike this one, doesn't check that the input string is valid.

Since

2.8.4

static wxString wxString::FromUTF8Unchecked (const char * *s*) [static]

Converts C string encoded in UTF-8 to wxString without checking its validity.

This method assumes that *s* is a valid UTF-8 sequence and doesn't do any validation (although an assert failure is triggered in debug builds if the string is invalid). Only use it if you are absolutely sure that *s* is a correct UTF-8 string (e.g. because it comes from another library using UTF-8) and if the performance matters, otherwise use slower (in UTF-8 build) but safer FromUTF8(). Passing a bad UTF-8 string to this function will result in creating a corrupted wxString and all the subsequent operations on it will be undefined.

Since

2.8.9

static wxString wxString::FromUTF8Unchecked (const char * *s*, size_t *len*) [static]

Converts C string encoded in UTF-8 to wxString without checking its validity.

This method assumes that *s* is a valid UTF-8 sequence and doesn't do any validation (although an assert failure is triggered in debug builds if the string is invalid). Only use it if you are absolutely sure that *s* is a correct UTF-8 string (e.g. because it comes from another library using UTF-8) and if the performance matters, otherwise use slower (in UTF-8 build) but safer FromUTF8(). Passing a bad UTF-8 string to this function will result in creating a corrupted wxString and all the subsequent operations on it will be undefined.

Since

2.8.9

wxUniChar wxString::GetChar (size_t *n*) const

Returns the character at position *n* (read-only).

const wxCStrData wxString::GetData ( ) const

wxWidgets compatibility conversion.

Same as c_str().

wxUniCharRef wxString::GetWritableChar (size_t *n*)

Returns a reference to the character at position *n*.

wxStringCharType* wxString::GetWriteBuf (size_t *len*)

Returns a writable buffer of at least *len* bytes.

It returns a pointer to a new memory block, and the existing data will not be copied. Call UngetWriteBuf() as soon as possible to put the string back into a reasonable state.

This method is deprecated, please use wxStringBuffer or wxStringBufferLength instead.

wxString& wxString::insert (size_t *nPos*, const wxString & *str*)

wxString& wxString::insert (size_t *nPos*, const wxString & *str*, size_t *nStart*, size_t *n*)

wxString& wxString::insert (size_t *nPos,* const char * *sz,* size_t *n*)

wxString& wxString::insert (size_t *nPos,* const wchar_t * *sz,* size_t *n*)

wxString& wxString::insert (size_t *nPos,* size_t *n,* wxUniChar *ch*)

iterator wxString::insert (iterator *it,* wxUniChar *ch*)

void wxString::insert (iterator *it,* const_iterator *first,* const_iterator *last*)

void wxString::insert (iterator *it,* size_type *n,* wxUniChar *ch*)

bool wxString::IsAscii ( ) const

Returns true if the string contains only ASCII characters.

See wxUniChar::IsAscii for more details.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

bool wxString::IsEmpty ( ) const

Returns true if the string is empty.

bool wxString::IsNull ( ) const

Returns true if the string is empty (same as wxString::IsEmpty).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

bool wxString::IsNumber ( ) const

Returns true if the string is an integer (with possible sign).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

bool wxString::IsSameAs (const wxString & *s,* bool *caseSensitive* = `true`) const

Test whether the string is equal to another string *s*.

The test is case-sensitive if *caseSensitive* is true (default) or not if it is false.

Returns

true if the string is equal to the other one, false otherwise.

See also

Cmp(), CmpNoCase()

bool wxString::IsSameAs (wxUniChar *ch,* bool *caseSensitive* = `true`) const

Test whether the string is equal to the single character *ch*.

The test is case-sensitive if *caseSensitive* is true (default) or not if it is false.

Returns

true if the string is equal to this character, false otherwise.

See also

Cmp(), CmpNoCase()

bool wxString::IsWord ( ) const

Returns true if the string is a word.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxUniChar wxString::Last ( ) const

Returns the last character.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxUniCharRef wxString::Last ( )

Returns a reference to the last character (writable).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxString wxString::Left (size_t *count*) const

Returns the first *count* characters of the string.

size_t wxString::Len ( ) const

Returns the length of the string.

size_t wxString::Length ( ) const

Returns the length of the string (same as Len).

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

size_t wxString::length ( ) const

wxString wxString::Lower ( ) const

Returns this string converted to the lower case.

See also

MakeLower()

void wxString::LowerCase ( )

Same as MakeLower.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxString& wxString::MakeCapitalized ( )

Converts the first characters of the string to the upper case and all the subsequent ones to the lower case and returns the result.

Since

2.9.0

See also

Capitalize()

wxString& wxString::MakeLower ( )

Converts all characters to lower case and returns the reference to the modified string.

See also

Lower()

wxString& wxString::MakeUpper ( )

Converts all characters to upper case and returns the reference to the modified string.

See also

Upper()

bool wxString::Matches (const wxString & *mask*) const

Returns true if the string contents matches a mask containing '*' and '?'.

size_type wxString::max_size ( ) const

const wxCharBuffer wxString::mb_str (const wxMBConv & *conv* = `wxConvLibc`) const

Returns the multibyte (C string) representation of the string using *conv's* wxMBConv::cWC2MB method and returns wxCharBuffer.

See also

wc_str(), utf8_str(), c_str(), wxMBConv

wxString wxString::Mid (size_t *first*, size_t *nCount* = wxString::npos) const

Returns a substring starting at *first*, with length *count*, or the rest of the string if *count* is the default value.

bool wxString::operator! ( ) const

Empty string is false, so !string will only return true if the string is empty.

See also

IsEmpty().

wxString wxString::operator() (size_t *start,* size_t *len*) const

Same as Mid() (substring extraction).

wxString wxString::operator+ (const wxString & *x,* const wxString & *y*)

Concatenation: returns a new string equal to the concatenation of the operands.

wxString wxString::operator+ (const wxString & *x,* wxUniChar *y*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxString::operator+= (const wxString & *str*)

Concatenation in place: the argument is appended to the string.

void wxString::operator+= (wxUniChar *c*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxString& wxString::operator<< (const wxString & *s*)

Appends the string literal *psz*.

wxString& wxString::operator<< (const char * *psz*)

Appends the string literal *psz*.

wxString& wxString::operator<< (const wchar_t * *pwz*)

Appends the string literal *psz*.

wxString& wxString::operator<< (const wxCStrData & *psz*)

Appends the string literal *psz*.

wxString& wxString::operator<< (char *ch*)

Appends the string literal *psz*.

wxString& wxString::operator<< (unsigned char *ch*)

Appends the string literal *psz*.

wxString& wxString::operator<< (wchar_t *ch*)

Appends the string literal *psz*.

wxString& wxString::operator<< (const wxCharBuffer & *s*)

Appends the string literal *psz*.

wxString& wxString::operator<< (const wxWCharBuffer & *s*)

Appends the string literal *psz*.

wxString& wxString::operator<< (wxUniChar *ch*)

Appends the string literal *psz*.

wxString& wxString::operator<< (wxUniCharRef *ch*)

Appends the string literal *psz*.

wxString& wxString::operator<< (unsigned int *ui*)

Appends the string literal *psz*.

wxString& wxString::operator<< (long *l*)

Appends the string literal *psz*.

wxString& wxString::operator<< (unsigned long *ul*)

Appends the string literal *psz*.

wxString& wxString::operator<< (wxLongLong_t *ll*)

Appends the string literal *psz*.

wxString& wxString::operator<< (wxULongLong_t *ul*)

Appends the string literal *psz*.

wxString& wxString::operator<< (float *f*)

Appends the string literal *psz*.

wxString& wxString::operator<< (double *d*)

Appends the string literal *psz*.

wxString wxString::operator= (const wxString & *str*)

Assignment: see the relative wxString constructor.

wxString wxString::operator= (wxUniChar *c*)

Assignment: see the relative wxString constructor.

wxUniChar wxString::operator[] (size_t *i*) const

Returns the *i-th* character of the string.

wxUniCharRef wxString::operator[] (size_t *i*)

Returns a writable reference to the *i-th* character of the string.

wxString& wxString::Pad (size_t *count*, wxUniChar *chPad* = ' ', bool *fromRight* = true)

Adds *count* copies of *chPad* to the beginning, or to the end of the string (the default).

Removes spaces from the left or from the right (default).

wxString& wxString::Prepend (const wxString & *str*)

Prepends *str* to this string, returning a reference to this string.

int wxString::Printf (const wxString & *pszFormat*, ...)

Similar to the standard function *sprintf()*.

Returns the number of characters written, or an integer less than zero on error. Note that if wxUSE_PRINTF_POS_PARAMS is set to 1, then this function supports Unix98-style positional parameters:

```
wxString str;

str.Printf(wxT("%d %d %d"), 1, 2, 3);
// str now contains "1 2 3"

str.Printf(wxT("%2$d %3$d %1$d"), 1, 2, 3);
// str now contains "2 3 1"
```

Note

This function will use a safe version of *vsprintf()* (usually called *vsnprintf()*) whenever available to always allocate the buffer of correct size. Unfortunately, this function is not available on all platforms and the dangerous *vsprintf()* will be used then which may lead to buffer overflows.

int wxString::PrintfV (const wxString & *pszFormat*, va_list *argPtr*)

Similar to vprintf.

Returns the number of characters written, or an integer less than zero on error.

const_reverse_iterator wxString::rbegin ( ) const

reverse_iterator wxString::rbegin ( )

wxString& wxString::Remove (size_t *pos*)

Removes all characters from the string starting at *pos*.

Use Truncate() as a more readable alternative.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxString& wxString::Remove (size_t *pos*, size_t *len*)

Removes *len* characters from the string, starting at *pos*.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxString& wxString::RemoveLast (size_t *n* = `1`)

Removes the last character.

const_reverse_iterator wxString::rend ( ) const

reverse_iterator wxString::rend ( )

size_t wxString::Replace (const wxString & *strOld*, const wxString & *strNew*, bool *replaceAll* = `true`)

Replace first (or all) occurrences of substring with another one.

Parameters

strOld	The string to search for replacing.
strNew	The substitution string.
replaceAll	If true a global replace will be done (default), otherwise only the first occurrence will be replaced.

Returns the number of replacements made.

wxString& wxString::replace (size_t *nStart*, size_t *nLen*, const wxString & *str*)

wxString& wxString::replace (size_t *nStart*, size_t *nLen*, size_t *nCount*, wxUniChar *ch*)

wxString& wxString::replace (size_t *nStart*, size_t *nLen*, const wxString & *str*, size_t *nStart2*, size_t *nLen2*)

wxString& wxString::replace (size_t *nStart*, size_t *nLen*, const char * *sz*, size_t *nCount*)

wxString& wxString::replace (size_t *nStart*, size_t *nLen*, const wchar_t * *sz*, size_t *nCount*)

wxString& wxString::replace (size_t *nStart*, size_t *nLen*, const wxString & *s*, size_t *nCount*)

wxString& wxString::replace (iterator *first*, iterator *last*, const wxString & *s*)

wxString& wxString::replace (iterator *first*, iterator *last*, const char * *s*, size_type *n*)

wxString& wxString::replace (iterator *first*, iterator *last*, const wchar_t * *s*, size_type *n*)

wxString& wxString::replace (iterator *first*, iterator *last*, size_type *n*, wxUniChar *ch*)

wxString& wxString::replace (iterator *first*, iterator *last*, const_iterator *first1*, const_iterator *last1*)

wxString& wxString::replace (iterator *first*, iterator *last*, const char * *first1*, const char * *last1*)

wxString& wxString::replace (iterator *first*, iterator *last*, const wchar_t * *first1*, const wchar_t * *last1*)

void wxString::reserve (size_t *sz*)

void wxString::resize (size_t *nSize,* wxUniChar *ch* = '\0')

size_t wxString::rfind (const wxString & *str,* size_t *nStart* = npos) const

size_t wxString::rfind (const char * *sz,* size_t *nStart* = npos, size_t *n* = npos) const

size_t wxString::rfind (const wchar_t * *sz,* size_t *nStart* = npos, size_t *n* = npos) const

size_t wxString::rfind (wxUniChar *ch,* size_t *nStart* = npos) const

wxString wxString::Right (size_t *count*) const

Returns the last *count* characters.

void wxString::SetChar (size_t *n,* wxUniChar *ch*)

Sets the character at position *n*.

bool wxString::Shrink ( )

Minimizes the string's memory.

This can be useful after a call to Alloc() if too much memory were preallocated.

size_type wxString::size ( ) const

bool wxString::StartsWith (const wxString & *prefix,* wxString * *rest* = NULL) const

This function can be used to test if the string starts with the specified *prefix*.

If it does, the function will return true and put the rest of the string (i.e. after the prefix) into *rest* string if it is not NULL. Otherwise, the function returns false and doesn't modify the *rest*.

wxString wxString::Strip (stripType *s* = trailing) const

Strip characters at the front and/or end.

This is the same as Trim() except that it doesn't change this string.

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

wxString wxString::substr (size_t *nStart* = 0, size_t *nLen* = npos) const

wxString wxString::SubString (size_t *from,* size_t *to*) const

Returns the part of the string between the indices *from* and *to* inclusive.

This is a wxWidgets 1.xx compatibility function, use Mid() instead (but note that parameters have different meaning).

void wxString::swap (wxString & *str*)

template<typename T > wxCharTypeBuffer<T> wxString::tchar_str (size_t * *len* = NULL) const

Returns buffer of the specified type containing the string data.

This method is only useful in template code, otherwise you should directly call mb_str() or wc_str() if you need to retrieve a narrow or wide string from this wxString. The template parameter *t* should be either `char` or `wchar_t`.

Notice that retrieving a char buffer in UTF-8 build will return the internal string representation in UTF-8 while in wchar_t build the char buffer will contain the conversion of the string to the encoding of the current locale (and so can fail).

Parameters

len	If non-NULL, filled with the length of the returned buffer.

Returns

buffer containing the string contents in the specified type, notice that it may be NULL if the conversion failed (e.g. Unicode string couldn't be converted to the current encoding when *T* is `char`).

const wxScopedCharBuffer wxString::To8BitData ( ) const

Converts the string to an 8-bit string in ISO-8859-1 encoding in the form of a wxCharBuffer (Unicode builds only).

This is a convenience method useful when storing binary data in wxString. It should be used *only* for this purpose. It is only valid to call this method on strings created using From8BitData().

Since

2.8.4

See also

wxString::From8BitData()

const char* wxString::ToAscii ( ) const

Converts the string to an ASCII, 7-bit string in the form of a wxCharBuffer (Unicode builds only) or a C string (ANSI builds).

Note that this conversion is only lossless if the string contains only ASCII characters as all the non-ASCII ones are replaced with the `'_'` (underscore) character.

Use mb_str() or utf8_str() to convert to other encodings.

const wxCharBuffer wxString::ToAscii ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxString::ToCDouble (double * *val*) const

Variant of ToDouble() always working in "C" locale.

Works like ToDouble() but unlike it this function expects the floating point number to be formatted always with the rules dictated by the "C" locale (in particular, the decimal point must be a dot), independently from the current application-wide locale (see wxLocale).

See also

ToDouble(), ToLong(), ToULong()

bool wxString::ToCLong (long * *val,* int *base* = 10) const

Variant of ToLong() always working in "C" locale.

Works like ToLong() but unlike it this function expects the integer number to be formatted always with the rules dictated by the "C" locale, independently from the current application-wide locale (see wxLocale).

See also

ToDouble(), ToLong(), ToULong()

bool wxString::ToCULong (unsigned long * *val,* int *base* = 10) const

Variant of ToULong() always working in "C" locale.

Works like ToULong() but unlike it this function expects the integer number to be formatted always with the rules dictated by the "C" locale, independently from the current application-wide locale (see wxLocale).

See also

ToDouble(), ToLong(), ToULong()

bool wxString::ToDouble (double * *val*) const

Attempts to convert the string to a floating point number.

Returns true on success (the number is stored in the location pointed to by *val*) or false if the string does not represent such number (the value of *val* may still be modified in this case).

Note that unlike ToCDouble() this function uses a localized version of `wxStrtod()` and thus needs as decimal point (and thousands separator) the locale-specific decimal point. Thus you should use this function only when you are sure that this string contains a floating point number formatted with the rules of the locale currently in use (see wxLocale).

Also notice that even this function is locale-specific it does not support strings with thousands separators in them, even if the current locale uses digits grouping. You may use wxNumberFormatter::FromString() to parse such strings.

Please refer to the documentation of the standard function `strtod()` for more details about the supported syntax.

See also

ToCDouble(), ToLong(), ToULong()

bool wxString::ToLong (long * *val,* int *base* = 10) const

Attempts to convert the string to a signed integer in base *base*.

Returns true on success in which case the number is stored in the location pointed to by *val* or false if the string does not represent a valid number in the given base (the value of *val* may still be modified in this case).

The value of *base* must be comprised between 2 and 36, inclusive, or be a special value 0 which means that the usual rules of `C` numbers are applied: if the number starts with `0x` it is considered to be in base 16, if it starts with `0` - in base 8 and in base 10 otherwise. Note that you may not want to specify the base 0 if you are parsing the numbers which may have leading zeroes as they can yield unexpected (to the user not familiar with C) results.

Note that unlike ToCLong() this function uses a localized version of `wxStrtol()`. Thus you should use this function only when you are sure that this string contains an integer number formatted with the rules of the locale currently in use (see wxLocale).

As with ToDouble(), this function does not support strings containing thousands separators even if the current locale uses digits grouping. You may use wxNumberFormatter::FromString() to parse such strings.

Please refer to the documentation of the standard function `strtol()` for more details about the supported syntax.

See also

ToCDouble(), ToDouble(), ToULong()

bool wxString::ToLongLong (wxLongLong_t * *val,* int *base* = `10`) const

This is exactly the same as ToLong() but works with 64 bit integer numbers.

Notice that currently it doesn't work (always returns false) if parsing of 64 bit numbers is not supported by the underlying C run-time library. Compilers with C99 support and Microsoft Visual C++ version 7 and higher do support this.

See also

ToLong(), ToULongLong()

std::string wxString::ToStdString ( ) const

Return the string as an std::string in current locale encoding.

Note that if the conversion of (Unicode) string contents to the current locale fails, the return string will be empty. Be sure to check for this to avoid silent data loss.

Instead of using this function it's also possible to write

```
std::string s;
wxString wxs;
...
s = std::string(wxs);
```

but using ToStdString() may make the code more clear.

Since

2.9.1

std::wstring wxString::ToStdWstring ( ) const

Return the string as an std::wstring.

Unlike ToStdString(), there is no danger of data loss when using this function.

Since

2.9.1

bool wxString::ToULong (unsigned long * *val*, int *base* = `10`) const

Attempts to convert the string to an unsigned integer in base *base*.

Returns true on success in which case the number is stored in the location pointed to by *val* or false if the string does not represent a valid number in the given base (the value of *val* may still be modified in this case).

Please notice that this function behaves in the same way as the standard `strtoul()` and so it simply converts negative numbers to unsigned representation instead of rejecting them (e.g. -1 is returned as `ULONG_MAX`).

See ToLong() for the more detailed description of the *base* parameter (and of the locale-specific behaviour of this function).

See also

ToCULong(), ToDouble(), ToLong()

bool wxString::ToULongLong (wxULongLong_t * *val*, int *base* = `10`) const

This is exactly the same as ToULong() but works with 64 bit integer numbers.

Please see ToLongLong() for additional remarks.

const wxScopedCharBuffer wxString::ToUTF8 ( ) const

Same as utf8_str().

wxString& wxString::Trim (bool *fromRight* = `true`)

Removes white-space (space, tabs, form feed, newline and carriage return) from the left or from the right end of the string (right is default).

wxString& wxString::Truncate (size_t *len*)

Truncate the string to the given length.

void wxString::UngetWriteBuf ( )

Puts the string back into a reasonable state (in which it can be used normally), after GetWriteBuf() was called.

The version of the function without the *len* parameter will calculate the new string length itself assuming that the string is terminated by the first `NUL` character in it while the second one will use the specified length and thus is the only version which should be used with the strings with embedded `NUL`s (it is also slightly more efficient as `strlen()` doesn't have to be called).

This method is deprecated, please use wxStringBuffer or wxStringBufferLength instead.

void wxString::UngetWriteBuf (size_t *len*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxString wxString::Upper ( ) const

Returns this string converted to upper case.

See also

MakeUpper()

void wxString::UpperCase ( )

The same as MakeUpper().

This is a wxWidgets 1.xx compatibility function; you should not use it in new code.

const wxScopedCharBuffer wxString::utf8_str ( ) const

Converts the strings contents to UTF-8 and returns it either as a temporary wxCharBuffer object or as a pointer to the internal string contents in UTF-8 build.

See also

wc_str(), c_str(), mb_str()

const wchar_t* wxString::wc_str ( ) const

Converts the strings contents to the wide character representation and returns it as a temporary wxWCharBuffer object (Unix and OS X) or returns a pointer to the internal string contents in wide character mode (Windows).

The macro wxWX2WCbuf is defined as the correct return type (without const).

See also

utf8_str(), c_str(), mb_str(), fn_str(), wchar_str()

const wxWCharBuffer wxString::wc_str ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxWritableWCharBuffer wxString::wchar_str ( ) const

Returns an object with string data that is implicitly convertible to `char*` pointer.

Note that changes to the returned buffer may or may not be lost (depending on the build) and so this function is only usable for passing strings to legacy libraries that don't have const-correct API. Use wxStringBuffer if you want to modify the string.

See also

mb_str(), wc_str(), fn_str(), c_str(), char_str()

const wxStringCharType* wxString::wx_str ( ) const

Explicit conversion to C string in the internal representation (either wchar_t* or UTF-8-encoded char*, depending on the build).

21.730.10 Member Data Documentation

const size_t wxString::npos `[static]`

An 'invalid' value for string index.

21.731 wxStringBuffer Class Reference

`#include <wx/string.h>`

21.731.1 Detailed Description

This tiny class allows you to conveniently access the wxString internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later.

For example, assuming you have a low-level OS function called "`GetMeaningOfLifeAsString(char *)`" returning the value in the provided buffer (which must be writable, of course) you might call it like this:

```
wxString theAnswer;
GetMeaningOfLifeAsString(wxStringBuffer(theAnswer, 1024));
if ( theAnswer != "42" )
    wxLogError("Something is very wrong!");
```

Note that the exact usage of this depends on whether or not `wxUSE_STL` is enabled. If `wxUSE_STL` is enabled, wxStringBuffer creates a separate empty character buffer, and if `wxUSE_STL` is disabled, it uses GetWriteBuf() from wxString, keeping the same buffer wxString uses intact. In other words, relying on wxStringBuffer containing the old wxString data is not a good idea if you want to build your program both with and without `wxUSE_STL`.

Library: wxBase

Category: Data Structures

Public Member Functions

- wxStringBuffer (const wxString &str, size_t len)

  Constructs a writable string buffer object associated with the given string and containing enough space for at least len characters.
- ~wxStringBuffer ()

  Restores the string passed to the constructor to the usable state by calling wxString::UngetWriteBuf() on it.
- wxStringCharType * operator wxStringCharType * ()

  Returns the writable pointer to a buffer of the size at least equal to the length specified in the constructor.

21.731.2 Constructor & Destructor Documentation

wxStringBuffer::wxStringBuffer (const wxString & *str,* size_t *len*)

Constructs a writable string buffer object associated with the given string and containing enough space for at least *len* characters.

Basically, this is equivalent to calling wxString::GetWriteBuf() and saving the result.

wxStringBuffer::~wxStringBuffer ( )

Restores the string passed to the constructor to the usable state by calling wxString::UngetWriteBuf() on it.

21.731.3 Member Function Documentation

wxStringCharType* wxStringBuffer::operator wxStringCharType * ( )

Returns the writable pointer to a buffer of the size at least equal to the length specified in the constructor.

21.732 wxStringBufferLength Class Reference

`#include <wx/string.h>`

21.732.1 Detailed Description

This tiny class allows you to conveniently access the wxString internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later, and allows the user to set the internal length of the string.

For example, assuming you have a low-level OS function called "`int GetMeaningOfLifeAsString(char *)`" copying the value in the provided buffer (which must be writable, of course), and returning the actual length of the string, you might call it like this:

```
wxString theAnswer;
wxStringBufferLength theAnswerBuffer(theAnswer, 1024);
int nLength = GetMeaningOfLifeAsString(theAnswerBuffer);
theAnswerBuffer.SetLength(nLength);
if ( theAnswer != "42" )
    wxLogError("Something is very wrong!");
```

Note that the exact usage of this depends on whether or not wxUSE_STL is enabled. If wxUSE_STL is enabled, wxStringBuffer creates a separate empty character buffer, and if wxUSE_STL is disabled, it uses GetWriteBuf() from wxString, keeping the same buffer wxString uses intact. In other words, relying on wxStringBuffer containing the old wxString data is not a good idea if you want to build your program both with and without wxUSE_STL.

Note that wxStringBuffer::SetLength **must** be called before wxStringBufferLength destructs.

Library: wxBase

Category: Data Structures

Public Member Functions

- wxStringBufferLength (const wxString &str, size_t len)

  Constructs a writable string buffer object associated with the given string and containing enough space for at least len characters.
- ~wxStringBufferLength ()

  Restores the string passed to the constructor to the usable state by calling wxString::UngetWriteBuf on it.
- void SetLength (size_t nLength)

  Sets the internal length of the string referred to by wxStringBufferLength to nLength characters.
- wxChar * operator wxChar * ()

  Returns the writable pointer to a buffer of the size at least equal to the length specified in the constructor.

21.732.2 Constructor & Destructor Documentation

wxStringBufferLength::wxStringBufferLength (const wxString & *str,* size_t *len*)

Constructs a writable string buffer object associated with the given string and containing enough space for at least *len* characters.

Basically, this is equivalent to calling wxString::GetWriteBuf and saving the result.

wxStringBufferLength::~wxStringBufferLength ( )

Restores the string passed to the constructor to the usable state by calling wxString::UngetWriteBuf on it.

21.732.3 Member Function Documentation

wxChar* wxStringBufferLength::operator wxChar * ( )

Returns the writable pointer to a buffer of the size at least equal to the length specified in the constructor.

void wxStringBufferLength::SetLength (size_t *nLength*)

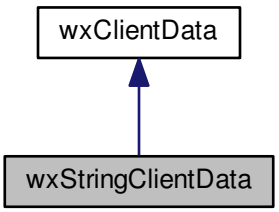
Sets the internal length of the string referred to by wxStringBufferLength to *nLength* characters.

Must be called before wxStringBufferLength destructs.

21.733 wxStringClientData Class Reference

`#include <wx/clntdata.h>`

Inheritance diagram for wxStringClientData:



21.733.1 Detailed Description

Predefined client data class for holding a string.

Library: wxBase

Category: Containers

Public Member Functions

- wxStringClientData ()

  Default constructor.

- wxStringClientData (const wxString &data)

  Create client data with string.

- const wxString & GetData () const

  Get string client data.

- void SetData (const wxString &data)

  Set string client data.

21.733.2 Constructor & Destructor Documentation

wxStringClientData::wxStringClientData ( )

Default constructor.

wxStringClientData::wxStringClientData (const wxString & *data*)

Create client data with string.

21.733.3 Member Function Documentation

const wxString& wxStringClientData::GetData ( ) const

Get string client data.

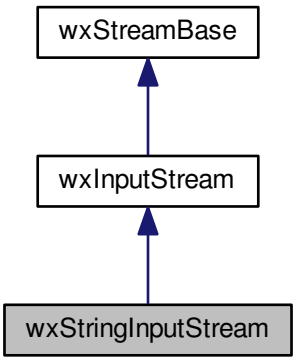
void wxStringClientData::SetData (const wxString & *data*)

Set string client data.

21.734 wxStringInputStream Class Reference

```
#include <wx/sstream.h>
```

Inheritance diagram for wxStringInputStream:



21.734.1 Detailed Description

This class implements an input stream which reads data from a string.

It supports seeking.

Library: wxBase

Category: Streams

Public Member Functions

- wxStringInputStream (const wxString &s)

  Creates a new read-only stream using the specified string.

Additional Inherited Members

21.734.2 Constructor & Destructor Documentation

wxStringInputStream::wxStringInputStream (const wxString & *s*)

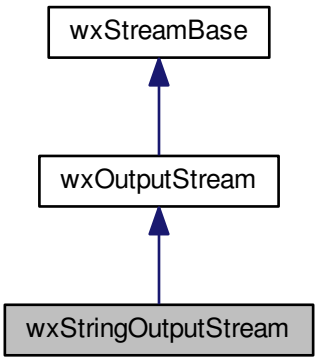
Creates a new read-only stream using the specified string.

Note that the string is copied by the stream so if the original string is modified after using this constructor, changes to it are not reflected when reading from stream.

21.735 wxStringOutputStream Class Reference

```
#include <wx/sstream.h>
```

Inheritance diagram for wxStringOutputStream:



21.735.1 Detailed Description

This class implements an output stream which writes data either to a user-provided or internally allocated string.

Note that currently this stream does not support seeking but can tell its current position.

Library: wxBase

Category: Streams

Public Member Functions

- wxStringOutputStream (wxString *pString=0, wxMBConv &conv=wxConvUTF8)

  Construct a new stream object writing the data to a string.
- const wxString & GetString () const

  Returns the string containing all the data written to the stream so far.

Additional Inherited Members

21.735.2 Constructor & Destructor Documentation

wxStringOutputStream::wxStringOutputStream (wxString * *pString* = `0`, wxMBConv & *conv* = `wxConvUTF8`)

Construct a new stream object writing the data to a string.

If the provided pointer is non-NULL, data will be written to it. Otherwise, an internal string is used for the data written to this stream, use GetString() to get access to it.

If *str* is used, data written to the stream is appended to the current contents of it, i.e. the string is not cleared here. However if it is not empty, the positions returned by wxOutputStream::TellO will be offset by the initial string length, i.e. initial stream position will be the initial length of the string and not 0.

Notice that the life time of *conv* must be greater than the life time of this object itself as it stores a reference to it. Also notice that with default value of this argument the data written to the stream must be valid UTF-8, pass `wxConvISO8859_1` to deal with arbitrary 8 bit data.

21.735.3 Member Function Documentation

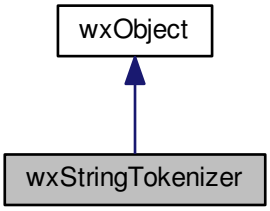
const wxString& wxStringOutputStream::GetString ( ) const

Returns the string containing all the data written to the stream so far.

21.736 wxStringTokenizer Class Reference

`#include <wx/tokenzr.h>`

Inheritance diagram for wxStringTokenizer:



21.736.1 Detailed Description

wxStringTokenizer helps you to break a string up into a number of tokens.

It replaces the standard C function `strtok()` and also extends it in a number of ways.

To use this class, you should create a wxStringTokenizer object, give it the string to tokenize and also the delimiters which separate tokens in the string (by default, white space characters will be used).

Then wxStringTokenizer::GetNextToken() may be called repeatedly until wxStringTokenizer::HasMoreTokens() returns false.

For example:

```
wxStringTokenizer tokenizer("first:second:third:fourth", ":");
while ( tokenizer.HasMoreTokens() )
{
    wxString token = tokenizer.GetNextToken();

    // process token here
}
```

Library: wxBase

Category: Data Structures

See also

wxStringTokenize()

Public Member Functions

- wxStringTokenizer ()

  Default constructor.
- wxStringTokenizer (const wxString &str, const wxString &delims=wxDEFAULT_DELIMITERS, wxStringTokenizerMode mode=wxTOKEN_DEFAULT)

  Constructor.
- size_t CountTokens () const

  Returns the number of tokens remaining in the input string.
- wxChar GetLastDelimiter () const

  Returns the delimiter which ended scan for the last token returned by GetNextToken() or `NUL` if there had been no calls to this function yet or if it returned the trailing empty token in wxTOKEN_RET_EMPTY_ALL mode.
- wxString GetNextToken ()

  Returns the next token or empty string if the end of string was reached.
- size_t GetPosition () const

  Returns the current position (i.e. one index after the last returned token or 0 if GetNextToken() has never been called) in the original string.
- wxString GetString () const

  Returns the part of the starting string without all token already extracted.
- bool HasMoreTokens () const

  Returns true if the tokenizer has further tokens, false if none are left.
- void SetString (const wxString &str, const wxString &delims=wxDEFAULT_DELIMITERS, wxStringTokenizerMode mode=wxTOKEN_DEFAULT)

  Initializes the tokenizer.

Additional Inherited Members

21.736.2 Constructor & Destructor Documentation

wxStringTokenizer::wxStringTokenizer ( )

Default constructor.

You must call SetString() before calling any other methods.

wxStringTokenizer::wxStringTokenizer (const wxString & *str*, const wxString & *delims* = wxDEFAULT_DELIMITERS, wxStringTokenizerMode *mode* = wxTOKEN_DEFAULT)

Constructor.

Pass the string to tokenize, a string containing delimiters, and the *mode* specifying how the string should be tokenized.

See also

SetString()

21.736.3 Member Function Documentation

size_t wxStringTokenizer::CountTokens ( ) const

Returns the number of tokens remaining in the input string.

The number of tokens returned by this function is decremented each time GetNextToken() is called and when it reaches 0, HasMoreTokens() returns false.

wxChar wxStringTokenizer::GetLastDelimiter ( ) const

Returns the delimiter which ended scan for the last token returned by GetNextToken() or `NUL` if there had been no calls to this function yet or if it returned the trailing empty token in wxTOKEN_RET_EMPTY_ALL mode.

Since

2.7.0

wxString wxStringTokenizer::GetNextToken ( )

Returns the next token or empty string if the end of string was reached.

size_t wxStringTokenizer::GetPosition ( ) const

Returns the current position (i.e. one index after the last returned token or 0 if GetNextToken() has never been called) in the original string.

wxString wxStringTokenizer::GetString ( ) const

Returns the part of the starting string without all token already extracted.

bool wxStringTokenizer::HasMoreTokens ( ) const

Returns true if the tokenizer has further tokens, false if none are left.

void wxStringTokenizer::SetString (const wxString & *str*, const wxString & *delims* = wxDEFAULT_DELIMITERS, wxStringTokenizerMode *mode* = wxTOKEN_DEFAULT)

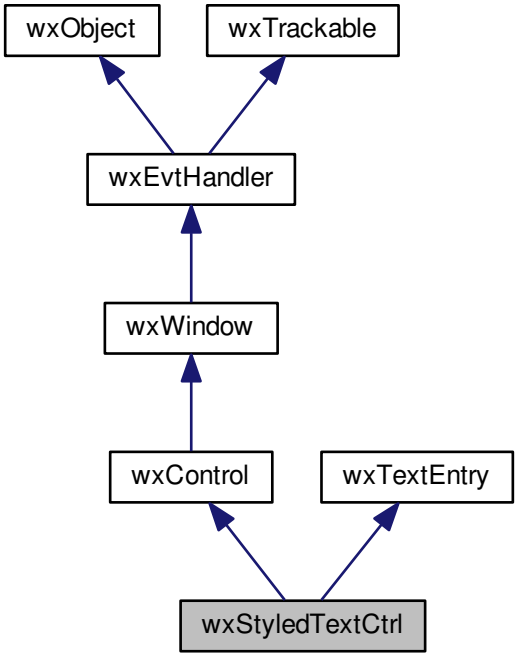
Initializes the tokenizer.

Pass the string to tokenize, a string containing delimiters, and the *mode* specifying how the string should be tokenized.

21.737 wxStyledTextCtrl Class Reference

`#include <wx/stc/stc.h>`

Inheritance diagram for wxStyledTextCtrl:



21.737.1 Detailed Description

A wxWidgets implementation of the Scintilla source code editing component.

As well as features found in standard text editing components, Scintilla includes features especially useful when editing and debugging source code. These include support for syntax styling, error indicators, code completion and call tips.

The selection margin can contain markers like those used in debuggers to indicate breakpoints and the current line. Styling choices are more open than with many editors, allowing the use of proportional fonts, bold and italics, multiple foreground and background colours and multiple fonts.

wxStyledTextCtrl is a 1 to 1 mapping of "raw" scintilla interface, whose documentation can be found in the Scintilla website (http://www.scintilla.org/).

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxStyledTextEvent& event)

Event macros for events emitted by this class:

- EVT_STC_CHANGE(id, fn): TOWRITE
- EVT_STC_STYLENEEDED(id, fn): TOWRITE
- EVT_STC_CHARADDED(id, fn): TOWRITE
- EVT_STC_SAVEPOINTREACHED(id, fn): TOWRITE

- EVT_STC_SAVEPOINTLEFT(id, fn): TOWRITE
- EVT_STC_ROMODIFYATTEMPT(id, fn): TOWRITE
- EVT_STC_KEY(id, fn): TOWRITE
- EVT_STC_DOUBLECLICK(id, fn): TOWRITE
- EVT_STC_UPDATEUI(id, fn): TOWRITE
- EVT_STC_MODIFIED(id, fn): TOWRITE
- EVT_STC_MACRORECORD(id, fn): TOWRITE
- EVT_STC_MARGINCLICK(id, fn): TOWRITE
- EVT_STC_NEEDSHOWN(id, fn): TOWRITE
- EVT_STC_PAINTED(id, fn): TOWRITE
- EVT_STC_USERLISTSELECTION(id, fn): TOWRITE
- EVT_STC_URIDROPPED(id, fn): TOWRITE
- EVT_STC_DWELLSTART(id, fn): TOWRITE
- EVT_STC_DWELLEND(id, fn): TOWRITE
- EVT_STC_START_DRAG(id, fn): Process a `wxEVT_STC_START_DRAG` event, generated when text is being dragged from the control. Details of the drag may be altered by changing the respective fields of the event; in particular, set an empty string to prohibit the drag entirely. Valid event functions: GetDragFlags, SetDragFlags, GetPosition, GetString, SetString.
- EVT_STC_DRAG_OVER(id, fn): TOWRITE
- EVT_STC_DO_DROP(id, fn): Process a `wxEVT_STC_DO_DROP` event, generated when text is being dropped into the control. Details of the drag may be altered by changing the respective fields of the event. Valid event functions: GetDragResult, SetDragResult, GetPosition, SetPosition, GetString, SetString, GetX, GetY.
- EVT_STC_ZOOM(id, fn): TOWRITE
- EVT_STC_HOTSPOT_CLICK(id, fn): TOWRITE
- EVT_STC_HOTSPOT_DCLICK(id, fn): TOWRITE
- EVT_STC_CALLTIP_CLICK(id, fn): TOWRITE
- EVT_STC_AUTOCOMP_SELECTION(id, fn): TOWRITE
- EVT_STC_INDICATOR_CLICK(id, fn): TOWRITE
- EVT_STC_INDICATOR_RELEASE(id, fn): TOWRITE
- EVT_STC_AUTOCOMP_CANCELLED(id, fn): TOWRITE
- EVT_STC_AUTOCOMP_CHAR_DELETED(id, fn): TOWRITE
- EVT_STC_HOTSPOT_RELEASE_CLICK(id, fn): TOWRITE
- EVT_STC_CLIPBOARD_COPY(id, fn): Process a `wxEVT_STC_CLIPBOARD_COPY` event, generated when text is being cut or copied to the clipboard. Use wxStyledTextEvent::SetString() to modify the text that will be placed on the clipboard. Valid event functions: GetString, SetString.

Since

3.1.0

- EVT_STC_CLIPBOARD_PASTE(id, fn): Process a `wxEVT_STC_CLIPBOARD_PASTE` event, generated when text is being pasted from the clipboard. Use wxStyledTextEvent::SetString() to modify the text that will be inserted into the control. Valid event functions: GetPosition, GetString, SetString.

Since

3.1.0

Library: wxSTC

Category: Scintilla Text Editor

See also

wxStyledTextEvent

Public Member Functions

- wxStyledTextCtrl (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxSTCNameStr)

  Ctor.
- wxStyledTextCtrl ()

  Default ctor.
- ~wxStyledTextCtrl ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxSTCNameStr)

  Create the UI elements for a STC that was created with the default ctor.
- void AddText (const wxString &text)

  Add text to the document at current position.
- void AddStyledText (const wxMemoryBuffer &data)

  Add array of cells to document.
- void InsertText (int pos, const wxString &text)

  Insert string at a position.
- void ClearAll ()

  Delete all text in the document.
- void DeleteRange (int pos, int deleteLength)

  Delete a range of text in the document.
- void ClearDocumentStyle ()

  Set all style bytes to 0, remove all folding information.
- int GetLength () const

  Returns the number of bytes in the document.
- int GetCharAt (int pos) const

  Returns the character byte at the position.
- int GetCurrentPos () const

  Returns the position of the caret.
- int GetAnchor () const

  Returns the position of the opposite end of the selection to the caret.
- int GetStyleAt (int pos) const

Returns the style byte at the position.

- void Redo ()

  Redoes the next action on the undo history.

- void SetUndoCollection (bool collectUndo)

  Choose between collecting actions into the undo history and discarding them.

- void SelectAll ()

  Select all the text in the document.

- void SetSavePoint ()

  Remember the current position in the undo history as the position at which the document was saved.

- wxMemoryBuffer GetStyledText (int startPos, int endPos)

  Retrieve a buffer of cells.

- bool CanRedo () const

  Are there any redoable actions in the undo history?

- int MarkerLineFromHandle (int handle)

  Retrieve the line number at which a particular marker is located.

- void MarkerDeleteHandle (int handle)

  Delete a marker.

- bool GetUndoCollection () const

  Is undo history being collected?

- int GetViewWhiteSpace () const

  Are white space characters currently visible? Returns one of SCWS_∗ constants.

- void SetViewWhiteSpace (int viewWS)

  Make white space characters invisible, always visible or visible outside indentation.

- int PositionFromPoint (wxPoint pt) const

  Find the position from a point within the window.

- int PositionFromPointClose (int x, int y)

  Find the position from a point within the window but return INVALID_POSITION if not close to text.

- void GotoLine (int line)

  Set caret to start of a line and ensure it is visible.

- void GotoPos (int pos)

  Set caret to a position and ensure it is visible.

- void SetAnchor (int posAnchor)

  Set the selection anchor to a position.

- wxString GetCurLine (int ∗linePos=NULL)

  Retrieve the text of the line containing the caret.

- int GetEndStyled () const

  Retrieve the position of the last correctly styled character.

- void ConvertEOLs (int eolMode)

  Convert all line endings in the document to one mode.

- int GetEOLMode () const

  Retrieve the current end of line mode - one of CRLF, CR, or LF.

- void SetEOLMode (int eolMode)

  Set the current end of line mode.

- void StartStyling (int pos, int mask)

  Set the current styling position to pos and the styling mask to mask.

- void SetStyling (int length, int style)

  Change style from current styling position for length characters to a style and move the current styling position to after this newly styled segment.

- bool GetBufferedDraw () const

  Is drawing done first into a buffer or direct to the screen?

- void SetBufferedDraw (bool buffered)

If drawing is buffered then each line of text is drawn into a bitmap buffer before drawing it to the screen to avoid flicker.

- void SetTabWidth (int tabWidth)

  Change the visible size of a tab to be a multiple of the width of a space character.
- int GetTabWidth () const

  Retrieve the visible size of a tab.
- void SetCodePage (int codePage)

  Set the code page used to interpret the bytes of the document as characters.
- void MarkerDefine (int markerNumber, int markerSymbol, const wxColour &foreground=wxNullColour, const wxColour &background=wxNullColour)

  Set the symbol used for a particular marker number, and optionally the fore and background colours.
- void MarkerSetForeground (int markerNumber, const wxColour &fore)

  Set the foreground colour used for a particular marker number.
- void MarkerSetBackground (int markerNumber, const wxColour &back)

  Set the background colour used for a particular marker number.
- void MarkerSetBackgroundSelected (int markerNumber, const wxColour &back)

  Set the background colour used for a particular marker number when its folding block is selected.
- void MarkerEnableHighlight (bool enabled)

  Enable/disable highlight for current folding bloc (smallest one that contains the caret)
- int MarkerAdd (int line, int markerNumber)

  Add a marker to a line, returning an ID which can be used to find or delete the marker.
- void MarkerDelete (int line, int markerNumber)

  Delete a marker from a line.
- void MarkerDeleteAll (int markerNumber)

  Delete all markers with a particular number from all lines.
- int MarkerGet (int line)

  Get a bit mask of all the markers set on a line.
- int MarkerNext (int lineStart, int markerMask)

  Find the next line at or after lineStart that includes a marker in mask.
- int MarkerPrevious (int lineStart, int markerMask)

  Find the previous line before lineStart that includes a marker in mask.
- void MarkerDefineBitmap (int markerNumber, const wxBitmap &bmp)

  Define a marker from a bitmap.
- void MarkerAddSet (int line, int set)

  Add a set of markers to a line.
- void MarkerSetAlpha (int markerNumber, int alpha)

  Set the alpha used for a marker that is drawn in the text area, not the margin.
- void SetMarginType (int margin, int marginType)

  Set a margin to be either numeric or symbolic.
- int GetMarginType (int margin) const

  Retrieve the type of a margin.
- void SetMarginWidth (int margin, int pixelWidth)

  Set the width of a margin to a width expressed in pixels.
- int GetMarginWidth (int margin) const

  Retrieve the width of a margin in pixels.
- void SetMarginMask (int margin, int mask)

  Set a mask that determines which markers are displayed in a margin.
- int GetMarginMask (int margin) const

  Retrieve the marker mask of a margin.
- void SetMarginSensitive (int margin, bool sensitive)

  Make a margin sensitive or insensitive to mouse clicks.
- bool GetMarginSensitive (int margin) const

Retrieve the mouse click sensitivity of a margin.

- void SetMarginCursor (int margin, int cursor)

  Set the cursor shown when the mouse is inside a margin.
- int GetMarginCursor (int margin) const

  Retrieve the cursor shown in a margin.
- void StyleClearAll ()

  Clear all the styles and make equivalent to the global default style.
- void StyleSetForeground (int style, const wxColour &fore)

  Set the foreground colour of a style.
- void StyleSetBackground (int style, const wxColour &back)

  Set the background colour of a style.
- void StyleSetBold (int style, bool bold)

  Set a style to be bold or not.
- void StyleSetItalic (int style, bool italic)

  Set a style to be italic or not.
- void StyleSetSize (int style, int sizePoints)

  Set the size of characters of a style.
- void StyleSetFaceName (int style, const wxString &fontName)

  Set the font of a style.
- void StyleSetEOLFilled (int style, bool filled)

  Set a style to have its end of line filled or not.
- void StyleResetDefault ()

  Reset the default style to its state at startup.
- void StyleSetUnderline (int style, bool underline)

  Set a style to be underlined or not.
- wxColour StyleGetForeground (int style) const

  Get the foreground colour of a style.
- wxColour StyleGetBackground (int style) const

  Get the background colour of a style.
- bool StyleGetBold (int style) const

  Get is a style bold or not.
- bool StyleGetItalic (int style) const

  Get is a style italic or not.
- int StyleGetSize (int style) const

  Get the size of characters of a style.
- wxString StyleGetFaceName (int style)

  Get the font facename of a style.
- bool StyleGetEOLFilled (int style) const

  Get is a style to have its end of line filled or not.
- bool StyleGetUnderline (int style) const

  Get is a style underlined or not.
- int StyleGetCase (int style) const

  Get is a style mixed case, or to force upper or lower case.
- int StyleGetCharacterSet (int style) const

  Get the character get of the font in a style.
- bool StyleGetVisible (int style) const

  Get is a style visible or not.
- bool StyleGetChangeable (int style) const

  Get is a style changeable or not (read only).
- bool StyleGetHotSpot (int style) const

  Get is a style a hotspot or not.

- void StyleSetCase (int style, int caseForce)

  Set a style to be mixed case, or to force upper or lower case.
- void StyleSetSizeFractional (int style, int caseForce)

  Set the size of characters of a style.
- int StyleGetSizeFractional (int style) const

  Get the size of characters of a style in points multiplied by 100.
- void StyleSetWeight (int style, int weight)

  Set the weight of characters of a style.
- int StyleGetWeight (int style) const

  Get the weight of characters of a style.
- void StyleSetHotSpot (int style, bool hotspot)

  Set a style to be a hotspot or not.
- void SetSelForeground (bool useSetting, const wxColour &fore)

  Set the foreground colour of the main and additional selections and whether to use this setting.
- void SetSelBackground (bool useSetting, const wxColour &back)

  Set the background colour of the main and additional selections and whether to use this setting.
- int GetSelAlpha () const

  Get the alpha of the selection.
- void SetSelAlpha (int alpha)

  Set the alpha of the selection.
- bool GetSelEOLFilled () const

  Is the selection end of line filled?
- void SetSelEOLFilled (bool filled)

  Set the selection to have its end of line filled or not.
- void SetCaretForeground (const wxColour &fore)

  Set the foreground colour of the caret.
- void CmdKeyAssign (int key, int modifiers, int cmd)

  When key+modifier combination km is pressed perform msg.
- void CmdKeyClear (int key, int modifiers)

  When key+modifier combination km is pressed do nothing.
- void CmdKeyClearAll ()

  Drop all key mappings.
- void SetStyleBytes (int length, char ∗styleBytes)

  Set the styles for a segment of the document.
- void StyleSetVisible (int style, bool visible)

  Set a style to be visible or not.
- int GetCaretPeriod () const

  Get the time in milliseconds that the caret is on and off.
- void SetCaretPeriod (int periodMilliseconds)

  Get the time in milliseconds that the caret is on and off.
- void SetWordChars (const wxString &characters)

  Set the set of characters making up words for when moving or selecting by word.
- wxString GetWordChars () const

  Get the set of characters making up words for when moving or selecting by word.
- void BeginUndoAction ()

  Start a sequence of actions that is undone and redone as a unit.
- void EndUndoAction ()

  End a sequence of actions that is undone and redone as a unit.
- void IndicatorSetStyle (int indic, int style)

  Set an indicator to plain, squiggle or TT.
- int IndicatorGetStyle (int indic) const

Retrieve the style of an indicator.

- void IndicatorSetForeground (int indic, const wxColour &fore)

  Set the foreground colour of an indicator.
- wxColour IndicatorGetForeground (int indic) const

  Retrieve the foreground colour of an indicator.
- void IndicatorSetUnder (int indic, bool under)

  Set an indicator to draw under text or over(default).
- bool IndicatorGetUnder (int indic) const

  Retrieve whether indicator drawn under or over text.
- void SetWhitespaceForeground (bool useSetting, const wxColour &fore)

  Set the foreground colour of all whitespace and whether to use this setting.
- void SetWhitespaceBackground (bool useSetting, const wxColour &back)

  Set the background colour of all whitespace and whether to use this setting.
- void SetWhitespaceSize (int size)

  Set the size of the dots used to mark space characters.
- int GetWhitespaceSize () const

  Get the size of the dots used to mark space characters.
- void SetStyleBits (int bits)

  Divide each styling byte into lexical class bits (default: 5) and indicator bits (default: 3).
- int GetStyleBits () const

  Retrieve number of bits in style bytes used to hold the lexical state.
- void SetLineState (int line, int state)

  Used to hold extra styling information for each line.
- int GetLineState (int line) const

  Retrieve the extra styling information for a line.
- int GetMaxLineState () const

  Retrieve the last line number that has line state.
- bool GetCaretLineVisible () const

  Is the background of the line containing the caret in a different colour?
- void SetCaretLineVisible (bool show)

  Display the background of the line containing the caret in a different colour.
- wxColour GetCaretLineBackground () const

  Get the colour of the background of the line containing the caret.
- void SetCaretLineBackground (const wxColour &back)

  Set the colour of the background of the line containing the caret.
- void StyleSetChangeable (int style, bool changeable)

  Set a style to be changeable or not (read only).
- void AutoCompShow (int lenEntered, const wxString &itemList)

  Display a auto-completion list.
- void AutoCompCancel ()

  Remove the auto-completion list from the screen.
- bool AutoCompActive ()

  Is there an auto-completion list visible?
- int AutoCompPosStart ()

  Retrieve the position of the caret when the auto-completion list was displayed.
- void AutoCompComplete ()

  User has selected an item so remove the list and insert the selection.
- void AutoCompStops (const wxString &characterSet)

  Define a set of character that when typed cancel the auto-completion list.
- void AutoCompSetSeparator (int separatorCharacter)

  Change the separator character in the string setting up an auto-completion list.

- int AutoCompGetSeparator () const

  Retrieve the auto-completion list separator character.
- void AutoCompSelect (const wxString &text)

  Select the item in the auto-completion list that starts with a string.
- void AutoCompSetCancelAtStart (bool cancel)

  Should the auto-completion list be cancelled if the user backspaces to a position before where the box was created.
- bool AutoCompGetCancelAtStart () const

  Retrieve whether auto-completion cancelled by backspacing before start.
- void AutoCompSetFillUps (const wxString &characterSet)

  Define a set of characters that when typed will cause the autocompletion to choose the selected item.
- void AutoCompSetChooseSingle (bool chooseSingle)

  Should a single item auto-completion list automatically choose the item.
- bool AutoCompGetChooseSingle () const

  Retrieve whether a single item auto-completion list automatically choose the item.
- void AutoCompSetIgnoreCase (bool ignoreCase)

  Set whether case is significant when performing auto-completion searches.
- bool AutoCompGetIgnoreCase () const

  Retrieve state of ignore case flag.
- void UserListShow (int listType, const wxString &itemList)

  Display a list of strings and send notification when user chooses one.
- void AutoCompSetAutoHide (bool autoHide)

  Set whether or not autocompletion is hidden automatically when nothing matches.
- bool AutoCompGetAutoHide () const

  Retrieve whether or not autocompletion is hidden automatically when nothing matches.
- void AutoCompSetDropRestOfWord (bool dropRestOfWord)

  Set whether or not autocompletion deletes any word characters after the inserted text upon completion.
- bool AutoCompGetDropRestOfWord () const

  Retrieve whether or not autocompletion deletes any word characters after the inserted text upon completion.
- void RegisterImage (int type, const wxBitmap &bmp)

  Register an image for use in autocompletion lists.
- void ClearRegisteredImages ()

  Clear all the registered images.
- int AutoCompGetTypeSeparator () const

  Retrieve the auto-completion list type-separator character.
- void AutoCompSetTypeSeparator (int separatorCharacter)

  Change the type-separator character in the string setting up an auto-completion list.
- void AutoCompSetMaxWidth (int characterCount)

  Set the maximum width, in characters, of auto-completion and user lists.
- int AutoCompGetMaxWidth () const

  Get the maximum width, in characters, of auto-completion and user lists.
- void AutoCompSetMaxHeight (int rowCount)

  Set the maximum height, in rows, of auto-completion and user lists.
- int AutoCompGetMaxHeight () const

  Set the maximum height, in rows, of auto-completion and user lists.
- void SetIndent (int indentSize)

  Set the number of spaces used for one level of indentation.
- int GetIndent () const

  Retrieve indentation size.
- void SetUseTabs (bool useTabs)

  Indentation will only use space characters if useTabs is false, otherwise it will use a combination of tabs and spaces.
- bool GetUseTabs () const

Retrieve whether tabs will be used in indentation.

- void SetLineIndentation (int line, int indentSize)

  Change the indentation of a line to a number of columns.
- int GetLineIndentation (int line) const

  Retrieve the number of columns that a line is indented.
- int GetLineIndentPosition (int line) const

  Retrieve the position before the first non indentation character on a line.
- int GetColumn (int pos) const

  Retrieve the column number of a position, taking tab width into account.
- int CountCharacters (int startPos, int endPos)

  Count characters between two positions.
- void SetUseHorizontalScrollBar (bool show)

  Show or hide the horizontal scroll bar.
- bool GetUseHorizontalScrollBar () const

  Is the horizontal scroll bar visible?
- void SetIndentationGuides (int indentView)

  Show or hide indentation guides.
- int GetIndentationGuides () const

  Are the indentation guides visible?
- void SetHighlightGuide (int column)

  Set the highlighted indentation guide column.
- int GetHighlightGuide () const

  Get the highlighted indentation guide column.
- int GetLineEndPosition (int line) const

  Get the position after the last visible characters on a line.
- int GetCodePage () const

  Get the code page used to interpret the bytes of the document as characters.
- wxColour GetCaretForeground () const

  Get the foreground colour of the caret.
- bool GetReadOnly () const

  In read-only mode?
- void SetCurrentPos (int pos)

  Sets the position of the caret.
- void SetSelectionStart (int pos)

  Sets the position that starts the selection - this becomes the anchor.
- int GetSelectionStart () const

  Returns the position at the start of the selection.
- void SetSelectionEnd (int pos)

  Sets the position that ends the selection - this becomes the currentPosition.
- int GetSelectionEnd () const

  Returns the position at the end of the selection.
- void SetEmptySelection (int pos)

  Set caret to a position, while removing any existing selection.
- void SetPrintMagnification (int magnification)

  Sets the print magnification added to the point size of each style for printing.
- int GetPrintMagnification () const

  Returns the print magnification.
- void SetPrintColourMode (int mode)

  Modify colours when printing for clearer printed text.
- int GetPrintColourMode () const

  Returns the print colour mode.

- int FindText (int minPos, int maxPos, const wxString &text, int flags=0)

  Find some text in the document.
- int FormatRange (bool doDraw, int startPos, int endPos, wxDC *draw, wxDC *target, wxRect renderRect, wxRect pageRect)

  On Windows, will draw the document into a display context such as a printer.
- int GetFirstVisibleLine () const

  Retrieve the display line at the top of the display.
- wxString GetLine (int line) const

  Retrieve the contents of a line.
- int GetLineCount () const

  Returns the number of lines in the document.
- void SetMarginLeft (int pixelWidth)

  Sets the size in pixels of the left margin.
- int GetMarginLeft () const

  Returns the size in pixels of the left margin.
- void SetMarginRight (int pixelWidth)

  Sets the size in pixels of the right margin.
- int GetMarginRight () const

  Returns the size in pixels of the right margin.
- bool GetModify () const

  Is the document different from when it was last saved?
- wxString GetSelectedText ()

  Retrieve the selected text.
- wxString GetTextRange (int startPos, int endPos)

  Retrieve a range of text.
- void HideSelection (bool normal)

  Draw the selection in normal style or with selection highlighted.
- int LineFromPosition (int pos) const

  Retrieve the line containing a position.
- int PositionFromLine (int line) const

  Retrieve the position at the start of a line.
- void LineScroll (int columns, int lines)

  Scroll horizontally and vertically.
- void EnsureCaretVisible ()

  Ensure the caret is visible.
- void ScrollRange (int secondary, int primary)

  Scroll the argument positions and the range between them into view giving priority to the primary position then the secondary position.
- void ReplaceSelection (const wxString &text)

  Replace the selected text with the argument text.
- void SetReadOnly (bool readOnly)

  Set to read only or read write.
- bool CanPaste () const

  Will a paste succeed?
- bool CanUndo () const

  Are there any undoable actions in the undo history?
- void EmptyUndoBuffer ()

  Delete the undo history.
- void Undo ()

  Undo one action in the undo history.
- void Cut ()

Cut the selection to the clipboard.

- void Copy ()

  Copy the selection to the clipboard.

- void Paste ()

  Paste the contents of the clipboard into the document replacing the selection.

- void Clear ()

  Clear the selection.

- void SetText (const wxString &text)

  Replace the contents of the document with the argument text.

- wxString GetText () const

  Retrieve all the text in the document.

- int GetTextLength () const

  Retrieve the number of characters in the document.

- void SetOvertype (bool overtype)

  Set to overtype (true) or insert mode.

- bool GetOvertype () const

  Returns true if overtype mode is active otherwise false is returned.

- void SetCaretWidth (int pixelWidth)

  Set the width of the insert mode caret.

- int GetCaretWidth () const

  Returns the width of the insert mode caret.

- void SetTargetStart (int pos)

  Sets the position that starts the target which is used for updating the document without affecting the scroll position.

- int GetTargetStart () const

  Get the position that starts the target.

- void SetTargetEnd (int pos)

  Sets the position that ends the target which is used for updating the document without affecting the scroll position.

- int GetTargetEnd () const

  Get the position that ends the target.

- int ReplaceTarget (const wxString &text)

  Replace the target text with the argument text.

- int ReplaceTargetRE (const wxString &text)

  Replace the target text with the argument text after \d processing.

- int SearchInTarget (const wxString &text)

  Search for a counted string in the target and set the target to the found range.

- void SetSearchFlags (int flags)

  Set the search flags used by SearchInTarget.

- int GetSearchFlags () const

  Get the search flags used by SearchInTarget.

- void CallTipShow (int pos, const wxString &definition)

  Show a call tip containing a definition near position pos.

- void CallTipCancel ()

  Remove the call tip from the screen.

- bool CallTipActive ()

  Is there an active call tip?

- int CallTipPosAtStart ()

  Retrieve the position where the caret was before displaying the call tip.

- void CallTipSetPosAtStart (int posStart)

  Set the start position in order to change when backspacing removes the calltip.

- void CallTipSetHighlight (int start, int end)

  Highlight a segment of the definition.

- void CallTipSetBackground (const wxColour &back)

  Set the background colour for the call tip.
- void CallTipSetForeground (const wxColour &fore)

  Set the foreground colour for the call tip.
- void CallTipSetForegroundHighlight (const wxColour &fore)

  Set the foreground colour for the highlighted part of the call tip.
- void CallTipUseStyle (int tabSize)

  Enable use of STYLE_CALLTIP and set call tip tab size in pixels.
- void CallTipSetPosition (bool above)

  Set position of calltip, above or below text.
- int VisibleFromDocLine (int line)

  Find the display line of a document line taking hidden lines into account.
- int DocLineFromVisible (int lineDisplay)

  Find the document line of a display line taking hidden lines into account.
- int WrapCount (int line)

  The number of display lines needed to wrap a document line.
- void SetFoldLevel (int line, int level)

  Set the fold level of a line.
- int GetFoldLevel (int line) const

  Retrieve the fold level of a line.
- int GetLastChild (int line, int level) const

  Find the last child line of a header line.
- int GetFoldParent (int line) const

  Find the parent line of a child line.
- void ShowLines (int lineStart, int lineEnd)

  Make a range of lines visible.
- void HideLines (int lineStart, int lineEnd)

  Make a range of lines invisible.
- bool GetLineVisible (int line) const

  Is a line visible?
- bool GetAllLinesVisible () const

  Are all lines visible?
- void SetFoldExpanded (int line, bool expanded)

  Show the children of a header line.
- bool GetFoldExpanded (int line) const

  Is a header line expanded?
- void ToggleFold (int line)

  Switch a header line between expanded and contracted.
- void FoldLine (int line, int action)

  Expand or contract a fold header.
- void FoldChildren (int line, int action)

  Expand or contract a fold header and its children.
- void ExpandChildren (int line, int level)

  Expand a fold header and all children.
- void FoldAll (int action)

  Expand or contract all fold headers.
- void EnsureVisible (int line)

  Ensure a particular line is visible by expanding any header line hiding it.
- void SetAutomaticFold (int automaticFold)

  Set automatic folding behaviours.
- int GetAutomaticFold () const

Get automatic folding behaviours.

- void SetFoldFlags (int flags)

  Set some style options for folding.
- void EnsureVisibleEnforcePolicy (int line)

  Ensure a particular line is visible by expanding any header line hiding it.
- void SetTabIndents (bool tabIndents)

  Sets whether a tab pressed when caret is within indentation indents.
- bool GetTabIndents () const

  Does a tab pressed when caret is within indentation indent?
- void SetBackSpaceUnIndents (bool bsUnIndents)

  Sets whether a backspace pressed when caret is within indentation unindents.
- bool GetBackSpaceUnIndents () const

  Does a backspace pressed when caret is within indentation unindent?
- void SetMouseDwellTime (int periodMilliseconds)

  Sets the time the mouse must sit still to generate a mouse dwell event.
- int GetMouseDwellTime () const

  Retrieve the time the mouse must sit still to generate a mouse dwell event.
- int WordStartPosition (int pos, bool onlyWordCharacters)

  Get position of start of word.
- int WordEndPosition (int pos, bool onlyWordCharacters)

  Get position of end of word.
- void SetWrapMode (int mode)

  Sets whether text is word wrapped.
- int GetWrapMode () const

  Retrieve whether text is word wrapped.
- void SetWrapVisualFlags (int wrapVisualFlags)

  Set the display mode of visual flags for wrapped lines.
- int GetWrapVisualFlags () const

  Retrive the display mode of visual flags for wrapped lines.
- void SetWrapVisualFlagsLocation (int wrapVisualFlagsLocation)

  Set the location of visual flags for wrapped lines.
- int GetWrapVisualFlagsLocation () const

  Retrive the location of visual flags for wrapped lines.
- void SetWrapStartIndent (int indent)

  Set the start indent for wrapped lines.
- int GetWrapStartIndent () const

  Retrive the start indent for wrapped lines.
- void SetWrapIndentMode (int mode)

  Sets how wrapped sublines are placed.
- int GetWrapIndentMode () const

  Retrieve how wrapped sublines are placed.
- void SetLayoutCache (int mode)

  Sets the degree of caching of layout information.
- int GetLayoutCache () const

  Retrieve the degree of caching of layout information.
- void SetScrollWidth (int pixelWidth)

  Sets the document width assumed for scrolling.
- int GetScrollWidth () const

  Retrieve the document width assumed for scrolling.
- void SetScrollWidthTracking (bool tracking)

  Sets whether the maximum width line displayed is used to set scroll width.

- bool GetScrollWidthTracking () const

  Retrieve whether the scroll width tracks wide lines.
- int TextWidth (int style, const wxString &text)

  Measure the pixel width of some text in a particular style.
- void SetEndAtLastLine (bool endAtLastLine)

  Sets the scroll range so that maximum scroll position has the last line at the bottom of the view (default).
- bool GetEndAtLastLine () const

  Retrieve whether the maximum scroll position has the last line at the bottom of the view.
- int TextHeight (int line)

  Retrieve the height of a particular line of text in pixels.
- void SetUseVerticalScrollBar (bool show)

  Show or hide the vertical scroll bar.
- bool GetUseVerticalScrollBar () const

  Is the vertical scroll bar visible?
- void AppendText (const wxString &text)

  Append a string to the end of the document without changing the selection.
- bool GetTwoPhaseDraw () const

  Is drawing done in two phases with backgrounds drawn before faoregrounds?
- void SetTwoPhaseDraw (bool twoPhase)

  In twoPhaseDraw mode, drawing is performed in two phases, first the background and then the foreground.
- void SetFirstVisibleLine (int lineDisplay)

  Scroll so that a display line is at the top of the display.
- void SetMultiPaste (int multiPaste)

  Change the effect of pasting when there are multiple selections.
- int GetMultiPaste () const

  Retrieve the effect of pasting when there are multiple selections.
- wxString GetTag (int tagNumber) const

  Retrieve the value of a tag from a regular expression search.
- void TargetFromSelection ()

  Make the target range start and end be the same as the selection range start and end.
- void LinesJoin ()

  Join the lines in the target.
- void LinesSplit (int pixelWidth)

  Split the lines in the target into lines that are less wide than pixelWidth where possible.
- void SetFoldMarginColour (bool useSetting, const wxColour &back)

  Set the colours used as a chequerboard pattern in the fold margin.
- void SetFoldMarginHiColour (bool useSetting, const wxColour &fore)
- void LineDown ()

  Move caret down one line.
- void LineDownExtend ()

  Move caret down one line extending selection to new caret position.
- void LineUp ()

  Move caret up one line.
- void LineUpExtend ()

  Move caret up one line extending selection to new caret position.
- void CharLeft ()

  Move caret left one character.
- void CharLeftExtend ()

  Move caret left one character extending selection to new caret position.
- void CharRight ()

  Move caret right one character.

- void CharRightExtend ()

  Move caret right one character extending selection to new caret position.
- void WordLeft ()

  Move caret left one word.
- void WordLeftExtend ()

  Move caret left one word extending selection to new caret position.
- void WordRight ()

  Move caret right one word.
- void WordRightExtend ()

  Move caret right one word extending selection to new caret position.
- void Home ()

  Move caret to first position on line.
- void HomeExtend ()

  Move caret to first position on line extending selection to new caret position.
- void LineEnd ()

  Move caret to last position on line.
- void LineEndExtend ()

  Move caret to last position on line extending selection to new caret position.
- void DocumentStart ()

  Move caret to first position in document.
- void DocumentStartExtend ()

  Move caret to first position in document extending selection to new caret position.
- void DocumentEnd ()

  Move caret to last position in document.
- void DocumentEndExtend ()

  Move caret to last position in document extending selection to new caret position.
- void PageUp ()

  Move caret one page up.
- void PageUpExtend ()

  Move caret one page up extending selection to new caret position.
- void PageDown ()

  Move caret one page down.
- void PageDownExtend ()

  Move caret one page down extending selection to new caret position.
- void EditToggleOvertype ()

  Switch from insert to overtype mode or the reverse.
- void Cancel ()

  Cancel any modes such as call tip or auto-completion list display.
- void DeleteBack ()

  Delete the selection or if no selection, the character before the caret.
- void Tab ()

  If selection is empty or all on one line replace the selection with a tab character.
- void BackTab ()

  Dedent the selected lines.
- void NewLine ()

  Insert a new line, may use a CRLF, CR or LF depending on EOL mode.
- void FormFeed ()

  Insert a Form Feed character.
- void VCHome ()

  Move caret to before first visible character on line.
- void VCHomeExtend ()

Like VCHome but extending selection to new caret position.

- void ZoomIn ()

  Magnify the displayed text by increasing the sizes by 1 point.
- void ZoomOut ()

  Make the displayed text smaller by decreasing the sizes by 1 point.
- void DelWordLeft ()

  Delete the word to the left of the caret.
- void DelWordRight ()

  Delete the word to the right of the caret.
- void DelWordRightEnd ()

  Delete the word to the right of the caret, but not the trailing non-word characters.
- void LineCut ()

  Cut the line containing the caret.
- void LineDelete ()

  Delete the line containing the caret.
- void LineTranspose ()

  Switch the current line with the previous.
- void LineDuplicate ()

  Duplicate the current line.
- void LowerCase ()

  Transform the selection to lower case.
- void UpperCase ()

  Transform the selection to upper case.
- void LineScrollDown ()

  Scroll the document down, keeping the caret visible.
- void LineScrollUp ()

  Scroll the document up, keeping the caret visible.
- void DeleteBackNotLine ()

  Delete the selection or if no selection, the character before the caret.
- void HomeDisplay ()

  Move caret to first position on display line.
- void HomeDisplayExtend ()

  Move caret to first position on display line extending selection to new caret position.
- void LineEndDisplay ()

  Move caret to last position on display line.
- void LineEndDisplayExtend ()

  Move caret to last position on display line extending selection to new caret position.
- void HomeWrap ()

  These are like their namesakes Home(Extend)?, LineEnd(Extend)?, VCHome(Extend)? except they behave differently when word-wrap is enabled: They go first to the start / end of the display line, like (Home|LineEnd)Display The difference is that, the cursor is already at the point, it goes on to the start or end of the document line, as appropriate for (Home|LineEnd|VCHome)(Extend)?.
- void HomeWrapExtend ()
- void LineEndWrap ()
- void LineEndWrapExtend ()
- void VCHomeWrap ()
- void VCHomeWrapExtend ()
- void LineCopy ()

  Copy the line containing the caret.
- void MoveCaretInsideView ()

  Move the caret inside current view if it's not there already.
- int LineLength (int line) const

How many characters are on a line, including end of line characters?

- void BraceHighlight (int pos1, int pos2)

  Highlight the characters at two positions.
- void BraceHighlightIndicator (bool useBraceHighlightIndicator, int indicator)

  Use specified indicator to highlight matching braces instead of changing their style.
- void BraceBadLight (int pos)

  Highlight the character at a position indicating there is no matching brace.
- void BraceBadLightIndicator (bool useBraceBadLightIndicator, int indicator)

  Use specified indicator to highlight non matching brace instead of changing its style.
- int BraceMatch (int pos)

  Find the position of a matching brace or INVALID_POSITION if no match.
- bool GetViewEOL () const

  Are the end of line characters visible?
- void SetViewEOL (bool visible)

  Make the end of line characters visible or invisible.
- void * GetDocPointer ()

  Retrieve a pointer to the document object.
- void SetDocPointer (void *docPointer)

  Change the document object used.
- void SetModEventMask (int mask)

  Set which document modification events are sent to the container.
- int GetEdgeColumn () const

  Retrieve the column number which text should be kept within.
- void SetEdgeColumn (int column)

  Set the column number of the edge.
- int GetEdgeMode () const

  Retrieve the edge highlight mode.
- void SetEdgeMode (int mode)

  The edge may be displayed by a line (EDGE_LINE) or by highlighting text that goes beyond it (EDGE_BACKGRO↩ UND) or not displayed at all (EDGE_NONE).
- wxColour GetEdgeColour () const

  Retrieve the colour used in edge indication.
- void SetEdgeColour (const wxColour &edgeColour)

  Change the colour used in edge indication.
- void SearchAnchor ()

  Sets the current caret position to be the search anchor.
- int SearchNext (int flags, const wxString &text)

  Find some text starting at the search anchor.
- int SearchPrev (int flags, const wxString &text)

  Find some text starting at the search anchor and moving backwards.
- int LinesOnScreen () const

  Retrieves the number of lines completely visible.
- void UsePopUp (bool allowPopUp)

  Set whether a pop up menu is displayed automatically when the user presses the wrong mouse button.
- bool SelectionIsRectangle () const

  Is the selection rectangular? The alternative is the more common stream selection.
- void SetZoom (int zoom)

  Set the zoom level.
- int GetZoom () const

  Retrieve the zoom level.
- void * CreateDocument ()

Create a new document object.

- void AddRefDocument (void *docPointer)

  Extend life of document.
- void ReleaseDocument (void *docPointer)

  Release a reference to the document, deleting document if it fades to black.
- int GetModEventMask () const

  Get which document modification events are sent to the container.
- void SetSTCFocus (bool focus)

  Change internal focus flag.
- bool GetSTCFocus () const

  Get internal focus flag.
- void SetStatus (int statusCode)

  Change error status - 0 = OK.
- int GetStatus () const

  Get error status.
- void SetMouseDownCaptures (bool captures)

  Set whether the mouse is captured when its button is pressed.
- bool GetMouseDownCaptures () const

  Get whether mouse gets captured.
- void SetSTCCursor (int cursorType)

  Sets the cursor to one of the SC_CURSOR values.*
- int GetSTCCursor () const

  Get cursor type.
- void SetControlCharSymbol (int symbol)

  Change the way control characters are displayed: If symbol is < 32, keep the drawn way, else, use the given character.
- int GetControlCharSymbol () const

  Get the way control characters are displayed.
- void WordPartLeft ()

  Move to the previous change in capitalisation.
- void WordPartLeftExtend ()

  Move to the previous change in capitalisation extending selection to new caret position.
- void WordPartRight ()

  Move to the change next in capitalisation.
- void WordPartRightExtend ()

  Move to the next change in capitalisation extending selection to new caret position.
- void SetVisiblePolicy (int visiblePolicy, int visibleSlop)

  Set the way the display area is determined when a particular line is to be moved to by Find, FindNext, GotoLine, etc.
- void DelLineLeft ()

  Delete back from the current position to the start of the line.
- void DelLineRight ()

  Delete forwards from the current position to the end of the line.
- void SetXOffset (int newOffset)

  Get and Set the xOffset (ie, horizontal scroll position).
- int GetXOffset () const
- void ChooseCaretX ()

  Set the last x chosen value to be the caret x position.
- void SetXCaretPolicy (int caretPolicy, int caretSlop)

  Set the way the caret is kept visible when going sideways.
- void SetYCaretPolicy (int caretPolicy, int caretSlop)

  Set the way the line the caret is on is kept visible.
- void SetPrintWrapMode (int mode)

Set printing to line wrapped (SC_WRAP_WORD) or not line wrapped (SC_WRAP_NONE).

- int GetPrintWrapMode () const

  Is printing line wrapped?

- void SetHotspotActiveForeground (bool useSetting, const wxColour &fore)

  Set a fore colour for active hotspots.

- wxColour GetHotspotActiveForeground () const

  Get the fore colour for active hotspots.

- void SetHotspotActiveBackground (bool useSetting, const wxColour &back)

  Set a back colour for active hotspots.

- wxColour GetHotspotActiveBackground () const

  Get the back colour for active hotspots.

- void SetHotspotActiveUnderline (bool underline)

  Enable / Disable underlining active hotspots.

- bool GetHotspotActiveUnderline () const

  Get whether underlining for active hotspots.

- void SetHotspotSingleLine (bool singleLine)

  Limit hotspots to single line so hotspots on two lines don't merge.

- bool GetHotspotSingleLine () const

  Get the HotspotSingleLine property.

- void ParaDown ()

  Move caret between paragraphs (delimited by empty lines).

- void ParaDownExtend ()
- void ParaUp ()
- void ParaUpExtend ()
- int PositionBefore (int pos)

  Given a valid document position, return the previous position taking code page into account.

- int PositionAfter (int pos)

  Given a valid document position, return the next position taking code page into account.

- int PositionRelative (int pos, int relative)

  Given a valid document position, return a position that differs in a number of characters.

- void CopyRange (int start, int end)

  Copy a range of text to the clipboard.

- void CopyText (int length, const wxString &text)

  Copy argument text to the clipboard.

- void SetSelectionMode (int mode)

  Set the selection mode to stream (SC_SEL_STREAM) or rectangular (SC_SEL_RECTANGLE/SC_SEL_THIN) or by lines (SC_SEL_LINES).

- int GetSelectionMode () const

  Get the mode of the current selection.

- int GetLineSelStartPosition (int line)

  Retrieve the position of the start of the selection at the given line (INVALID_POSITION if no selection on this line).

- int GetLineSelEndPosition (int line)

  Retrieve the position of the end of the selection at the given line (INVALID_POSITION if no selection on this line).

- void LineDownRectExtend ()

  Move caret down one line, extending rectangular selection to new caret position.

- void LineUpRectExtend ()

  Move caret up one line, extending rectangular selection to new caret position.

- void CharLeftRectExtend ()

  Move caret left one character, extending rectangular selection to new caret position.

- void CharRightRectExtend ()

  Move caret right one character, extending rectangular selection to new caret position.

- void HomeRectExtend ()

  Move caret to first position on line, extending rectangular selection to new caret position.
- void VCHomeRectExtend ()

  Move caret to before first visible character on line.
- void LineEndRectExtend ()

  Move caret to last position on line, extending rectangular selection to new caret position.
- void PageUpRectExtend ()

  Move caret one page up, extending rectangular selection to new caret position.
- void PageDownRectExtend ()

  Move caret one page down, extending rectangular selection to new caret position.
- void StutteredPageUp ()

  Move caret to top of page, or one page up if already at top of page.
- void StutteredPageUpExtend ()

  Move caret to top of page, or one page up if already at top of page, extending selection to new caret position.
- void StutteredPageDown ()

  Move caret to bottom of page, or one page down if already at bottom of page.
- void StutteredPageDownExtend ()

  Move caret to bottom of page, or one page down if already at bottom of page, extending selection to new caret position.
- void WordLeftEnd ()

  Move caret left one word, position cursor at end of word.
- void WordLeftEndExtend ()

  Move caret left one word, position cursor at end of word, extending selection to new caret position.
- void WordRightEnd ()

  Move caret right one word, position cursor at end of word.
- void WordRightEndExtend ()

  Move caret right one word, position cursor at end of word, extending selection to new caret position.
- void SetWhitespaceChars (const wxString &characters)

  Set the set of characters making up whitespace for when moving or selecting by word.
- wxString GetWhitespaceChars () const

  Get the set of characters making up whitespace for when moving or selecting by word.
- void SetPunctuationChars (const wxString &characters)

  Set the set of characters making up punctuation characters Should be called after SetWordChars.
- wxString GetPunctuationChars () const

  Get the set of characters making up punctuation characters.
- void SetCharsDefault ()

  Reset the set of characters for whitespace and word characters to the defaults.
- int AutoCompGetCurrent () const

  Get currently selected item position in the auto-completion list.
- void AutoCompSetCaseInsensitiveBehaviour (int behaviour)

  Set auto-completion case insensitive behaviour to either prefer case-sensitive matches or have no preference.
- int AutoCompGetCaseInsensitiveBehaviour () const

  Get auto-completion case insensitive behaviour.
- void AutoCompSetOrder (int order)

  Set the way autocompletion lists are ordered.
- int AutoCompGetOrder () const

  Get the way autocompletion lists are ordered.
- void Allocate (int bytes)

  Enlarge the document to a particular size of text bytes.
- int FindColumn (int line, int column)

  Find the position of a column on a line taking into account tabs and multi-byte characters.

- int GetCaretSticky () const

  Can the caret preferred x position only be changed by explicit movement commands?
- void SetCaretSticky (int useCaretStickyBehaviour)

  Stop the caret preferred x position changing when the user types.
- void ToggleCaretSticky ()

  Switch between sticky and non-sticky: meant to be bound to a key.
- void SetPasteConvertEndings (bool convert)

  Enable/Disable convert-on-paste for line endings.
- bool GetPasteConvertEndings () const

  Get convert-on-paste setting.
- void SelectionDuplicate ()

  Duplicate the selection.
- void SetCaretLineBackAlpha (int alpha)

  Set background alpha of the caret line.
- int GetCaretLineBackAlpha () const

  Get the background alpha of the caret line.
- void SetCaretStyle (int caretStyle)

  Set the style of the caret to be drawn.
- int GetCaretStyle () const

  Returns the current style of the caret.
- void SetIndicatorCurrent (int indicator)

  Set the indicator used for IndicatorFillRange and IndicatorClearRange.
- int GetIndicatorCurrent () const

  Get the current indicator.
- void SetIndicatorValue (int value)

  Set the value used for IndicatorFillRange.
- int GetIndicatorValue () const

  Get the current indicator value.
- void IndicatorFillRange (int position, int fillLength)

  Turn a indicator on over a range.
- void IndicatorClearRange (int position, int clearLength)

  Turn a indicator off over a range.
- int IndicatorAllOnFor (int position)

  Are any indicators present at position?
- int IndicatorValueAt (int indicator, int position)

  What value does a particular indicator have at at a position?
- int IndicatorStart (int indicator, int position)

  Where does a particular indicator start?
- int IndicatorEnd (int indicator, int position)

  Where does a particular indicator end?
- void SetPositionCacheSize (int size)

  Set number of entries in position cache.
- int GetPositionCacheSize () const

  How many entries are allocated to the position cache?
- void CopyAllowLine ()

  Copy the selection, if selection empty copy the line with the caret.
- const char * GetCharacterPointer () const

  Compact the document buffer and return a read-only pointer to the characters in the document.
- const char * GetRangePointer (int position, int rangeLength) const

  Return a read-only pointer to a range of characters in the document.
- int GetGapPosition () const

Return a position which, to avoid performance costs, should not be within the range of a call to GetRangePointer.

- void SetKeysUnicode (bool keysUnicode)

  Always interpret keyboard input as Unicode.
- bool GetKeysUnicode () const

  Are keys always interpreted as Unicode?
- void IndicatorSetAlpha (int indicator, int alpha)

  Set the alpha fill colour of the given indicator.
- int IndicatorGetAlpha (int indicator) const

  Get the alpha fill colour of the given indicator.
- void IndicatorSetOutlineAlpha (int indicator, int alpha)

  Set the alpha outline colour of the given indicator.
- int IndicatorGetOutlineAlpha (int indicator) const

  Get the alpha outline colour of the given indicator.
- void SetExtraAscent (int extraAscent)

  Set extra ascent for each line.
- int GetExtraAscent () const

  Get extra ascent for each line.
- void SetExtraDescent (int extraDescent)

  Set extra descent for each line.
- int GetExtraDescent () const

  Get extra descent for each line.
- int GetMarkerSymbolDefined (int markerNumber)

  Which symbol was defined for markerNumber with MarkerDefine.
- void MarginSetText (int line, const wxString &text)

  Set the text in the text margin for a line.
- wxString MarginGetText (int line) const

  Get the text in the text margin for a line.
- void MarginSetStyle (int line, int style)

  Set the style number for the text margin for a line.
- int MarginGetStyle (int line) const

  Get the style number for the text margin for a line.
- void MarginSetStyles (int line, const wxString &styles)

  Set the style in the text margin for a line.
- wxString MarginGetStyles (int line) const

  Get the styles in the text margin for a line.
- void MarginTextClearAll ()

  Clear the margin text on all lines.
- void MarginSetStyleOffset (int style)

  Get the start of the range of style numbers used for margin text.
- int MarginGetStyleOffset () const

  Get the start of the range of style numbers used for margin text.
- void SetMarginOptions (int marginOptions)

  Set the margin options.
- int GetMarginOptions () const

  Get the margin options.
- void AnnotationSetText (int line, const wxString &text)

  Set the annotation text for a line.
- wxString AnnotationGetText (int line) const

  Get the annotation text for a line.
- void AnnotationSetStyle (int line, int style)

  Set the style number for the annotations for a line.

- int AnnotationGetStyle (int line) const

  Get the style number for the annotations for a line.
- void AnnotationSetStyles (int line, const wxString &styles)

  Set the annotation styles for a line.
- wxString AnnotationGetStyles (int line) const

  Get the annotation styles for a line.
- int AnnotationGetLines (int line) const

  Get the number of annotation lines for a line.
- void AnnotationClearAll ()

  Clear the annotations from all lines.
- void AnnotationSetVisible (int visible)

  Set the visibility for the annotations for a view.
- int AnnotationGetVisible () const

  Get the visibility for the annotations for a view.
- void AnnotationSetStyleOffset (int style)

  Get the start of the range of style numbers used for annotations.
- int AnnotationGetStyleOffset () const

  Get the start of the range of style numbers used for annotations.
- void ReleaseAllExtendedStyles ()

  Release all extended (>255) style numbers.
- int AllocateExtendedStyles (int numberStyles)

  Allocate some extended (>255) style numbers and return the start of the range.
- void AddUndoAction (int token, int flags)

  Add a container action to the undo stack.
- int CharPositionFromPoint (int x, int y)

  Find the position of a character from a point within the window.
- int CharPositionFromPointClose (int x, int y)

  Find the position of a character from a point within the window.
- void SetMouseSelectionRectangularSwitch (bool mouseSelectionRectangularSwitch)

  Set whether switching to rectangular mode while selecting with the mouse is allowed.
- bool GetMouseSelectionRectangularSwitch () const

  Whether switching to rectangular mode while selecting with the mouse is allowed.
- void SetMultipleSelection (bool multipleSelection)

  Set whether multiple selections can be made.
- bool GetMultipleSelection () const

  Whether multiple selections can be made.
- void SetAdditionalSelectionTyping (bool additionalSelectionTyping)

  Set whether typing can be performed into multiple selections.
- bool GetAdditionalSelectionTyping () const

  Whether typing can be performed into multiple selections.
- void SetAdditionalCaretsBlink (bool additionalCaretsBlink)

  Set whether additional carets will blink.
- bool GetAdditionalCaretsBlink () const

  Whether additional carets will blink.
- void SetAdditionalCaretsVisible (bool additionalCaretsBlink)

  Set whether additional carets are visible.
- bool GetAdditionalCaretsVisible () const

  Whether additional carets are visible.
- int GetSelections () const

  How many selections are there?
- bool GetSelectionEmpty () const

Is every selected range empty?

- void ClearSelections ()

  Clear selections to a single empty stream selection.
- int AddSelection (int caret, int anchor)

  Add a selection.
- void DropSelectionN (int selection)

  Drop one selection.
- void SetMainSelection (int selection)

  Set the main selection.
- int GetMainSelection () const

  Which selection is the main selection.
- void SetSelectionNCaret (int selection, int pos)
- int GetSelectionNCaret (int selection) const
- void SetSelectionNAnchor (int selection, int posAnchor)
- int GetSelectionNAnchor (int selection) const
- void SetSelectionNCaretVirtualSpace (int selection, int space)
- int GetSelectionNCaretVirtualSpace (int selection) const
- void SetSelectionNAnchorVirtualSpace (int selection, int space)
- int GetSelectionNAnchorVirtualSpace (int selection) const
- void SetSelectionNStart (int selection, int pos)

  Sets the position that starts the selection - this becomes the anchor.
- int GetSelectionNStart (int selection) const

  Returns the position at the start of the selection.
- void SetSelectionNEnd (int selection, int pos)

  Sets the position that ends the selection - this becomes the currentPosition.
- int GetSelectionNEnd (int selection) const

  Returns the position at the end of the selection.
- void SetRectangularSelectionCaret (int pos)
- int GetRectangularSelectionCaret () const
- void SetRectangularSelectionAnchor (int posAnchor)
- int GetRectangularSelectionAnchor () const
- void SetRectangularSelectionCaretVirtualSpace (int space)
- int GetRectangularSelectionCaretVirtualSpace () const
- void SetRectangularSelectionAnchorVirtualSpace (int space)
- int GetRectangularSelectionAnchorVirtualSpace () const
- void SetVirtualSpaceOptions (int virtualSpaceOptions)
- int GetVirtualSpaceOptions () const
- void SetRectangularSelectionModifier (int modifier)

  On GTK+, allow selecting the modifier key to use for mouse-based rectangular selection.
- int GetRectangularSelectionModifier () const

  Get the modifier key used for rectangular selection.
- void SetAdditionalSelForeground (const wxColour &fore)

  Set the foreground colour of additional selections.
- void SetAdditionalSelBackground (const wxColour &back)

  Set the background colour of additional selections.
- void SetAdditionalSelAlpha (int alpha)

  Set the alpha of the selection.
- int GetAdditionalSelAlpha () const

  Get the alpha of the selection.
- void SetAdditionalCaretForeground (const wxColour &fore)

  Set the foreground colour of additional carets.
- wxColour GetAdditionalCaretForeground () const

Get the foreground colour of additional carets.

- void RotateSelection ()

  Set the main selection to the next selection.
- void SwapMainAnchorCaret ()

  Swap that caret and anchor of the main selection.
- int ChangeLexerState (int start, int end)

  Indicate that the internal state of a lexer has changed over a range and therefore there may be a need to redraw.
- int ContractedFoldNext (int lineStart)

  Find the next line at or after lineStart that is a contracted fold header line.
- void VerticalCentreCaret ()

  Centre current line in window.
- void MoveSelectedLinesUp ()

  Move the selected lines up one line, shifting the line above after the selection.
- void MoveSelectedLinesDown ()

  Move the selected lines down one line, shifting the line below before the selection.
- void SetIdentifier (int identifier)

  Set the identifier reported as idFrom in notification messages.
- int GetIdentifier () const

  Get the identifier.
- void RGBAImageSetWidth (int width)

  Set the width for future RGBA image data.
- void RGBAImageSetHeight (int height)

  Set the height for future RGBA image data.
- void RGBAImageSetScale (int scalePercent)

  Set the scale factor in percent for future RGBA image data.
- void MarkerDefineRGBAImage (int markerNumber, const unsigned char ∗pixels)

  Define a marker from RGBA data.
- void RegisterRGBAImage (int type, const unsigned char ∗pixels)

  Register an RGBA image for use in autocompletion lists.
- void ScrollToStart ()

  Scroll to start of document.
- void ScrollToEnd ()

  Scroll to end of document.
- void SetTechnology (int technology)

  Set the technology used.
- int GetTechnology () const

  Get the tech.
- void ∗ CreateLoader (int bytes) const

  Create an ILoader∗.
- void VCHomeDisplay ()

  Move caret to before first visible character on display line.
- void VCHomeDisplayExtend ()

  Like VCHomeDisplay but extending selection to new caret position.
- bool GetCaretLineVisibleAlways () const

  Is the caret line always visible?
- void SetCaretLineVisibleAlways (bool alwaysVisible)

  Sets the caret line to always visible.
- void SetLineEndTypesAllowed (int lineEndBitSet)

  Set the line end types that the application wants to use.
- int GetLineEndTypesAllowed () const

  Get the line end types currently allowed.

- int GetLineEndTypesActive () const

  Get the line end types currently recognised.
- void SetRepresentation (const wxString &encodedCharacter, const wxString &representation)

  Set the way a character is drawn.
- wxString GetRepresentation (const wxString &encodedCharacter) const

  Set the way a character is drawn.
- void ClearRepresentation (const wxString &encodedCharacter)

  Remove a character representation.
- void StartRecord ()

  Start notifying the container of all key presses and commands.
- void StopRecord ()

  Stop notifying the container of all key presses and commands.
- void SetLexer (int lexer)

  Set the lexing language of the document.
- int GetLexer () const

  Retrieve the lexing language of the document.
- void Colourise (int start, int end)

  Colourise a segment of the document using the current lexing language.
- void SetProperty (const wxString &key, const wxString &value)

  Set up a value that may be used by a lexer for some optional feature.
- void SetKeyWords (int keywordSet, const wxString &keyWords)

  Set up the key words used by the lexer.
- void SetLexerLanguage (const wxString &language)

  Set the lexing language of the document based on string name.
- wxString GetProperty (const wxString &key)

  Retrieve a 'property' value previously set with SetProperty.
- wxString GetPropertyExpanded (const wxString &key)

  Retrieve a 'property' value previously set with SetProperty, with '$()' variable replacement on returned buffer.
- int GetPropertyInt (const wxString &key) const

  Retrieve a 'property' value previously set with SetProperty, interpreted as an int AFTER any '$()' variable replacement.
- int GetStyleBitsNeeded () const

  Retrieve the number of bits the current lexer needs for styling.
- void * PrivateLexerCall (int operation, void *pointer)

  For private communication between an application and a known lexer.
- wxString PropertyNames () const

  Retrieve a '
  ' separated list of properties understood by the current lexer.
- int PropertyType (const wxString &name)

  Retrieve the type of a property.
- wxString DescribeProperty (const wxString &name) const

  Describe a property.
- wxString DescribeKeyWordSets () const

  Retrieve a '
  ' separated list of descriptions of the keyword sets understood by the current lexer.
- int GetLineEndTypesSupported () const

  Bit set of LineEndType enumertion for which line ends beyond the standard LF, CR, and CRLF are supported by the lexer.
- int AllocateSubStyles (int styleBase, int numberStyles)

  Allocate a set of sub styles for a particular base style, returning start of range.
- int GetSubStylesStart (int styleBase) const

  The starting style number for the sub styles associated with a base style.

- int GetSubStylesLength (int styleBase) const

  The number of sub styles associated with a base style.
- int GetStyleFromSubStyle (int subStyle) const

  For a sub style, return the base style, else return the argument.
- int GetPrimaryStyleFromStyle (int style) const

  For a secondary style, return the primary style, else return the argument.
- void FreeSubStyles ()

  Free allocated sub styles.
- void SetIdentifiers (int style, const wxString &identifiers)

  Set the identifiers that are shown in a particular style.
- int DistanceToSecondaryStyles () const

  Where styles are duplicated by a feature such as active/inactive code return the distance between the two types.
- wxString GetSubStyleBases () const

  Get the set of base styles that can be extended with sub styles.
- int GetCurrentLine ()

  Returns the line number of the line with the caret.
- void StyleSetSpec (int styleNum, const wxString &spec)

  Extract style settings from a spec-string which is composed of one or more of the following comma separated elements:
- wxFont StyleGetFont (int style)

  Get the font of a style.
- void StyleSetFont (int styleNum, wxFont &font)

  Set style size, face, bold, italic, and underline attributes from a wxFont's attributes.
- void StyleSetFontAttr (int styleNum, int size, const wxString &faceName, bool bold, bool italic, bool underline, wxFontEncoding encoding=wxFONTENCODING_DEFAULT)

  Set all font style attributes at once.
- void StyleSetCharacterSet (int style, int characterSet)

  Set the character set of the font in a style.
- void StyleSetFontEncoding (int style, wxFontEncoding encoding)

  Set the font encoding to be used by a style.
- void CmdKeyExecute (int cmd)

  Perform one of the operations defined by the wxSTC_CMD_∗ constants.
- void SetMargins (int left, int right)

  Set the left and right margin in the edit area, measured in pixels.
- wxPoint PointFromPosition (int pos)

  Retrieve the point in the window where a position is displayed.
- void ScrollToLine (int line)

  Scroll enough to make the given line visible.
- void ScrollToColumn (int column)

  Scroll enough to make the given column visible.
- wxIntPtr SendMsg (int msg, wxUIntPtr wp=0, wxIntPtr lp=0) const

  Send a message to Scintilla.
- void SetVScrollBar (wxScrollBar ∗bar)

  Set the vertical scrollbar to use instead of the ont that's built-in.
- void SetHScrollBar (wxScrollBar ∗bar)

  Set the horizontal scrollbar to use instead of the ont that's built-in.
- bool GetLastKeydownProcessed ()

  Can be used to prevent the EVT_CHAR handler from adding the char.
- void SetLastKeydownProcessed (bool val)
- bool SaveFile (const wxString &filename)

  Write the contents of the editor to filename.

- bool LoadFile (const wxString &filename)

  Load the contents of filename into the editor.
- wxDragResult DoDragEnter (wxCoord x, wxCoord y, wxDragResult defaultRes)

  Allow for simulating a DnD DragEnter.
- wxDragResult DoDragOver (wxCoord x, wxCoord y, wxDragResult defaultRes)

  Allow for simulating a DnD DragOver.
- void DoDragLeave ()

  Allow for simulating a DnD DragLeave.
- bool DoDropText (long x, long y, const wxString &data)

  Allow for simulating a DnD DropText.
- void AnnotationClearLine (int line)

  Clear annotations from the given line.
- void AddTextRaw (const char *text, int length=-1)

  Add text to the document at current position.
- void InsertTextRaw (int pos, const char *text)

  Insert string at a position.
- wxCharBuffer GetCurLineRaw (int *linePos=NULL)

  Retrieve the text of the line containing the caret.
- wxCharBuffer GetLineRaw (int line)

  Retrieve the contents of a line.
- wxCharBuffer GetSelectedTextRaw ()

  Retrieve the selected text.
- wxCharBuffer GetTextRangeRaw (int startPos, int endPos)

  Retrieve a range of text.
- void SetTextRaw (const char *text)

  Replace the contents of the document with the argument text.
- wxCharBuffer GetTextRaw ()

  Retrieve all the text in the document.
- void AppendTextRaw (const char *text, int length=-1)

  Append a string to the end of the document without changing the selection.
- virtual void WriteText (const wxString &text)

  Writes the text into the text control at the current insertion position.
- virtual void Remove (long from, long to)

  Removes the text starting at the first given position up to (but not including) the character at the last position.
- virtual void Replace (long from, long to, const wxString &text)

  Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.
- virtual void SetInsertionPoint (long pos)

  Sets the insertion point at the given position.
- virtual long GetInsertionPoint () const

  Returns the insertion point, or cursor, position.
- virtual long GetLastPosition () const

  Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.
- virtual void SetSelection (long from, long to)

  Selects the text starting at the first position up to (but not including) the character at the last position.
- virtual void SelectNone ()

  Deselects selected text in the control.
- virtual void GetSelection (long *from, long *to) const

  Gets the current selection span.
- virtual bool IsEditable () const

Returns true if the controls contents may be edited by user (note that it always can be changed by the program).

- virtual void SetEditable (bool editable)

  *Makes the text item editable or read-only, overriding the **wxTE_READONLY** flag.*
- virtual int GetLineLength (long n) const
- virtual wxString GetLineText (long n) const
- virtual int GetNumberOfLines () const
- virtual bool IsModified () const
- virtual void MarkDirty ()
- virtual void DiscardEdits ()
- virtual bool SetStyle (long start, long end, const wxTextAttr &style)
- virtual bool GetStyle (long position, wxTextAttr &style)
- virtual bool SetDefaultStyle (const wxTextAttr &style)
- virtual long XYToPosition (long x, long y) const
- virtual bool PositionToXY (long pos, long *x, long *y) const
- virtual void ShowPosition (long pos)
- virtual wxTextCtrlHitTestResult HitTest (const wxPoint &pt, long *pos) const
- virtual wxTextCtrlHitTestResult HitTest (const wxPoint &pt, wxTextCoord *col, wxTextCoord *row) const

Static Public Member Functions

- static wxVersionInfo GetLibraryVersionInfo ()

Additional Inherited Members

21.737.2 Constructor & Destructor Documentation

wxStyledTextCtrl::wxStyledTextCtrl (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxSTCNameStr)

Ctor.

wxStyledTextCtrl::wxStyledTextCtrl ( )

Default ctor.

wxStyledTextCtrl::~wxStyledTextCtrl ( )

Destructor.

21.737.3 Member Function Documentation

void wxStyledTextCtrl::AddRefDocument (void * *docPointer*)

Extend life of document.

int wxStyledTextCtrl::AddSelection (int *caret*, int *anchor*)

Add a selection.

void wxStyledTextCtrl::AddStyledText (const wxMemoryBuffer & *data*)

Add array of cells to document.

void wxStyledTextCtrl::AddText (const wxString & *text*)

Add text to the document at current position.

void wxStyledTextCtrl::AddTextRaw (const char * *text,* int *length* = -1)

Add text to the document at current position.

void wxStyledTextCtrl::AddUndoAction (int *token,* int *flags*)

Add a container action to the undo stack.

void wxStyledTextCtrl::Allocate (int *bytes*)

Enlarge the document to a particular size of text bytes.

int wxStyledTextCtrl::AllocateExtendedStyles (int *numberStyles*)

Allocate some extended (>255) style numbers and return the start of the range.

int wxStyledTextCtrl::AllocateSubStyles (int *styleBase,* int *numberStyles*)

Allocate a set of sub styles for a particular base style, returning start of range.

void wxStyledTextCtrl::AnnotationClearAll ( )

Clear the annotations from all lines.

void wxStyledTextCtrl::AnnotationClearLine (int *line*)

Clear annotations from the given line.

int wxStyledTextCtrl::AnnotationGetLines (int *line*) const

Get the number of annotation lines for a line.

int wxStyledTextCtrl::AnnotationGetStyle (int *line*) const

Get the style number for the annotations for a line.

int wxStyledTextCtrl::AnnotationGetStyleOffset ( ) const

Get the start of the range of style numbers used for annotations.

wxString wxStyledTextCtrl::AnnotationGetStyles (int *line*) const

Get the annotation styles for a line.

wxString wxStyledTextCtrl::AnnotationGetText (int *line*) const

Get the annotation text for a line.

int wxStyledTextCtrl::AnnotationGetVisible ( ) const

Get the visibility for the annotations for a view.

void wxStyledTextCtrl::AnnotationSetStyle (int *line,* int *style*)

Set the style number for the annotations for a line.

void wxStyledTextCtrl::AnnotationSetStyleOffset (int *style*)

Get the start of the range of style numbers used for annotations.

void wxStyledTextCtrl::AnnotationSetStyles (int *line,* const wxString & *styles*)

Set the annotation styles for a line.

void wxStyledTextCtrl::AnnotationSetText (int *line,* const wxString & *text*)

Set the annotation text for a line.

void wxStyledTextCtrl::AnnotationSetVisible (int *visible*)

Set the visibility for the annotations for a view.

void wxStyledTextCtrl::AppendText (const wxString & *text*) `[virtual]`

Append a string to the end of the document without changing the selection.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::AppendTextRaw (const char * *text,* int *length* = -1)

Append a string to the end of the document without changing the selection.

bool wxStyledTextCtrl::AutoCompActive ( )

Is there an auto-completion list visible?

void wxStyledTextCtrl::AutoCompCancel ( )

Remove the auto-completion list from the screen.

void wxStyledTextCtrl::AutoCompComplete ( )

User has selected an item so remove the list and insert the selection.

bool wxStyledTextCtrl::AutoCompGetAutoHide ( ) const

Retrieve whether or not autocompletion is hidden automatically when nothing matches.

bool wxStyledTextCtrl::AutoCompGetCancelAtStart ( ) const

Retrieve whether auto-completion cancelled by backspacing before start.

int wxStyledTextCtrl::AutoCompGetCaseInsensitiveBehaviour ( ) const

Get auto-completion case insensitive behaviour.

bool wxStyledTextCtrl::AutoCompGetChooseSingle ( ) const

Retrieve whether a single item auto-completion list automatically choose the item.

int wxStyledTextCtrl::AutoCompGetCurrent ( ) const

Get currently selected item position in the auto-completion list.

bool wxStyledTextCtrl::AutoCompGetDropRestOfWord ( ) const

Retrieve whether or not autocompletion deletes any word characters after the inserted text upon completion.

bool wxStyledTextCtrl::AutoCompGetIgnoreCase ( ) const

Retrieve state of ignore case flag.

int wxStyledTextCtrl::AutoCompGetMaxHeight ( ) const

Set the maximum height, in rows, of auto-completion and user lists.

int wxStyledTextCtrl::AutoCompGetMaxWidth ( ) const

Get the maximum width, in characters, of auto-completion and user lists.

int wxStyledTextCtrl::AutoCompGetOrder ( ) const

Get the way autocompletion lists are ordered.

int wxStyledTextCtrl::AutoCompGetSeparator ( ) const

Retrieve the auto-completion list separator character.

int wxStyledTextCtrl::AutoCompGetTypeSeparator ( ) const

Retrieve the auto-completion list type-separator character.

int wxStyledTextCtrl::AutoCompPosStart ( )

Retrieve the position of the caret when the auto-completion list was displayed.

void wxStyledTextCtrl::AutoCompSelect (const wxString & *text*)

Select the item in the auto-completion list that starts with a string.

void wxStyledTextCtrl::AutoCompSetAutoHide (bool *autoHide*)

Set whether or not autocompletion is hidden automatically when nothing matches.

void wxStyledTextCtrl::AutoCompSetCancelAtStart (bool *cancel*)

Should the auto-completion list be cancelled if the user backspaces to a position before where the box was created.

void wxStyledTextCtrl::AutoCompSetCaseInsensitiveBehaviour (int *behaviour*)

Set auto-completion case insensitive behaviour to either prefer case-sensitive matches or have no preference.

void wxStyledTextCtrl::AutoCompSetChooseSingle (bool *chooseSingle*)

Should a single item auto-completion list automatically choose the item.

void wxStyledTextCtrl::AutoCompSetDropRestOfWord (bool *dropRestOfWord*)

Set whether or not autocompletion deletes any word characters after the inserted text upon completion.

void wxStyledTextCtrl::AutoCompSetFillUps (const wxString & *characterSet*)

Define a set of characters that when typed will cause the autocompletion to choose the selected item.

void wxStyledTextCtrl::AutoCompSetIgnoreCase (bool *ignoreCase*)

Set whether case is significant when performing auto-completion searches.

void wxStyledTextCtrl::AutoCompSetMaxHeight (int *rowCount*)

Set the maximum height, in rows, of auto-completion and user lists.

The default is 5 rows.

void wxStyledTextCtrl::AutoCompSetMaxWidth (int *characterCount*)

Set the maximum width, in characters, of auto-completion and user lists.

Set to 0 to autosize to fit longest item, which is the default.

void wxStyledTextCtrl::AutoCompSetOrder (int *order*)

Set the way autocompletion lists are ordered.

void wxStyledTextCtrl::AutoCompSetSeparator (int *separatorCharacter*)

Change the separator character in the string setting up an auto-completion list.

Default is space but can be changed if items contain space.

void wxStyledTextCtrl::AutoCompSetTypeSeparator (int *separatorCharacter*)

Change the type-separator character in the string setting up an auto-completion list.

Default is '?' but can be changed if items contain '?'.

void wxStyledTextCtrl::AutoCompShow (int *lenEntered,* const wxString & *itemList*)

Display a auto-completion list.

The lenEntered parameter indicates how many characters before the caret should be used to provide context.

void wxStyledTextCtrl::AutoCompStops (const wxString & *characterSet*)

Define a set of character that when typed cancel the auto-completion list.

void wxStyledTextCtrl::BackTab ( )

Dedent the selected lines.

void wxStyledTextCtrl::BeginUndoAction ( )

Start a sequence of actions that is undone and redone as a unit.

May be nested.

void wxStyledTextCtrl::BraceBadLight (int *pos*)

Highlight the character at a position indicating there is no matching brace.

void wxStyledTextCtrl::BraceBadLightIndicator (bool *useBraceBadLightIndicator,* int *indicator*)

Use specified indicator to highlight non matching brace instead of changing its style.

void wxStyledTextCtrl::BraceHighlight (int *pos1,* int *pos2*)

Highlight the characters at two positions.

void wxStyledTextCtrl::BraceHighlightIndicator (bool *useBraceHighlightIndicator,* int *indicator*)

Use specified indicator to highlight matching braces instead of changing their style.

int wxStyledTextCtrl::BraceMatch (int *pos*)

Find the position of a matching brace or INVALID_POSITION if no match.

bool wxStyledTextCtrl::CallTipActive ( )

Is there an active call tip?

void wxStyledTextCtrl::CallTipCancel ( )

Remove the call tip from the screen.

int wxStyledTextCtrl::CallTipPosAtStart ( )

Retrieve the position where the caret was before displaying the call tip.

void wxStyledTextCtrl::CallTipSetBackground (const wxColour & *back*)

Set the background colour for the call tip.

void wxStyledTextCtrl::CallTipSetForeground (const wxColour & *fore*)

Set the foreground colour for the call tip.

void wxStyledTextCtrl::CallTipSetForegroundHighlight (const wxColour & *fore*)

Set the foreground colour for the highlighted part of the call tip.

void wxStyledTextCtrl::CallTipSetHighlight (int *start,* int *end*)

Highlight a segment of the definition.

void wxStyledTextCtrl::CallTipSetPosAtStart (int *posStart*)

Set the start position in order to change when backspacing removes the calltip.

void wxStyledTextCtrl::CallTipSetPosition (bool *above*)

Set position of calltip, above or below text.

void wxStyledTextCtrl::CallTipShow (int *pos,* const wxString & *definition*)

Show a call tip containing a definition near position pos.

void wxStyledTextCtrl::CallTipUseStyle (int *tabSize*)

Enable use of STYLE_CALLTIP and set call tip tab size in pixels.

void wxStyledTextCtrl::Cancel ( )

Cancel any modes such as call tip or auto-completion list display.

bool wxStyledTextCtrl::CanPaste ( ) const `[virtual]`

Will a paste succeed?

Reimplemented from wxTextEntry.

bool wxStyledTextCtrl::CanRedo ( ) const `[virtual]`

Are there any redoable actions in the undo history?

Reimplemented from wxTextEntry.

bool wxStyledTextCtrl::CanUndo ( ) const `[virtual]`

Are there any undoable actions in the undo history?

Reimplemented from wxTextEntry.

int wxStyledTextCtrl::ChangeLexerState (int *start*, int *end*)

Indicate that the internal state of a lexer has changed over a range and therefore there may be a need to redraw.

void wxStyledTextCtrl::CharLeft ( )

Move caret left one character.

void wxStyledTextCtrl::CharLeftExtend ( )

Move caret left one character extending selection to new caret position.

void wxStyledTextCtrl::CharLeftRectExtend ( )

Move caret left one character, extending rectangular selection to new caret position.

int wxStyledTextCtrl::CharPositionFromPoint (int *x*, int *y*)

Find the position of a character from a point within the window.

int wxStyledTextCtrl::CharPositionFromPointClose (int *x*, int *y*)

Find the position of a character from a point within the window.

Return INVALID_POSITION if not close to text.

void wxStyledTextCtrl::CharRight ( )

Move caret right one character.

void wxStyledTextCtrl::CharRightExtend ( )

Move caret right one character extending selection to new caret position.

void wxStyledTextCtrl::CharRightRectExtend ( )

Move caret right one character, extending rectangular selection to new caret position.

void wxStyledTextCtrl::ChooseCaretX ( )

Set the last x chosen value to be the caret x position.

void wxStyledTextCtrl::Clear ( ) `[virtual]`

Clear the selection.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::ClearAll ( )

Delete all text in the document.

void wxStyledTextCtrl::ClearDocumentStyle ( )

Set all style bytes to 0, remove all folding information.

void wxStyledTextCtrl::ClearRegisteredImages ( )

Clear all the registered images.

void wxStyledTextCtrl::ClearRepresentation (const wxString & *encodedCharacter*)

Remove a character representation.

void wxStyledTextCtrl::ClearSelections ( )

Clear selections to a single empty stream selection.

void wxStyledTextCtrl::CmdKeyAssign (int *key,* int *modifiers,* int *cmd*)

When key+modifier combination km is pressed perform msg.

void wxStyledTextCtrl::CmdKeyClear (int *key,* int *modifiers*)

When key+modifier combination km is pressed do nothing.

void wxStyledTextCtrl::CmdKeyClearAll ( )

Drop all key mappings.

void wxStyledTextCtrl::CmdKeyExecute (int *cmd*)

Perform one of the operations defined by the wxSTC_CMD_* constants.

void wxStyledTextCtrl::Colourise (int *start*, int *end*)

Colourise a segment of the document using the current lexing language.

int wxStyledTextCtrl::ContractedFoldNext (int *lineStart*)

Find the next line at or after lineStart that is a contracted fold header line.

Return -1 when no more lines.

void wxStyledTextCtrl::ConvertEOLs (int *eolMode*)

Convert all line endings in the document to one mode.

void wxStyledTextCtrl::Copy ( ) [virtual]

Copy the selection to the clipboard.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::CopyAllowLine ( )

Copy the selection, if selection empty copy the line with the caret.

void wxStyledTextCtrl::CopyRange (int *start*, int *end*)

Copy a range of text to the clipboard.

Positions are clipped into the document.

void wxStyledTextCtrl::CopyText (int *length*, const wxString & *text*)

Copy argument text to the clipboard.

int wxStyledTextCtrl::CountCharacters (int *startPos*, int *endPos*)

Count characters between two positions.

bool wxStyledTextCtrl::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxSTCNameStr)

Create the UI elements for a STC that was created with the default ctor.

(For 2-phase create.)

void* wxStyledTextCtrl::CreateDocument ( )

Create a new document object.

Starts with reference count of 1 and not selected into editor.

void* wxStyledTextCtrl::CreateLoader (int *bytes*) const

Create an ILoader*.

void wxStyledTextCtrl::Cut ( ) [virtual]

Cut the selection to the clipboard.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::DeleteBack ( )

Delete the selection or if no selection, the character before the caret.

void wxStyledTextCtrl::DeleteBackNotLine ( )

Delete the selection or if no selection, the character before the caret.

Will not delete the character before at the start of a line.

void wxStyledTextCtrl::DeleteRange (int *pos,* int *deleteLength*)

Delete a range of text in the document.

void wxStyledTextCtrl::DelLineLeft ( )

Delete back from the current position to the start of the line.

void wxStyledTextCtrl::DelLineRight ( )

Delete forwards from the current position to the end of the line.

void wxStyledTextCtrl::DelWordLeft ( )

Delete the word to the left of the caret.

void wxStyledTextCtrl::DelWordRight ( )

Delete the word to the right of the caret.

void wxStyledTextCtrl::DelWordRightEnd ( )

Delete the word to the right of the caret, but not the trailing non-word characters.

wxString wxStyledTextCtrl::DescribeKeyWordSets ( ) const

Retrieve a '
' separated list of descriptions of the keyword sets understood by the current lexer.

wxString wxStyledTextCtrl::DescribeProperty (const wxString & *name*) const

Describe a property.

virtual void wxStyledTextCtrl::DiscardEdits ( ) [virtual]

int wxStyledTextCtrl::DistanceToSecondaryStyles ( ) const

Where styles are duplicated by a feature such as active/inactive code return the distance between the two types.

int wxStyledTextCtrl::DocLineFromVisible (int *lineDisplay*)

Find the document line of a display line taking hidden lines into account.

void wxStyledTextCtrl::DocumentEnd ( )

Move caret to last position in document.

void wxStyledTextCtrl::DocumentEndExtend ( )

Move caret to last position in document extending selection to new caret position.

void wxStyledTextCtrl::DocumentStart ( )

Move caret to first position in document.

void wxStyledTextCtrl::DocumentStartExtend ( )

Move caret to first position in document extending selection to new caret position.

wxDragResult wxStyledTextCtrl::DoDragEnter (wxCoord *x*, wxCoord *y*, wxDragResult *defaultRes*)

Allow for simulating a DnD DragEnter.

Since

3.1.0

void wxStyledTextCtrl::DoDragLeave ( )

Allow for simulating a DnD DragLeave.

Since

3.1.0

wxDragResult wxStyledTextCtrl::DoDragOver (wxCoord *x,* wxCoord *y,* wxDragResult *defaultRes*)

Allow for simulating a DnD DragOver.

bool wxStyledTextCtrl::DoDropText (long *x,* long *y,* const wxString & *data*)

Allow for simulating a DnD DropText.

void wxStyledTextCtrl::DropSelectionN (int *selection*)

Drop one selection.

void wxStyledTextCtrl::EditToggleOvertype ( )

Switch from insert to overtype mode or the reverse.

void wxStyledTextCtrl::EmptyUndoBuffer ( )

Delete the undo history.

void wxStyledTextCtrl::EndUndoAction ( )

End a sequence of actions that is undone and redone as a unit.

void wxStyledTextCtrl::EnsureCaretVisible ( )

Ensure the caret is visible.

void wxStyledTextCtrl::EnsureVisible (int *line*)

Ensure a particular line is visible by expanding any header line hiding it.

void wxStyledTextCtrl::EnsureVisibleEnforcePolicy (int *line*)

Ensure a particular line is visible by expanding any header line hiding it.
Use the currently set visibility policy to determine which range to display.

void wxStyledTextCtrl::ExpandChildren (int *line,* int *level*)

Expand a fold header and all children.
Use the level argument instead of the line's current level.

int wxStyledTextCtrl::FindColumn (int *line,* int *column*)

Find the position of a column on a line taking into account tabs and multi-byte characters.
If beyond end of line, return line end position.

int wxStyledTextCtrl::FindText (int *minPos,* int *maxPos,* const wxString & *text,* int *flags* = 0)

Find some text in the document.

void wxStyledTextCtrl::FoldAll (int *action*)

Expand or contract all fold headers.

void wxStyledTextCtrl::FoldChildren (int *line,* int *action*)

Expand or contract a fold header and its children.

void wxStyledTextCtrl::FoldLine (int *line,* int *action*)

Expand or contract a fold header.

int wxStyledTextCtrl::FormatRange (bool *doDraw,* int *startPos,* int *endPos,* wxDC * *draw,* wxDC * *target,* wxRect *renderRect,* wxRect *pageRect*)

On Windows, will draw the document into a display context such as a printer.

void wxStyledTextCtrl::FormFeed ( )

Insert a Form Feed character.

void wxStyledTextCtrl::FreeSubStyles ( )

Free allocated sub styles.

wxColour wxStyledTextCtrl::GetAdditionalCaretForeground ( ) const

Get the foreground colour of additional carets.

bool wxStyledTextCtrl::GetAdditionalCaretsBlink ( ) const

Whether additional carets will blink.

bool wxStyledTextCtrl::GetAdditionalCaretsVisible ( ) const

Whether additional carets are visible.

int wxStyledTextCtrl::GetAdditionalSelAlpha ( ) const

Get the alpha of the selection.

bool wxStyledTextCtrl::GetAdditionalSelectionTyping ( ) const

Whether typing can be performed into multiple selections.

bool wxStyledTextCtrl::GetAllLinesVisible ( ) const

Are all lines visible?

int wxStyledTextCtrl::GetAnchor ( ) const

Returns the position of the opposite end of the selection to the caret.

int wxStyledTextCtrl::GetAutomaticFold ( ) const

Get automatic folding behaviours.

bool wxStyledTextCtrl::GetBackSpaceUnIndents ( ) const

Does a backspace pressed when caret is within indentation unindent?

bool wxStyledTextCtrl::GetBufferedDraw ( ) const

Is drawing done first into a buffer or direct to the screen?

wxColour wxStyledTextCtrl::GetCaretForeground ( ) const

Get the foreground colour of the caret.

int wxStyledTextCtrl::GetCaretLineBackAlpha ( ) const

Get the background alpha of the caret line.

wxColour wxStyledTextCtrl::GetCaretLineBackground ( ) const

Get the colour of the background of the line containing the caret.

bool wxStyledTextCtrl::GetCaretLineVisible ( ) const

Is the background of the line containing the caret in a different colour?

bool wxStyledTextCtrl::GetCaretLineVisibleAlways ( ) const

Is the caret line always visible?

int wxStyledTextCtrl::GetCaretPeriod ( ) const

Get the time in milliseconds that the caret is on and off.

int wxStyledTextCtrl::GetCaretSticky ( ) const

Can the caret preferred x position only be changed by explicit movement commands?

int wxStyledTextCtrl::GetCaretStyle ( ) const

Returns the current style of the caret.

int wxStyledTextCtrl::GetCaretWidth ( ) const

Returns the width of the insert mode caret.

const char* wxStyledTextCtrl::GetCharacterPointer ( ) const

Compact the document buffer and return a read-only pointer to the characters in the document.

int wxStyledTextCtrl::GetCharAt (int *pos*) const

Returns the character byte at the position.

int wxStyledTextCtrl::GetCodePage ( ) const

Get the code page used to interpret the bytes of the document as characters.

int wxStyledTextCtrl::GetColumn (int *pos*) const

Retrieve the column number of a position, taking tab width into account.

int wxStyledTextCtrl::GetControlCharSymbol ( ) const

Get the way control characters are displayed.

wxString wxStyledTextCtrl::GetCurLine (int * *linePos* = NULL)

Retrieve the text of the line containing the caret.

Returns the index of the caret on the line.

wxCharBuffer wxStyledTextCtrl::GetCurLineRaw (int * *linePos* = NULL)

Retrieve the text of the line containing the caret.

Returns the index of the caret on the line.

int wxStyledTextCtrl::GetCurrentLine ( )

Returns the line number of the line with the caret.

int wxStyledTextCtrl::GetCurrentPos ( ) const

Returns the position of the caret.

void* wxStyledTextCtrl::GetDocPointer ( )

Retrieve a pointer to the document object.

wxColour wxStyledTextCtrl::GetEdgeColour ( ) const

Retrieve the colour used in edge indication.

int wxStyledTextCtrl::GetEdgeColumn ( ) const

Retrieve the column number which text should be kept within.

int wxStyledTextCtrl::GetEdgeMode ( ) const

Retrieve the edge highlight mode.

bool wxStyledTextCtrl::GetEndAtLastLine ( ) const

Retrieve whether the maximum scroll position has the last line at the bottom of the view.

int wxStyledTextCtrl::GetEndStyled ( ) const

Retrieve the position of the last correctly styled character.

int wxStyledTextCtrl::GetEOLMode ( ) const

Retrieve the current end of line mode - one of CRLF, CR, or LF.

int wxStyledTextCtrl::GetExtraAscent ( ) const

Get extra ascent for each line.

int wxStyledTextCtrl::GetExtraDescent ( ) const

Get extra descent for each line.

int wxStyledTextCtrl::GetFirstVisibleLine ( ) const

Retrieve the display line at the top of the display.

bool wxStyledTextCtrl::GetFoldExpanded (int *line*) const

Is a header line expanded?

int wxStyledTextCtrl::GetFoldLevel (int *line*) const

Retrieve the fold level of a line.

int wxStyledTextCtrl::GetFoldParent (int *line*) const

Find the parent line of a child line.

int wxStyledTextCtrl::GetGapPosition ( ) const

Return a position which, to avoid performance costs, should not be within the range of a call to GetRangePointer.

int wxStyledTextCtrl::GetHighlightGuide ( ) const

Get the highlighted indentation guide column.

wxColour wxStyledTextCtrl::GetHotspotActiveBackground ( ) const

Get the back colour for active hotspots.

wxColour wxStyledTextCtrl::GetHotspotActiveForeground ( ) const

Get the fore colour for active hotspots.

bool wxStyledTextCtrl::GetHotspotActiveUnderline ( ) const

Get whether underlining for active hotspots.

bool wxStyledTextCtrl::GetHotspotSingleLine ( ) const

Get the HotspotSingleLine property.

int wxStyledTextCtrl::GetIdentifier ( ) const

Get the identifier.

int wxStyledTextCtrl::GetIndent ( ) const

Retrieve indentation size.

int wxStyledTextCtrl::GetIndentationGuides ( ) const

Are the indentation guides visible?

int wxStyledTextCtrl::GetIndicatorCurrent ( ) const

Get the current indicator.

int wxStyledTextCtrl::GetIndicatorValue ( ) const

Get the current indicator value.

virtual long wxStyledTextCtrl::GetInsertionPoint ( ) const `[virtual]`

Returns the insertion point, or cursor, position.

This is defined as the zero based index of the character position to the right of the insertion point. For example, if the insertion point is at the end of the single-line text control, it is equal to GetLastPosition().

Notice that insertion position is, in general, different from the index of the character the cursor position at in the string returned by GetValue(). While this is always the case for the single line controls, multi-line controls can use two characters "\\r\\n" as line separator (this is notably the case under MSW) meaning that indices in the control and its string value are offset by 1 for every line.

Hence to correctly get the character at the current cursor position, taking into account that there can be none if the cursor is at the end of the string, you could do the following:

```
wxString GetCurrentChar(wxTextCtrl *tc)
{
    long pos = tc->GetInsertionPoint();
    if ( pos == tc->GetLastPosition() )
        return wxString();

    return tc->GetRange(pos, pos + 1);
}
```

Reimplemented from wxTextEntry.

bool wxStyledTextCtrl::GetKeysUnicode ( ) const

Are keys always interpreted as Unicode?

int wxStyledTextCtrl::GetLastChild (int *line,* int *level*) const

Find the last child line of a header line.

bool wxStyledTextCtrl::GetLastKeydownProcessed ( )

Can be used to prevent the EVT_CHAR handler from adding the char.

virtual long wxStyledTextCtrl::GetLastPosition ( ) const [virtual]

Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.

Reimplemented from wxTextEntry.

int wxStyledTextCtrl::GetLayoutCache ( ) const

Retrieve the degree of caching of layout information.

int wxStyledTextCtrl::GetLength ( ) const

Returns the number of bytes in the document.

int wxStyledTextCtrl::GetLexer ( ) const

Retrieve the lexing language of the document.

static wxVersionInfo wxStyledTextCtrl::GetLibraryVersionInfo ( ) [static]

wxString wxStyledTextCtrl::GetLine (int *line*) const

Retrieve the contents of a line.

int wxStyledTextCtrl::GetLineCount ( ) const

Returns the number of lines in the document.

There is always at least one.

int wxStyledTextCtrl::GetLineEndPosition (int *line*) const

Get the position after the last visible characters on a line.

int wxStyledTextCtrl::GetLineEndTypesActive ( ) const

Get the line end types currently recognised.

May be a subset of the allowed types due to lexer limitation.

int wxStyledTextCtrl::GetLineEndTypesAllowed ( ) const

Get the line end types currently allowed.

int wxStyledTextCtrl::GetLineEndTypesSupported ( ) const

Bit set of LineEndType enumertion for which line ends beyond the standard LF, CR, and CRLF are supported by the lexer.

int wxStyledTextCtrl::GetLineIndentation (int *line*) const

Retrieve the number of columns that a line is indented.

int wxStyledTextCtrl::GetLineIndentPosition (int *line*) const

Retrieve the position before the first non indentation character on a line.

virtual int wxStyledTextCtrl::GetLineLength (long *n*) const `[virtual]`

wxCharBuffer wxStyledTextCtrl::GetLineRaw (int *line*)

Retrieve the contents of a line.

int wxStyledTextCtrl::GetLineSelEndPosition (int *line*)

Retrieve the position of the end of the selection at the given line (INVALID_POSITION if no selection on this line).

int wxStyledTextCtrl::GetLineSelStartPosition (int *line*)

Retrieve the position of the start of the selection at the given line (INVALID_POSITION if no selection on this line).

int wxStyledTextCtrl::GetLineState (int *line*) const

Retrieve the extra styling information for a line.

virtual wxString wxStyledTextCtrl::GetLineText (long *n*) const `[virtual]`

bool wxStyledTextCtrl::GetLineVisible (int *line*) const

Is a line visible?

int wxStyledTextCtrl::GetMainSelection ( ) const

Which selection is the main selection.

int wxStyledTextCtrl::GetMarginCursor (int *margin*) const

Retrieve the cursor shown in a margin.

int wxStyledTextCtrl::GetMarginLeft ( ) const

Returns the size in pixels of the left margin.

int wxStyledTextCtrl::GetMarginMask (int *margin*) const

Retrieve the marker mask of a margin.

int wxStyledTextCtrl::GetMarginOptions ( ) const

Get the margin options.

int wxStyledTextCtrl::GetMarginRight ( ) const

Returns the size in pixels of the right margin.

bool wxStyledTextCtrl::GetMarginSensitive (int *margin*) const

Retrieve the mouse click sensitivity of a margin.

int wxStyledTextCtrl::GetMarginType (int *margin*) const

Retrieve the type of a margin.

int wxStyledTextCtrl::GetMarginWidth (int *margin*) const

Retrieve the width of a margin in pixels.

int wxStyledTextCtrl::GetMarkerSymbolDefined (int *markerNumber*)

Which symbol was defined for markerNumber with MarkerDefine.

int wxStyledTextCtrl::GetMaxLineState ( ) const

Retrieve the last line number that has line state.

int wxStyledTextCtrl::GetModEventMask ( ) const

Get which document modification events are sent to the container.

bool wxStyledTextCtrl::GetModify ( ) const

Is the document different from when it was last saved?

bool wxStyledTextCtrl::GetMouseDownCaptures ( ) const

Get whether mouse gets captured.

int wxStyledTextCtrl::GetMouseDwellTime ( ) const

Retrieve the time the mouse must sit still to generate a mouse dwell event.

bool wxStyledTextCtrl::GetMouseSelectionRectangularSwitch ( ) const

Whether switching to rectangular mode while selecting with the mouse is allowed.

int wxStyledTextCtrl::GetMultiPaste ( ) const

Retrieve the effect of pasting when there are multiple selections.

bool wxStyledTextCtrl::GetMultipleSelection ( ) const

Whether multiple selections can be made.

virtual int wxStyledTextCtrl::GetNumberOfLines ( ) const `[virtual]`

bool wxStyledTextCtrl::GetOvertype ( ) const

Returns true if overtype mode is active otherwise false is returned.

bool wxStyledTextCtrl::GetPasteConvertEndings ( ) const

Get convert-on-paste setting.

int wxStyledTextCtrl::GetPositionCacheSize ( ) const

How many entries are allocated to the position cache?

int wxStyledTextCtrl::GetPrimaryStyleFromStyle (int *style*) const

For a secondary style, return the primary style, else return the argument.

int wxStyledTextCtrl::GetPrintColourMode ( ) const

Returns the print colour mode.

int wxStyledTextCtrl::GetPrintMagnification ( ) const

Returns the print magnification.

int wxStyledTextCtrl::GetPrintWrapMode ( ) const

Is printing line wrapped?

wxString wxStyledTextCtrl::GetProperty (const wxString & *key*)

Retrieve a 'property' value previously set with SetProperty.

wxString wxStyledTextCtrl::GetPropertyExpanded (const wxString & *key*)

Retrieve a 'property' value previously set with SetProperty, with '$()' variable replacement on returned buffer.

int wxStyledTextCtrl::GetPropertyInt (const wxString & *key*) const

Retrieve a 'property' value previously set with SetProperty, interpreted as an int AFTER any '$()' variable replacement.

wxString wxStyledTextCtrl::GetPunctuationChars ( ) const

Get the set of characters making up punctuation characters.

const char* wxStyledTextCtrl::GetRangePointer (int *position,* int *rangeLength*) const

Return a read-only pointer to a range of characters in the document.

May move the gap so that the range is contiguous, but will only move up to rangeLength bytes.

bool wxStyledTextCtrl::GetReadOnly ( ) const

In read-only mode?

int wxStyledTextCtrl::GetRectangularSelectionAnchor ( ) const

int wxStyledTextCtrl::GetRectangularSelectionAnchorVirtualSpace ( ) const

int wxStyledTextCtrl::GetRectangularSelectionCaret ( ) const

int wxStyledTextCtrl::GetRectangularSelectionCaretVirtualSpace ( ) const

int wxStyledTextCtrl::GetRectangularSelectionModifier ( ) const

Get the modifier key used for rectangular selection.

wxString wxStyledTextCtrl::GetRepresentation (const wxString & *encodedCharacter*) const

Set the way a character is drawn.

int wxStyledTextCtrl::GetScrollWidth ( ) const

Retrieve the document width assumed for scrolling.

bool wxStyledTextCtrl::GetScrollWidthTracking ( ) const

Retrieve whether the scroll width tracks wide lines.

int wxStyledTextCtrl::GetSearchFlags ( ) const

Get the search flags used by SearchInTarget.

int wxStyledTextCtrl::GetSelAlpha ( ) const

Get the alpha of the selection.

wxString wxStyledTextCtrl::GetSelectedText ( )

Retrieve the selected text.

wxCharBuffer wxStyledTextCtrl::GetSelectedTextRaw ( )

Retrieve the selected text.

virtual void wxStyledTextCtrl::GetSelection (long * *from,* long * *to*) const `[virtual]`

Gets the current selection span.

If the returned values are equal, there was no selection. Please note that the indices returned may be used with the other wxTextCtrl methods but don't necessarily represent the correct indices into the string returned by GetValue() for multiline controls under Windows (at least,) you should use GetStringSelection() to get the selected text.

Parameters

from	The returned first position.
to	The returned last position.

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (from, to).

Reimplemented from wxTextEntry.

bool wxStyledTextCtrl::GetSelectionEmpty ( ) const

Is every selected range empty?

int wxStyledTextCtrl::GetSelectionEnd ( ) const

Returns the position at the end of the selection.

int wxStyledTextCtrl::GetSelectionMode ( ) const

Get the mode of the current selection.

int wxStyledTextCtrl::GetSelectionNAnchor (int *selection*) const

int wxStyledTextCtrl::GetSelectionNAnchorVirtualSpace (int *selection*) const

int wxStyledTextCtrl::GetSelectionNCaret (int *selection*) const

int wxStyledTextCtrl::GetSelectionNCaretVirtualSpace (int *selection*) const

int wxStyledTextCtrl::GetSelectionNEnd (int *selection*) const

Returns the position at the end of the selection.

int wxStyledTextCtrl::GetSelectionNStart (int *selection*) const

Returns the position at the start of the selection.

int wxStyledTextCtrl::GetSelections ( ) const

How many selections are there?

int wxStyledTextCtrl::GetSelectionStart ( ) const

Returns the position at the start of the selection.

bool wxStyledTextCtrl::GetSelEOLFilled ( ) const

Is the selection end of line filled?

int wxStyledTextCtrl::GetStatus ( ) const

Get error status.

int wxStyledTextCtrl::GetSTCCursor ( ) const

Get cursor type.

bool wxStyledTextCtrl::GetSTCFocus ( ) const

Get internal focus flag.

virtual bool wxStyledTextCtrl::GetStyle (long *position,* wxTextAttr & *style*) `[virtual]`

int wxStyledTextCtrl::GetStyleAt (int *pos*) const

Returns the style byte at the position.

int wxStyledTextCtrl::GetStyleBits ( ) const

Retrieve number of bits in style bytes used to hold the lexical state.

int wxStyledTextCtrl::GetStyleBitsNeeded ( ) const

Retrieve the number of bits the current lexer needs for styling.

wxMemoryBuffer wxStyledTextCtrl::GetStyledText (int *startPos,* int *endPos*)

Retrieve a buffer of cells.

int wxStyledTextCtrl::GetStyleFromSubStyle (int *subStyle*) const

For a sub style, return the base style, else return the argument.

wxString wxStyledTextCtrl::GetSubStyleBases ( ) const

Get the set of base styles that can be extended with sub styles.

int wxStyledTextCtrl::GetSubStylesLength (int *styleBase*) const

The number of sub styles associated with a base style.

int wxStyledTextCtrl::GetSubStylesStart (int *styleBase*) const

The starting style number for the sub styles associated with a base style.

bool wxStyledTextCtrl::GetTabIndents ( ) const

Does a tab pressed when caret is within indentation indent?

int wxStyledTextCtrl::GetTabWidth ( ) const

Retrieve the visible size of a tab.

wxString wxStyledTextCtrl::GetTag (int *tagNumber*) const

Retrieve the value of a tag from a regular expression search.

int wxStyledTextCtrl::GetTargetEnd ( ) const

Get the position that ends the target.

int wxStyledTextCtrl::GetTargetStart ( ) const

Get the position that starts the target.

int wxStyledTextCtrl::GetTechnology ( ) const

Get the tech.

wxString wxStyledTextCtrl::GetText ( ) const

Retrieve all the text in the document.

int wxStyledTextCtrl::GetTextLength ( ) const

Retrieve the number of characters in the document.

wxString wxStyledTextCtrl::GetTextRange (int *startPos,* int *endPos*)

Retrieve a range of text.

wxCharBuffer wxStyledTextCtrl::GetTextRangeRaw (int *startPos,* int *endPos*)

Retrieve a range of text.

wxCharBuffer wxStyledTextCtrl::GetTextRaw ( )

Retrieve all the text in the document.

bool wxStyledTextCtrl::GetTwoPhaseDraw ( ) const

Is drawing done in two phases with backgrounds drawn before faoregrounds?

bool wxStyledTextCtrl::GetUndoCollection ( ) const

Is undo history being collected?

bool wxStyledTextCtrl::GetUseHorizontalScrollBar ( ) const

Is the horizontal scroll bar visible?

bool wxStyledTextCtrl::GetUseTabs ( ) const

Retrieve whether tabs will be used in indentation.

bool wxStyledTextCtrl::GetUseVerticalScrollBar ( ) const

Is the vertical scroll bar visible?

bool wxStyledTextCtrl::GetViewEOL ( ) const

Are the end of line characters visible?

int wxStyledTextCtrl::GetViewWhiteSpace ( ) const

Are white space characters currently visible? Returns one of SCWS_∗ constants.

int wxStyledTextCtrl::GetVirtualSpaceOptions ( ) const

wxString wxStyledTextCtrl::GetWhitespaceChars ( ) const

Get the set of characters making up whitespace for when moving or selecting by word.

int wxStyledTextCtrl::GetWhitespaceSize ( ) const

Get the size of the dots used to mark space characters.

wxString wxStyledTextCtrl::GetWordChars ( ) const

Get the set of characters making up words for when moving or selecting by word.

int wxStyledTextCtrl::GetWrapIndentMode ( ) const

Retrieve how wrapped sublines are placed.
Default is fixed.

int wxStyledTextCtrl::GetWrapMode ( ) const

Retrieve whether text is word wrapped.

int wxStyledTextCtrl::GetWrapStartIndent ( ) const

Retrive the start indent for wrapped lines.

int wxStyledTextCtrl::GetWrapVisualFlags ( ) const

Retrive the display mode of visual flags for wrapped lines.

int wxStyledTextCtrl::GetWrapVisualFlagsLocation ( ) const

Retrive the location of visual flags for wrapped lines.

int wxStyledTextCtrl::GetXOffset ( ) const

int wxStyledTextCtrl::GetZoom ( ) const

Retrieve the zoom level.

void wxStyledTextCtrl::GotoLine (int *line*)

Set caret to start of a line and ensure it is visible.

void wxStyledTextCtrl::GotoPos (int *pos*)

Set caret to a position and ensure it is visible.

void wxStyledTextCtrl::HideLines (int *lineStart,* int *lineEnd*)

Make a range of lines invisible.

void wxStyledTextCtrl::HideSelection (bool *normal*)

Draw the selection in normal style or with selection highlighted.

virtual wxTextCtrlHitTestResult wxStyledTextCtrl::HitTest (const wxPoint & *pt,* long * *pos*) const `[virtual]`

virtual wxTextCtrlHitTestResult wxStyledTextCtrl::HitTest (const wxPoint & *pt,* wxTextCoord * *col,* wxTextCoord * *row*) const `[virtual]`

void wxStyledTextCtrl::Home ( )

Move caret to first position on line.

void wxStyledTextCtrl::HomeDisplay ( )

Move caret to first position on display line.

void wxStyledTextCtrl::HomeDisplayExtend ( )

Move caret to first position on display line extending selection to new caret position.

void wxStyledTextCtrl::HomeExtend ( )

Move caret to first position on line extending selection to new caret position.

void wxStyledTextCtrl::HomeRectExtend ( )

Move caret to first position on line, extending rectangular selection to new caret position.

void wxStyledTextCtrl::HomeWrap ( )

These are like their namesakes Home(Extend)?, LineEnd(Extend)?, VCHome(Extend)? except they behave differently when word-wrap is enabled: They go first to the start / end of the display line, like (Home|LineEnd)Display The difference is that, the cursor is already at the point, it goes on to the start or end of the document line, as appropriate for (Home|LineEnd|VCHome)(Extend)?.

void wxStyledTextCtrl::HomeWrapExtend ( )

int wxStyledTextCtrl::IndicatorAllOnFor (int *position*)

Are any indicators present at position?

void wxStyledTextCtrl::IndicatorClearRange (int *position,* int *clearLength*)

Turn a indicator off over a range.

int wxStyledTextCtrl::IndicatorEnd (int *indicator,* int *position*)

Where does a particular indicator end?

void wxStyledTextCtrl::IndicatorFillRange (int *position,* int *fillLength*)

Turn a indicator on over a range.

int wxStyledTextCtrl::IndicatorGetAlpha (int *indicator*) const

Get the alpha fill colour of the given indicator.

wxColour wxStyledTextCtrl::IndicatorGetForeground (int *indic*) const

Retrieve the foreground colour of an indicator.

int wxStyledTextCtrl::IndicatorGetOutlineAlpha (int *indicator*) const

Get the alpha outline colour of the given indicator.

int wxStyledTextCtrl::IndicatorGetStyle (int *indic*) const

Retrieve the style of an indicator.

bool wxStyledTextCtrl::IndicatorGetUnder (int *indic*) const

Retrieve whether indicator drawn under or over text.

void wxStyledTextCtrl::IndicatorSetAlpha (int *indicator,* int *alpha*)

Set the alpha fill colour of the given indicator.

void wxStyledTextCtrl::IndicatorSetForeground (int *indic,* const wxColour & *fore*)

Set the foreground colour of an indicator.

void wxStyledTextCtrl::IndicatorSetOutlineAlpha (int *indicator,* int *alpha*)

Set the alpha outline colour of the given indicator.

void wxStyledTextCtrl::IndicatorSetStyle (int *indic,* int *style*)

Set an indicator to plain, squiggle or TT.

void wxStyledTextCtrl::IndicatorSetUnder (int *indic,* bool *under*)

Set an indicator to draw under text or over(default).

int wxStyledTextCtrl::IndicatorStart (int *indicator,* int *position*)

Where does a particular indicator start?

int wxStyledTextCtrl::IndicatorValueAt (int *indicator,* int *position*)

What value does a particular indicator have at at a position?

void wxStyledTextCtrl::InsertText (int *pos,* const wxString & *text*)

Insert string at a position.

void wxStyledTextCtrl::InsertTextRaw (int *pos,* const char * *text*)

Insert string at a position.

virtual bool wxStyledTextCtrl::IsEditable ( ) const `[virtual]`

Returns true if the controls contents may be edited by user (note that it always can be changed by the program).

In other words, this functions returns true if the control hasn't been put in read-only mode by a previous call to SetEditable().

Reimplemented from wxTextEntry.

virtual bool wxStyledTextCtrl::IsModified ( ) const `[virtual]`

void wxStyledTextCtrl::LineCopy ( )

Copy the line containing the caret.

void wxStyledTextCtrl::LineCut ( )

Cut the line containing the caret.

void wxStyledTextCtrl::LineDelete ( )

Delete the line containing the caret.

void wxStyledTextCtrl::LineDown ( )

Move caret down one line.

void wxStyledTextCtrl::LineDownExtend ( )

Move caret down one line extending selection to new caret position.

void wxStyledTextCtrl::LineDownRectExtend ( )

Move caret down one line, extending rectangular selection to new caret position.

void wxStyledTextCtrl::LineDuplicate ( )

Duplicate the current line.

void wxStyledTextCtrl::LineEnd ( )

Move caret to last position on line.

void wxStyledTextCtrl::LineEndDisplay ( )

Move caret to last position on display line.

void wxStyledTextCtrl::LineEndDisplayExtend ( )

Move caret to last position on display line extending selection to new caret position.

void wxStyledTextCtrl::LineEndExtend ( )

Move caret to last position on line extending selection to new caret position.

void wxStyledTextCtrl::LineEndRectExtend ( )

Move caret to last position on line, extending rectangular selection to new caret position.

void wxStyledTextCtrl::LineEndWrap ( )

void wxStyledTextCtrl::LineEndWrapExtend ( )

int wxStyledTextCtrl::LineFromPosition (int *pos*) const

Retrieve the line containing a position.

int wxStyledTextCtrl::LineLength (int *line*) const

How many characters are on a line, including end of line characters?

void wxStyledTextCtrl::LineScroll (int *columns,* int *lines*)

Scroll horizontally and vertically.

void wxStyledTextCtrl::LineScrollDown ( )

Scroll the document down, keeping the caret visible.

void wxStyledTextCtrl::LineScrollUp ( )

Scroll the document up, keeping the caret visible.

void wxStyledTextCtrl::LinesJoin ( )

Join the lines in the target.

int wxStyledTextCtrl::LinesOnScreen ( ) const

Retrieves the number of lines completely visible.

void wxStyledTextCtrl::LinesSplit (int *pixelWidth*)

Split the lines in the target into lines that are less wide than pixelWidth where possible.

void wxStyledTextCtrl::LineTranspose ( )

Switch the current line with the previous.

void wxStyledTextCtrl::LineUp ( )

Move caret up one line.

void wxStyledTextCtrl::LineUpExtend ( )

Move caret up one line extending selection to new caret position.

void wxStyledTextCtrl::LineUpRectExtend ( )

Move caret up one line, extending rectangular selection to new caret position.

bool wxStyledTextCtrl::LoadFile (const wxString & *filename*)

Load the contents of filename into the editor.

void wxStyledTextCtrl::LowerCase ( )

Transform the selection to lower case.

int wxStyledTextCtrl::MarginGetStyle (int *line*) const

Get the style number for the text margin for a line.

int wxStyledTextCtrl::MarginGetStyleOffset ( ) const

Get the start of the range of style numbers used for margin text.

wxString wxStyledTextCtrl::MarginGetStyles (int *line*) const

Get the styles in the text margin for a line.

wxString wxStyledTextCtrl::MarginGetText (int *line*) const

Get the text in the text margin for a line.

void wxStyledTextCtrl::MarginSetStyle (int *line,* int *style*)

Set the style number for the text margin for a line.

void wxStyledTextCtrl::MarginSetStyleOffset (int *style*)

Get the start of the range of style numbers used for margin text.

void wxStyledTextCtrl::MarginSetStyles (int *line,* const wxString & *styles*)

Set the style in the text margin for a line.

void wxStyledTextCtrl::MarginSetText (int *line,* const wxString & *text*)

Set the text in the text margin for a line.

void wxStyledTextCtrl::MarginTextClearAll ( )

Clear the margin text on all lines.

virtual void wxStyledTextCtrl::MarkDirty ( ) [virtual]

int wxStyledTextCtrl::MarkerAdd (int *line,* int *markerNumber*)

Add a marker to a line, returning an ID which can be used to find or delete the marker.

void wxStyledTextCtrl::MarkerAddSet (int *line,* int *set*)

Add a set of markers to a line.

void wxStyledTextCtrl::MarkerDefine (int *markerNumber,* int *markerSymbol,* const wxColour & *foreground* = wxNullColour, const wxColour & *background* = wxNullColour)

Set the symbol used for a particular marker number, and optionally the fore and background colours.

void wxStyledTextCtrl::MarkerDefineBitmap (int *markerNumber,* const wxBitmap & *bmp*)

Define a marker from a bitmap.

void wxStyledTextCtrl::MarkerDefineRGBAImage (int *markerNumber,* const unsigned char * *pixels*)

Define a marker from RGBA data.

It has the width and height from RGBAImageSetWidth/Height

void wxStyledTextCtrl::MarkerDelete (int *line,* int *markerNumber*)

Delete a marker from a line.

void wxStyledTextCtrl::MarkerDeleteAll (int *markerNumber*)

Delete all markers with a particular number from all lines.

void wxStyledTextCtrl::MarkerDeleteHandle (int *handle*)

Delete a marker.

void wxStyledTextCtrl::MarkerEnableHighlight (bool *enabled*)

Enable/disable highlight for current folding bloc (smallest one that contains the caret)

int wxStyledTextCtrl::MarkerGet (int *line*)

Get a bit mask of all the markers set on a line.

int wxStyledTextCtrl::MarkerLineFromHandle (int *handle*)

Retrieve the line number at which a particular marker is located.

int wxStyledTextCtrl::MarkerNext (int *lineStart,* int *markerMask*)

Find the next line at or after lineStart that includes a marker in mask.
Return -1 when no more lines.

int wxStyledTextCtrl::MarkerPrevious (int *lineStart,* int *markerMask*)

Find the previous line before lineStart that includes a marker in mask.

void wxStyledTextCtrl::MarkerSetAlpha (int *markerNumber,* int *alpha*)

Set the alpha used for a marker that is drawn in the text area, not the margin.

void wxStyledTextCtrl::MarkerSetBackground (int *markerNumber,* const wxColour & *back*)

Set the background colour used for a particular marker number.

void wxStyledTextCtrl::MarkerSetBackgroundSelected (int *markerNumber,* const wxColour & *back*)

Set the background colour used for a particular marker number when its folding block is selected.

void wxStyledTextCtrl::MarkerSetForeground (int *markerNumber,* const wxColour & *fore*)

Set the foreground colour used for a particular marker number.

void wxStyledTextCtrl::MoveCaretInsideView ( )

Move the caret inside current view if it's not there already.

void wxStyledTextCtrl::MoveSelectedLinesDown ( )

Move the selected lines down one line, shifting the line below before the selection.

void wxStyledTextCtrl::MoveSelectedLinesUp ( )

Move the selected lines up one line, shifting the line above after the selection.

void wxStyledTextCtrl::NewLine ( )

Insert a new line, may use a CRLF, CR or LF depending on EOL mode.

void wxStyledTextCtrl::PageDown ( )

Move caret one page down.

void wxStyledTextCtrl::PageDownExtend ( )

Move caret one page down extending selection to new caret position.

void wxStyledTextCtrl::PageDownRectExtend ( )

Move caret one page down, extending rectangular selection to new caret position.

void wxStyledTextCtrl::PageUp ( )

Move caret one page up.

void wxStyledTextCtrl::PageUpExtend ( )

Move caret one page up extending selection to new caret position.

void wxStyledTextCtrl::PageUpRectExtend ( )

Move caret one page up, extending rectangular selection to new caret position.

void wxStyledTextCtrl::ParaDown ( )

Move caret between paragraphs (delimited by empty lines).

void wxStyledTextCtrl::ParaDownExtend ( )

void wxStyledTextCtrl::ParaUp ( )

void wxStyledTextCtrl::ParaUpExtend ( )

void wxStyledTextCtrl::Paste ( ) [virtual]

Paste the contents of the clipboard into the document replacing the selection.

Reimplemented from wxTextEntry.

wxPoint wxStyledTextCtrl::PointFromPosition (int *pos*)

Retrieve the point in the window where a position is displayed.

int wxStyledTextCtrl::PositionAfter (int *pos*)

Given a valid document position, return the next position taking code page into account.

Maximum value returned is the last position in the document.

int wxStyledTextCtrl::PositionBefore (int *pos*)

Given a valid document position, return the previous position taking code page into account.

Returns 0 if passed 0.

int wxStyledTextCtrl::PositionFromLine (int *line*) const

Retrieve the position at the start of a line.

int wxStyledTextCtrl::PositionFromPoint (wxPoint *pt*) const

Find the position from a point within the window.

int wxStyledTextCtrl::PositionFromPointClose (int *x*, int *y*)

Find the position from a point within the window but return INVALID_POSITION if not close to text.

int wxStyledTextCtrl::PositionRelative (int *pos*, int *relative*)

Given a valid document position, return a position that differs in a number of characters.

Returned value is always between 0 and last position in document.

virtual bool wxStyledTextCtrl::PositionToXY (long *pos*, long * *x*, long * *y*) const [virtual]

void* wxStyledTextCtrl::PrivateLexerCall (int *operation*, void * *pointer*)

For private communication between an application and a known lexer.

wxString wxStyledTextCtrl::PropertyNames ( ) const

Retrieve a '
' separated list of properties understood by the current lexer.

int wxStyledTextCtrl::PropertyType (const wxString & *name*)

Retrieve the type of a property.

void wxStyledTextCtrl::Redo ( ) [virtual]

Redoes the next action on the undo history.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::RegisterImage (int *type*, const wxBitmap & *bmp*)

Register an image for use in autocompletion lists.

void wxStyledTextCtrl::RegisterRGBAImage (int *type*, const unsigned char * *pixels*)

Register an RGBA image for use in autocompletion lists.

It has the width and height from RGBAImageSetWidth/Height

void wxStyledTextCtrl::ReleaseAllExtendedStyles ( )

Release all extended (>255) style numbers.

void wxStyledTextCtrl::ReleaseDocument (void * *docPointer*)

Release a reference to the document, deleting document if it fades to black.

virtual void wxStyledTextCtrl::Remove (long *from*, long *to*) [virtual]

Removes the text starting at the first given position up to (but not including) the character at the last position.

This function puts the current insertion point position at *to* as a side effect.

Parameters

from	The first position.
to	The last position.

Reimplemented from wxTextEntry.

virtual void wxStyledTextCtrl::Replace (long *from*, long *to*, const wxString & *value*) [virtual]

Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.

This function puts the current insertion point position at *to* as a side effect.

Parameters

from	The first position.
to	The last position.

value	The value to replace the existing text with.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::ReplaceSelection (const wxString & *text*)

Replace the selected text with the argument text.

int wxStyledTextCtrl::ReplaceTarget (const wxString & *text*)

Replace the target text with the argument text.

Text is counted so it can contain NULs. Returns the length of the replacement text.

int wxStyledTextCtrl::ReplaceTargetRE (const wxString & *text*)

Replace the target text with the argument text after \d processing.

Text is counted so it can contain NULs. Looks for \d where d is between 1 and 9 and replaces these with the strings matched in the last search operation which were surrounded by (and). Returns the length of the replacement text including any change caused by processing the \d patterns.

void wxStyledTextCtrl::RGBAImageSetHeight (int *height*)

Set the height for future RGBA image data.

void wxStyledTextCtrl::RGBAImageSetScale (int *scalePercent*)

Set the scale factor in percent for future RGBA image data.

void wxStyledTextCtrl::RGBAImageSetWidth (int *width*)

Set the width for future RGBA image data.

void wxStyledTextCtrl::RotateSelection ( )

Set the main selection to the next selection.

bool wxStyledTextCtrl::SaveFile (const wxString & *filename*)

Write the contents of the editor to filename.

void wxStyledTextCtrl::ScrollRange (int *secondary*, int *primary*)

Scroll the argument positions and the range between them into view giving priority to the primary position then the secondary position.

This may be used to make a search match visible.

void wxStyledTextCtrl::ScrollToColumn (int *column*)

Scroll enough to make the given column visible.

void wxStyledTextCtrl::ScrollToEnd ( )

Scroll to end of document.

void wxStyledTextCtrl::ScrollToLine (int *line*)

Scroll enough to make the given line visible.

void wxStyledTextCtrl::ScrollToStart ( )

Scroll to start of document.

void wxStyledTextCtrl::SearchAnchor ( )

Sets the current caret position to be the search anchor.

int wxStyledTextCtrl::SearchInTarget (const wxString & *text*)

Search for a counted string in the target and set the target to the found range.

Text is counted so it can contain NULs. Returns length of range or -1 for failure in which case target is not moved.

int wxStyledTextCtrl::SearchNext (int *flags,* const wxString & *text*)

Find some text starting at the search anchor.

Does not ensure the selection is visible.

int wxStyledTextCtrl::SearchPrev (int *flags,* const wxString & *text*)

Find some text starting at the search anchor and moving backwards.

Does not ensure the selection is visible.

void wxStyledTextCtrl::SelectAll ( ) [virtual]

Select all the text in the document.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::SelectionDuplicate ( )

Duplicate the selection.

If selection empty duplicate the line containing the caret.

bool wxStyledTextCtrl::SelectionIsRectangle ( ) const

Is the selection rectangular? The alternative is the more common stream selection.

virtual void wxStyledTextCtrl::SelectNone ( ) [virtual]

Deselects selected text in the control.

Since

2.9.5

Reimplemented from wxTextEntry.

wxIntPtr wxStyledTextCtrl::SendMsg (int *msg*, wxUIntPtr *wp* = 0, wxIntPtr *lp* = 0) const

Send a message to Scintilla.

void wxStyledTextCtrl::SetAdditionalCaretForeground (const wxColour & *fore*)

Set the foreground colour of additional carets.

void wxStyledTextCtrl::SetAdditionalCaretsBlink (bool *additionalCaretsBlink*)

Set whether additional carets will blink.

void wxStyledTextCtrl::SetAdditionalCaretsVisible (bool *additionalCaretsBlink*)

Set whether additional carets are visible.

void wxStyledTextCtrl::SetAdditionalSelAlpha (int *alpha*)

Set the alpha of the selection.

void wxStyledTextCtrl::SetAdditionalSelBackground (const wxColour & *back*)

Set the background colour of additional selections.

Must have previously called SetSelBack with non-zero first argument for this to have an effect.

void wxStyledTextCtrl::SetAdditionalSelectionTyping (bool *additionalSelectionTyping*)

Set whether typing can be performed into multiple selections.

void wxStyledTextCtrl::SetAdditionalSelForeground (const wxColour & *fore*)

Set the foreground colour of additional selections.

Must have previously called SetSelFore with non-zero first argument for this to have an effect.

void wxStyledTextCtrl::SetAnchor (int *posAnchor*)

Set the selection anchor to a position.

The anchor is the opposite end of the selection from the caret.

void wxStyledTextCtrl::SetAutomaticFold (int *automaticFold*)

Set automatic folding behaviours.

void wxStyledTextCtrl::SetBackSpaceUnIndents (bool *bsUnIndents*)

Sets whether a backspace pressed when caret is within indentation unindents.

void wxStyledTextCtrl::SetBufferedDraw (bool *buffered*)

If drawing is buffered then each line of text is drawn into a bitmap buffer before drawing it to the screen to avoid flicker.

void wxStyledTextCtrl::SetCaretForeground (const wxColour & *fore*)

Set the foreground colour of the caret.

void wxStyledTextCtrl::SetCaretLineBackAlpha (int *alpha*)

Set background alpha of the caret line.

void wxStyledTextCtrl::SetCaretLineBackground (const wxColour & *back*)

Set the colour of the background of the line containing the caret.

void wxStyledTextCtrl::SetCaretLineVisible (bool *show*)

Display the background of the line containing the caret in a different colour.

void wxStyledTextCtrl::SetCaretLineVisibleAlways (bool *alwaysVisible*)

Sets the caret line to always visible.

void wxStyledTextCtrl::SetCaretPeriod (int *periodMilliseconds*)

Get the time in milliseconds that the caret is on and off.

0 = steady on.

void wxStyledTextCtrl::SetCaretSticky (int *useCaretStickyBehaviour*)

Stop the caret preferred x position changing when the user types.

void wxStyledTextCtrl::SetCaretStyle (int *caretStyle*)

Set the style of the caret to be drawn.

void wxStyledTextCtrl::SetCaretWidth (int *pixelWidth*)

Set the width of the insert mode caret.

void wxStyledTextCtrl::SetCharsDefault ( )

Reset the set of characters for whitespace and word characters to the defaults.

void wxStyledTextCtrl::SetCodePage (int *codePage*)

Set the code page used to interpret the bytes of the document as characters.

void wxStyledTextCtrl::SetControlCharSymbol (int *symbol*)

Change the way control characters are displayed: If symbol is $<$ 32, keep the drawn way, else, use the given character.

void wxStyledTextCtrl::SetCurrentPos (int *pos*)

Sets the position of the caret.

virtual bool wxStyledTextCtrl::SetDefaultStyle (const wxTextAttr & *style*) `[virtual]`

void wxStyledTextCtrl::SetDocPointer (void * *docPointer*)

Change the document object used.

void wxStyledTextCtrl::SetEdgeColour (const wxColour & *edgeColour*)

Change the colour used in edge indication.

void wxStyledTextCtrl::SetEdgeColumn (int *column*)

Set the column number of the edge.

If text goes past the edge then it is highlighted.

void wxStyledTextCtrl::SetEdgeMode (int *mode*)

The edge may be displayed by a line (EDGE_LINE) or by highlighting text that goes beyond it (EDGE_BACKGR↩OUND) or not displayed at all (EDGE_NONE).

virtual void wxStyledTextCtrl::SetEditable (bool *editable*) `[virtual]`

Makes the text item editable or read-only, overriding the **wxTE_READONLY** flag.

Parameters

editable	If true, the control is editable. If false, the control is read-only.

See also

IsEditable()

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::SetEmptySelection (int *pos*)

Set caret to a position, while removing any existing selection.

void wxStyledTextCtrl::SetEndAtLastLine (bool *endAtLastLine*)

Sets the scroll range so that maximum scroll position has the last line at the bottom of the view (default).

Setting this to false allows scrolling one page below the last line.

void wxStyledTextCtrl::SetEOLMode (int *eolMode*)

Set the current end of line mode.

void wxStyledTextCtrl::SetExtraAscent (int *extraAscent*)

Set extra ascent for each line.

void wxStyledTextCtrl::SetExtraDescent (int *extraDescent*)

Set extra descent for each line.

void wxStyledTextCtrl::SetFirstVisibleLine (int *lineDisplay*)

Scroll so that a display line is at the top of the display.

void wxStyledTextCtrl::SetFoldExpanded (int *line,* bool *expanded*)

Show the children of a header line.

void wxStyledTextCtrl::SetFoldFlags (int *flags*)

Set some style options for folding.

void wxStyledTextCtrl::SetFoldLevel (int *line,* int *level*)

Set the fold level of a line.

This encodes an integer level along with flags indicating whether the line is a header and whether it is effectively white space.

void wxStyledTextCtrl::SetFoldMarginColour (bool *useSetting,* const wxColour & *back*)

Set the colours used as a chequerboard pattern in the fold margin.

void wxStyledTextCtrl::SetFoldMarginHiColour (bool *useSetting,* const wxColour & *fore*)

void wxStyledTextCtrl::SetHighlightGuide (int *column*)

Set the highlighted indentation guide column.

0 = no highlighted guide.

void wxStyledTextCtrl::SetHotspotActiveBackground (bool *useSetting,* const wxColour & *back*)

Set a back colour for active hotspots.

void wxStyledTextCtrl::SetHotspotActiveForeground (bool *useSetting,* const wxColour & *fore*)

Set a fore colour for active hotspots.

void wxStyledTextCtrl::SetHotspotActiveUnderline (bool *underline*)

Enable / Disable underlining active hotspots.

void wxStyledTextCtrl::SetHotspotSingleLine (bool *singleLine*)

Limit hotspots to single line so hotspots on two lines don't merge.

void wxStyledTextCtrl::SetHScrollBar (wxScrollBar * *bar*)

Set the horizontal scrollbar to use instead of the ont that's built-in.

void wxStyledTextCtrl::SetIdentifier (int *identifier*)

Set the identifier reported as idFrom in notification messages.

void wxStyledTextCtrl::SetIdentifiers (int *style,* const wxString & *identifiers*)

Set the identifiers that are shown in a particular style.

void wxStyledTextCtrl::SetIndent (int *indentSize*)

Set the number of spaces used for one level of indentation.

void wxStyledTextCtrl::SetIndentationGuides (int *indentView*)

Show or hide indentation guides.

void wxStyledTextCtrl::SetIndicatorCurrent (int *indicator*)

Set the indicator used for IndicatorFillRange and IndicatorClearRange.

void wxStyledTextCtrl::SetIndicatorValue (int *value*)

Set the value used for IndicatorFillRange.

virtual void wxStyledTextCtrl::SetInsertionPoint (long *pos*) `[virtual]`

Sets the insertion point at the given position.

Parameters

pos	Position to set, in the range from 0 to GetLastPosition() inclusive.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::SetKeysUnicode (bool *keysUnicode*)

Always interpret keyboard input as Unicode.

void wxStyledTextCtrl::SetKeyWords (int *keywordSet,* const wxString & *keyWords*)

Set up the key words used by the lexer.

void wxStyledTextCtrl::SetLastKeydownProcessed (bool *val*)

void wxStyledTextCtrl::SetLayoutCache (int *mode*)

Sets the degree of caching of layout information.

void wxStyledTextCtrl::SetLexer (int *lexer*)

Set the lexing language of the document.

void wxStyledTextCtrl::SetLexerLanguage (const wxString & *language*)

Set the lexing language of the document based on string name.

void wxStyledTextCtrl::SetLineEndTypesAllowed (int *lineEndBitSet*)

Set the line end types that the application wants to use.
May not be used if incompatible with lexer or encoding.

void wxStyledTextCtrl::SetLineIndentation (int *line,* int *indentSize*)

Change the indentation of a line to a number of columns.

void wxStyledTextCtrl::SetLineState (int *line,* int *state*)

Used to hold extra styling information for each line.

void wxStyledTextCtrl::SetMainSelection (int *selection*)

Set the main selection.

void wxStyledTextCtrl::SetMarginCursor (int *margin,* int *cursor*)

Set the cursor shown when the mouse is inside a margin.

void wxStyledTextCtrl::SetMarginLeft (int *pixelWidth*)

Sets the size in pixels of the left margin.

void wxStyledTextCtrl::SetMarginMask (int *margin,* int *mask*)

Set a mask that determines which markers are displayed in a margin.

void wxStyledTextCtrl::SetMarginOptions (int *marginOptions*)

Set the margin options.

void wxStyledTextCtrl::SetMarginRight (int *pixelWidth*)

Sets the size in pixels of the right margin.

void wxStyledTextCtrl::SetMargins (int *left,* int *right*)

Set the left and right margin in the edit area, measured in pixels.

void wxStyledTextCtrl::SetMarginSensitive (int *margin,* bool *sensitive*)

Make a margin sensitive or insensitive to mouse clicks.

void wxStyledTextCtrl::SetMarginType (int *margin,* int *marginType*)

Set a margin to be either numeric or symbolic.

void wxStyledTextCtrl::SetMarginWidth (int *margin,* int *pixelWidth*)

Set the width of a margin to a width expressed in pixels.

void wxStyledTextCtrl::SetModEventMask (int *mask*)

Set which document modification events are sent to the container.

void wxStyledTextCtrl::SetMouseDownCaptures (bool *captures*)

Set whether the mouse is captured when its button is pressed.

void wxStyledTextCtrl::SetMouseDwellTime (int *periodMilliseconds*)

Sets the time the mouse must sit still to generate a mouse dwell event.

void wxStyledTextCtrl::SetMouseSelectionRectangularSwitch (bool *mouseSelectionRectangularSwitch*)

Set whether switching to rectangular mode while selecting with the mouse is allowed.

void wxStyledTextCtrl::SetMultiPaste (int *multiPaste*)

Change the effect of pasting when there are multiple selections.

void wxStyledTextCtrl::SetMultipleSelection (bool *multipleSelection*)

Set whether multiple selections can be made.

void wxStyledTextCtrl::SetOvertype (bool *overtype*)

Set to overtype (true) or insert mode.

void wxStyledTextCtrl::SetPasteConvertEndings (bool *convert*)

Enable/Disable convert-on-paste for line endings.

void wxStyledTextCtrl::SetPositionCacheSize (int *size*)

Set number of entries in position cache.

void wxStyledTextCtrl::SetPrintColourMode (int *mode*)

Modify colours when printing for clearer printed text.

void wxStyledTextCtrl::SetPrintMagnification (int *magnification*)

Sets the print magnification added to the point size of each style for printing.

void wxStyledTextCtrl::SetPrintWrapMode (int *mode*)

Set printing to line wrapped (SC_WRAP_WORD) or not line wrapped (SC_WRAP_NONE).

void wxStyledTextCtrl::SetProperty (const wxString & *key,* const wxString & *value*)

Set up a value that may be used by a lexer for some optional feature.

void wxStyledTextCtrl::SetPunctuationChars (const wxString & *characters*)

Set the set of characters making up punctuation characters Should be called after SetWordChars.

void wxStyledTextCtrl::SetReadOnly (bool *readOnly*)

Set to read only or read write.

void wxStyledTextCtrl::SetRectangularSelectionAnchor (int *posAnchor*)

void wxStyledTextCtrl::SetRectangularSelectionAnchorVirtualSpace (int *space*)

void wxStyledTextCtrl::SetRectangularSelectionCaret (int *pos*)

void wxStyledTextCtrl::SetRectangularSelectionCaretVirtualSpace (int *space*)

void wxStyledTextCtrl::SetRectangularSelectionModifier (int *modifier*)

On GTK+, allow selecting the modifier key to use for mouse-based rectangular selection.

Often the window manager requires Alt+Mouse Drag for moving windows. Valid values are SCMOD_CTRL(default), SCMOD_ALT, or SCMOD_SUPER.

void wxStyledTextCtrl::SetRepresentation (const wxString & *encodedCharacter,* const wxString & *representation*)

Set the way a character is drawn.

void wxStyledTextCtrl::SetSavePoint ( )

Remember the current position in the undo history as the position at which the document was saved.

void wxStyledTextCtrl::SetScrollWidth (int *pixelWidth*)

Sets the document width assumed for scrolling.

void wxStyledTextCtrl::SetScrollWidthTracking (bool *tracking*)

Sets whether the maximum width line displayed is used to set scroll width.

void wxStyledTextCtrl::SetSearchFlags (int *flags*)

Set the search flags used by SearchInTarget.

void wxStyledTextCtrl::SetSelAlpha (int *alpha*)

Set the alpha of the selection.

void wxStyledTextCtrl::SetSelBackground (bool *useSetting,* const wxColour & *back*)

Set the background colour of the main and additional selections and whether to use this setting.

virtual void wxStyledTextCtrl::SetSelection (long *from,* long *to*) `[virtual]`

Selects the text starting at the first position up to (but not including) the character at the last position.

If both parameters are equal to -1 all text in the control is selected.

Notice that the insertion point will be moved to *from* by this function.

Parameters

from	The first position.
to	The last position.

See also

SelectAll()

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::SetSelectionEnd (int *pos*)

Sets the position that ends the selection - this becomes the currentPosition.

void wxStyledTextCtrl::SetSelectionMode (int *mode*)

Set the selection mode to stream (SC_SEL_STREAM) or rectangular (SC_SEL_RECTANGLE/SC_SEL_THIN) or by lines (SC_SEL_LINES).

void wxStyledTextCtrl::SetSelectionNAnchor (int *selection,* int *posAnchor*)

void wxStyledTextCtrl::SetSelectionNAnchorVirtualSpace (int *selection,* int *space*)

void wxStyledTextCtrl::SetSelectionNCaret (int *selection,* int *pos*)

void wxStyledTextCtrl::SetSelectionNCaretVirtualSpace (int *selection,* int *space*)

void wxStyledTextCtrl::SetSelectionNEnd (int *selection,* int *pos*)

Sets the position that ends the selection - this becomes the currentPosition.

void wxStyledTextCtrl::SetSelectionNStart (int *selection,* int *pos*)

Sets the position that starts the selection - this becomes the anchor.

void wxStyledTextCtrl::SetSelectionStart (int *pos*)

Sets the position that starts the selection - this becomes the anchor.

void wxStyledTextCtrl::SetSelEOLFilled (bool *filled*)

Set the selection to have its end of line filled or not.

void wxStyledTextCtrl::SetSelForeground (bool *useSetting,* const wxColour & *fore*)

Set the foreground colour of the main and additional selections and whether to use this setting.

void wxStyledTextCtrl::SetStatus (int *statusCode*)

Change error status - 0 = OK.

void wxStyledTextCtrl::SetSTCCursor (int *cursorType*)

Sets the cursor to one of the SC_CURSOR∗ values.

void wxStyledTextCtrl::SetSTCFocus (bool *focus*)

Change internal focus flag.

virtual bool wxStyledTextCtrl::SetStyle (long *start,* long *end,* const wxTextAttr & *style*) [virtual]

void wxStyledTextCtrl::SetStyleBits (int *bits*)

Divide each styling byte into lexical class bits (default: 5) and indicator bits (default: 3).

If a lexer requires more than 32 lexical states, then this is used to expand the possible states.

void wxStyledTextCtrl::SetStyleBytes (int *length,* char * *styleBytes*)

Set the styles for a segment of the document.

void wxStyledTextCtrl::SetStyling (int *length,* int *style*)

Change style from current styling position for length characters to a style and move the current styling position to after this newly styled segment.

void wxStyledTextCtrl::SetTabIndents (bool *tabIndents*)

Sets whether a tab pressed when caret is within indentation indents.

void wxStyledTextCtrl::SetTabWidth (int *tabWidth*)

Change the visible size of a tab to be a multiple of the width of a space character.

void wxStyledTextCtrl::SetTargetEnd (int *pos*)

Sets the position that ends the target which is used for updating the document without affecting the scroll position.

void wxStyledTextCtrl::SetTargetStart (int *pos*)

Sets the position that starts the target which is used for updating the document without affecting the scroll position.

void wxStyledTextCtrl::SetTechnology (int *technology*)

Set the technology used.

void wxStyledTextCtrl::SetText (const wxString & *text*)

Replace the contents of the document with the argument text.

void wxStyledTextCtrl::SetTextRaw (const char * *text*)

Replace the contents of the document with the argument text.

void wxStyledTextCtrl::SetTwoPhaseDraw (bool *twoPhase*)

In twoPhaseDraw mode, drawing is performed in two phases, first the background and then the foreground.

This avoids chopping off characters that overlap the next run.

void wxStyledTextCtrl::SetUndoCollection (bool *collectUndo*)

Choose between collecting actions into the undo history and discarding them.

void wxStyledTextCtrl::SetUseHorizontalScrollBar (bool *show*)

Show or hide the horizontal scroll bar.

void wxStyledTextCtrl::SetUseTabs (bool *useTabs*)

Indentation will only use space characters if useTabs is false, otherwise it will use a combination of tabs and spaces.

void wxStyledTextCtrl::SetUseVerticalScrollBar (bool *show*)

Show or hide the vertical scroll bar.

void wxStyledTextCtrl::SetViewEOL (bool *visible*)

Make the end of line characters visible or invisible.

void wxStyledTextCtrl::SetViewWhiteSpace (int *viewWS*)

Make white space characters invisible, always visible or visible outside indentation.

void wxStyledTextCtrl::SetVirtualSpaceOptions (int *virtualSpaceOptions*)

void wxStyledTextCtrl::SetVisiblePolicy (int *visiblePolicy,* int *visibleSlop*)

Set the way the display area is determined when a particular line is to be moved to by Find, FindNext, GotoLine, etc.

void wxStyledTextCtrl::SetVScrollBar (wxScrollBar * *bar*)

Set the vertical scrollbar to use instead of the ont that's built-in.

void wxStyledTextCtrl::SetWhitespaceBackground (bool *useSetting,* const wxColour & *back*)

Set the background colour of all whitespace and whether to use this setting.

void wxStyledTextCtrl::SetWhitespaceChars (const wxString & *characters*)

Set the set of characters making up whitespace for when moving or selecting by word.

Should be called after SetWordChars.

void wxStyledTextCtrl::SetWhitespaceForeground (bool *useSetting,* const wxColour & *fore*)

Set the foreground colour of all whitespace and whether to use this setting.

void wxStyledTextCtrl::SetWhitespaceSize (int *size*)

Set the size of the dots used to mark space characters.

void wxStyledTextCtrl::SetWordChars (const wxString & *characters*)

Set the set of characters making up words for when moving or selecting by word.

First sets defaults like SetCharsDefault.

void wxStyledTextCtrl::SetWrapIndentMode (int *mode*)

Sets how wrapped sublines are placed.

Default is fixed.

void wxStyledTextCtrl::SetWrapMode (int *mode*)

Sets whether text is word wrapped.

void wxStyledTextCtrl::SetWrapStartIndent (int *indent*)

Set the start indent for wrapped lines.

void wxStyledTextCtrl::SetWrapVisualFlags (int *wrapVisualFlags*)

Set the display mode of visual flags for wrapped lines.

void wxStyledTextCtrl::SetWrapVisualFlagsLocation (int *wrapVisualFlagsLocation*)

Set the location of visual flags for wrapped lines.

void wxStyledTextCtrl::SetXCaretPolicy (int *caretPolicy,* int *caretSlop*)

Set the way the caret is kept visible when going sideways.

The exclusion zone is given in pixels.

void wxStyledTextCtrl::SetXOffset (int *newOffset*)

Get and Set the xOffset (ie, horizontal scroll position).

void wxStyledTextCtrl::SetYCaretPolicy (int *caretPolicy,* int *caretSlop*)

Set the way the line the caret is on is kept visible.

The exclusion zone is given in lines.

void wxStyledTextCtrl::SetZoom (int *zoom*)

Set the zoom level.

This number of points is added to the size of all fonts. It may be positive to magnify or negative to reduce.

void wxStyledTextCtrl::ShowLines (int *lineStart,* int *lineEnd*)

Make a range of lines visible.

virtual void wxStyledTextCtrl::ShowPosition (long *pos*) `[virtual]`

void wxStyledTextCtrl::StartRecord ( )

Start notifying the container of all key presses and commands.

void wxStyledTextCtrl::StartStyling (int *pos,* int *mask*)

Set the current styling position to pos and the styling mask to mask.

The styling mask can be used to protect some bits in each styling byte from modification.

void wxStyledTextCtrl::StopRecord ( )

Stop notifying the container of all key presses and commands.

void wxStyledTextCtrl::StutteredPageDown ( )

Move caret to bottom of page, or one page down if already at bottom of page.

void wxStyledTextCtrl::StutteredPageDownExtend ( )

Move caret to bottom of page, or one page down if already at bottom of page, extending selection to new caret position.

void wxStyledTextCtrl::StutteredPageUp ( )

Move caret to top of page, or one page up if already at top of page.

void wxStyledTextCtrl::StutteredPageUpExtend ( )

Move caret to top of page, or one page up if already at top of page, extending selection to new caret position.

void wxStyledTextCtrl::StyleClearAll ( )

Clear all the styles and make equivalent to the global default style.

wxColour wxStyledTextCtrl::StyleGetBackground (int *style*) const

Get the background colour of a style.

bool wxStyledTextCtrl::StyleGetBold (int *style*) const

Get is a style bold or not.

int wxStyledTextCtrl::StyleGetCase (int *style*) const

Get is a style mixed case, or to force upper or lower case.

bool wxStyledTextCtrl::StyleGetChangeable (int *style*) const

Get is a style changeable or not (read only).
Experimental feature, currently buggy.

int wxStyledTextCtrl::StyleGetCharacterSet (int *style*) const

Get the character get of the font in a style.

bool wxStyledTextCtrl::StyleGetEOLFilled (int *style*) const

Get is a style to have its end of line filled or not.

wxString wxStyledTextCtrl::StyleGetFaceName (int *style*)

Get the font facename of a style.

wxFont wxStyledTextCtrl::StyleGetFont (int *style*)

Get the font of a style.

wxColour wxStyledTextCtrl::StyleGetForeground (int *style*) const

Get the foreground colour of a style.

bool wxStyledTextCtrl::StyleGetHotSpot (int *style*) const

Get is a style a hotspot or not.

bool wxStyledTextCtrl::StyleGetItalic (int *style*) const

Get is a style italic or not.

int wxStyledTextCtrl::StyleGetSize (int *style*) const

Get the size of characters of a style.

int wxStyledTextCtrl::StyleGetSizeFractional (int *style*) const

Get the size of characters of a style in points multiplied by 100.

bool wxStyledTextCtrl::StyleGetUnderline (int *style*) const

Get is a style underlined or not.

bool wxStyledTextCtrl::StyleGetVisible (int *style*) const

Get is a style visible or not.

int wxStyledTextCtrl::StyleGetWeight (int *style*) const

Get the weight of characters of a style.

void wxStyledTextCtrl::StyleResetDefault ( )

Reset the default style to its state at startup.

void wxStyledTextCtrl::StyleSetBackground (int *style,* const wxColour & *back*)

Set the background colour of a style.

void wxStyledTextCtrl::StyleSetBold (int *style,* bool *bold*)

Set a style to be bold or not.

void wxStyledTextCtrl::StyleSetCase (int *style,* int *caseForce*)

Set a style to be mixed case, or to force upper or lower case.

void wxStyledTextCtrl::StyleSetChangeable (int *style,* bool *changeable*)

Set a style to be changeable or not (read only).

Experimental feature, currently buggy.

void wxStyledTextCtrl::StyleSetCharacterSet (int *style,* int *characterSet*)

Set the character set of the font in a style.

Converts the Scintilla character set values to a wxFontEncoding.

void wxStyledTextCtrl::StyleSetEOLFilled (int *style,* bool *filled*)

Set a style to have its end of line filled or not.

void wxStyledTextCtrl::StyleSetFaceName (int *style,* const wxString & *fontName*)

Set the font of a style.

void wxStyledTextCtrl::StyleSetFont (int *styleNum,* wxFont & *font*)

Set style size, face, bold, italic, and underline attributes from a wxFont's attributes.

void wxStyledTextCtrl::StyleSetFontAttr (int *styleNum,* int *size,* const wxString & *faceName,* bool *bold,* bool *italic,* bool *underline,* wxFontEncoding *encoding* = wxFONTENCODING_DEFAULT)

Set all font style attributes at once.

void wxStyledTextCtrl::StyleSetFontEncoding (int *style,* wxFontEncoding *encoding*)

Set the font encoding to be used by a style.

void wxStyledTextCtrl::StyleSetForeground (int *style,* const wxColour & *fore*)

Set the foreground colour of a style.

void wxStyledTextCtrl::StyleSetHotSpot (int *style,* bool *hotspot*)

Set a style to be a hotspot or not.

void wxStyledTextCtrl::StyleSetItalic (int *style,* bool *italic*)

Set a style to be italic or not.

void wxStyledTextCtrl::StyleSetSize (int *style,* int *sizePoints*)

Set the size of characters of a style.

void wxStyledTextCtrl::StyleSetSizeFractional (int *style,* int *caseForce*)

Set the size of characters of a style.

Size is in points multiplied by 100.

void wxStyledTextCtrl::StyleSetSpec (int *styleNum,* const wxString & *spec*)

Extract style settings from a spec-string which is composed of one or more of the following comma separated elements:

bold turns on bold italic turns on italics fore:[name or #RRGGBB] sets the foreground colour back:[name or #RR↩ GGBB] sets the background colour face:[facename] sets the font face name to use size:[num] sets the font size in points eol turns on eol filling underline turns on underlining

void wxStyledTextCtrl::StyleSetUnderline (int *style,* bool *underline*)

Set a style to be underlined or not.

void wxStyledTextCtrl::StyleSetVisible (int *style,* bool *visible*)

Set a style to be visible or not.

void wxStyledTextCtrl::StyleSetWeight (int *style,* int *weight*)

Set the weight of characters of a style.

void wxStyledTextCtrl::SwapMainAnchorCaret ( )

Swap that caret and anchor of the main selection.

void wxStyledTextCtrl::Tab ( )

If selection is empty or all on one line replace the selection with a tab character.

If more than one line selected, indent the lines.

void wxStyledTextCtrl::TargetFromSelection ( )

Make the target range start and end be the same as the selection range start and end.

int wxStyledTextCtrl::TextHeight (int *line*)

Retrieve the height of a particular line of text in pixels.

int wxStyledTextCtrl::TextWidth (int *style,* const wxString & *text*)

Measure the pixel width of some text in a particular style.

NUL terminated text argument. Does not handle tab or control characters.

void wxStyledTextCtrl::ToggleCaretSticky ( )

Switch between sticky and non-sticky: meant to be bound to a key.

void wxStyledTextCtrl::ToggleFold (int *line*)

Switch a header line between expanded and contracted.

void wxStyledTextCtrl::Undo ( ) [virtual]

Undo one action in the undo history.

Reimplemented from wxTextEntry.

void wxStyledTextCtrl::UpperCase ( )

Transform the selection to upper case.

void wxStyledTextCtrl::UsePopUp (bool *allowPopUp*)

Set whether a pop up menu is displayed automatically when the user presses the wrong mouse button.

void wxStyledTextCtrl::UserListShow (int *listType,* const wxString & *itemList*)

Display a list of strings and send notification when user chooses one.

void wxStyledTextCtrl::VCHome ( )

Move caret to before first visible character on line.

If already there move to first character on line.

void wxStyledTextCtrl::VCHomeDisplay ( )

Move caret to before first visible character on display line.

If already there move to first character on display line.

void wxStyledTextCtrl::VCHomeDisplayExtend ( )

Like VCHomeDisplay but extending selection to new caret position.

void wxStyledTextCtrl::VCHomeExtend ( )

Like VCHome but extending selection to new caret position.

void wxStyledTextCtrl::VCHomeRectExtend ( )

Move caret to before first visible character on line.

If already there move to first character on line. In either case, extend rectangular selection to new caret position.

void wxStyledTextCtrl::VCHomeWrap ( )

void wxStyledTextCtrl::VCHomeWrapExtend ( )

void wxStyledTextCtrl::VerticalCentreCaret ( )

Centre current line in window.

int wxStyledTextCtrl::VisibleFromDocLine (int *line*)

Find the display line of a document line taking hidden lines into account.

int wxStyledTextCtrl::WordEndPosition (int *pos,* bool *onlyWordCharacters*)

Get position of end of word.

void wxStyledTextCtrl::WordLeft ( )

Move caret left one word.

void wxStyledTextCtrl::WordLeftEnd ( )

Move caret left one word, position cursor at end of word.

void wxStyledTextCtrl::WordLeftEndExtend ( )

Move caret left one word, position cursor at end of word, extending selection to new caret position.

void wxStyledTextCtrl::WordLeftExtend ( )

Move caret left one word extending selection to new caret position.

void wxStyledTextCtrl::WordPartLeft ( )

Move to the previous change in capitalisation.

void wxStyledTextCtrl::WordPartLeftExtend ( )

Move to the previous change in capitalisation extending selection to new caret position.

void wxStyledTextCtrl::WordPartRight ( )

Move to the change next in capitalisation.

void wxStyledTextCtrl::WordPartRightExtend ( )

Move to the next change in capitalisation extending selection to new caret position.

void wxStyledTextCtrl::WordRight ( )

Move caret right one word.

void wxStyledTextCtrl::WordRightEnd ( )

Move caret right one word, position cursor at end of word.

void wxStyledTextCtrl::WordRightEndExtend ( )

Move caret right one word, position cursor at end of word, extending selection to new caret position.

void wxStyledTextCtrl::WordRightExtend ( )

Move caret right one word extending selection to new caret position.

int wxStyledTextCtrl::WordStartPosition (int *pos,* bool *onlyWordCharacters*)

Get position of start of word.

int wxStyledTextCtrl::WrapCount (int *line*)

The number of display lines needed to wrap a document line.

virtual void wxStyledTextCtrl::WriteText (const wxString & *text*) `[virtual]`

Writes the text into the text control at the current insertion position.

Parameters

text	Text to write to the text control.

Remarks

Newlines in the text string are the only control characters allowed, and they will cause appropriate line breaks. See operator<<() and AppendText() for more convenient ways of writing to the window. After the write operation, the insertion point will be at the end of the inserted text, so subsequent write operations will be appended. To append text after the user may have interacted with the control, call wxTextCtrl::SetInsertionPointEnd() before writing.

Reimplemented from wxTextEntry.

virtual long wxStyledTextCtrl::XYToPosition (long *x*, long *y*) const `[virtual]`

void wxStyledTextCtrl::ZoomIn ( )

Magnify the displayed text by increasing the sizes by 1 point.

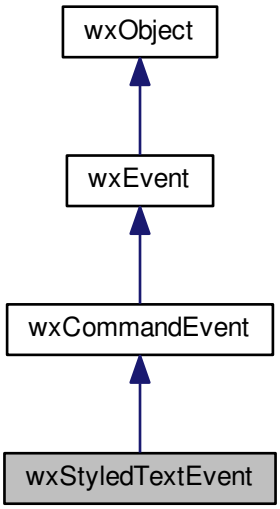
void wxStyledTextCtrl::ZoomOut ( )

Make the displayed text smaller by decreasing the sizes by 1 point.

21.738 wxStyledTextEvent Class Reference

`#include <wx/stc/stc.h>`

Inheritance diagram for wxStyledTextEvent:



21.738.1 Detailed Description

The type of events sent from wxStyledTextCtrl.

Todo list styled text ctrl events.

Library: wxSTC

Category: Events, Scintilla Text Editor

Public Member Functions

- wxStyledTextEvent (wxEventType commandType=0, int id=0)
- wxStyledTextEvent (const wxStyledTextEvent &event)
- ~wxStyledTextEvent ()
- void SetPosition (int pos)
- void SetKey (int k)
- void SetModifiers (int m)
- void SetModificationType (int t)
- void SetText (const wxString &t)
- void SetLength (int len)
- void SetLinesAdded (int num)
- void SetLine (int val)
- void SetFoldLevelNow (int val)
- void SetFoldLevelPrev (int val)
- void SetMargin (int val)
- void SetMessage (int val)
- void SetWParam (int val)
- void SetLParam (int val)
- void SetListType (int val)
- void SetX (int val)
- void SetY (int val)
- void SetToken (int val)
- void SetAnnotationLinesAdded (int val)
- void SetUpdated (int val)
- void SetDragText (const wxString &val)
- void SetDragFlags (int flags)
- void SetDragResult (wxDragResult val)
- int GetPosition () const
- int GetKey () const
- int GetModifiers () const
- int GetModificationType () const
- wxString GetText () const
- int GetLength () const
- int GetLinesAdded () const
- int GetLine () const
- int GetFoldLevelNow () const
- int GetFoldLevelPrev () const
- int GetMargin () const
- int GetMessage () const
- int GetWParam () const
- int GetLParam () const

- int GetListType () const
- int GetX () const
- int GetY () const
- int GetToken () const
- int GetAnnotationsLinesAdded () const
- int GetUpdated () const
- wxString GetDragText ()
- int GetDragFlags ()
- wxDragResult GetDragResult ()
- bool GetShift () const
- bool GetControl () const
- bool GetAlt () const

Additional Inherited Members

21.738.2 Constructor & Destructor Documentation

wxStyledTextEvent::wxStyledTextEvent (wxEventType *commandType* = 0, int *id* = 0)

wxStyledTextEvent::wxStyledTextEvent (const wxStyledTextEvent & *event*)

wxStyledTextEvent::~wxStyledTextEvent ( )

21.738.3 Member Function Documentation

bool wxStyledTextEvent::GetAlt ( ) const

int wxStyledTextEvent::GetAnnotationsLinesAdded ( ) const

bool wxStyledTextEvent::GetControl ( ) const

int wxStyledTextEvent::GetDragFlags ( )

wxDragResult wxStyledTextEvent::GetDragResult ( )

wxString wxStyledTextEvent::GetDragText ( )

Deprecated Use GetString() instead.

int wxStyledTextEvent::GetFoldLevelNow ( ) const

int wxStyledTextEvent::GetFoldLevelPrev ( ) const

int wxStyledTextEvent::GetKey ( ) const

int wxStyledTextEvent::GetLength ( ) const

int wxStyledTextEvent::GetLine ( ) const

int wxStyledTextEvent::GetLinesAdded ( ) const

int wxStyledTextEvent::GetListType ( ) const

int wxStyledTextEvent::GetLParam ( ) const

int wxStyledTextEvent::GetMargin ( ) const

int wxStyledTextEvent::GetMessage ( ) const

int wxStyledTextEvent::GetModificationType ( ) const

int wxStyledTextEvent::GetModifiers ( ) const

int wxStyledTextEvent::GetPosition ( ) const

bool wxStyledTextEvent::GetShift ( ) const

wxString wxStyledTextEvent::GetText ( ) const

Deprecated Use GetString() instead.

int wxStyledTextEvent::GetToken ( ) const

int wxStyledTextEvent::GetUpdated ( ) const

int wxStyledTextEvent::GetWParam ( ) const

int wxStyledTextEvent::GetX ( ) const

int wxStyledTextEvent::GetY ( ) const

void wxStyledTextEvent::SetAnnotationLinesAdded (int *val*)

void wxStyledTextEvent::SetDragFlags (int *flags*)

void wxStyledTextEvent::SetDragResult (wxDragResult *val*)

void wxStyledTextEvent::SetDragText (const wxString & *val*)

void wxStyledTextEvent::SetFoldLevelNow (int *val*)

void wxStyledTextEvent::SetFoldLevelPrev (int *val*)

void wxStyledTextEvent::SetKey (int *k*)

void wxStyledTextEvent::SetLength (int *len*)

void wxStyledTextEvent::SetLine (int *val*)

void wxStyledTextEvent::SetLinesAdded (int *num*)

void wxStyledTextEvent::SetListType (int *val*)

void wxStyledTextEvent::SetLParam (int *val*)

void wxStyledTextEvent::SetMargin (int *val*)

void wxStyledTextEvent::SetMessage (int *val*)

void wxStyledTextEvent::SetModificationType (int *t*)

void wxStyledTextEvent::SetModifiers (int *m*)

void wxStyledTextEvent::SetPosition (int *pos*)

void wxStyledTextEvent::SetText (const wxString & *t*)

void wxStyledTextEvent::SetToken (int *val*)

void wxStyledTextEvent::SetUpdated (int *val*)

void wxStyledTextEvent::SetWParam (int *val*)

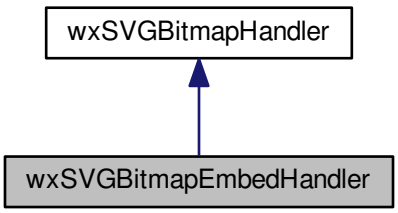
void wxStyledTextEvent::SetX (int *val*)

void wxStyledTextEvent::SetY (int *val*)

21.739 wxSVGBitmapEmbedHandler Class Reference

`#include <wx/dcsvg.h>`

Inheritance diagram for wxSVGBitmapEmbedHandler:



21.739.1 Detailed Description

Handler embedding bitmaps as base64-encoded PNGs into the SVG.

See also

wxSVGFileDC::SetBitmapHandler().

Library: wxCore

Category: Device Contexts

Since

3.1.0

Public Member Functions

- virtual bool ProcessBitmap (const wxBitmap &bitmap, wxCoord x, wxCoord y, wxOutputStream &stream) const

  Writes the bitmap representation as SVG to the given stream.

21.739.2 Member Function Documentation

virtual bool wxSVGBitmapEmbedHandler::ProcessBitmap (const wxBitmap & *bitmap*, wxCoord *x*, wxCoord *y*, wxOutputStream & *stream*) const `[virtual]`

Writes the bitmap representation as SVG to the given stream.

The XML generated by this function will be inserted into the SVG file inline with the XML generated by the main wxSVGFileDC class so it is important that the XML is properly formed.

Parameters

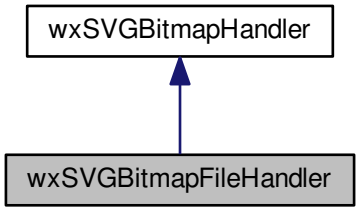
bitmap	A valid bitmap to add to SVG.
x	Horizontal position of the bitmap.
y	Vertical position of the bitmap.
stream	The stream to write SVG contents to.

Implements wxSVGBitmapHandler.

21.740 wxSVGBitmapFileHandler Class Reference

`#include <wx/dcsvg.h>`

Inheritance diagram for wxSVGBitmapFileHandler:



21.740.1 Detailed Description

Handler saving a bitmap to an external file and linking to it from the SVG.

This handler is used by default by wxSVGFileDC.

See also

wxSVGFileDC::SetBitmapHandler().

Library: wxCore

Category: Device Contexts

Since

3.1.0

Public Member Functions

- virtual bool ProcessBitmap (const wxBitmap &bitmap, wxCoord x, wxCoord y, wxOutputStream &stream) const

  Writes the bitmap representation as SVG to the given stream.

21.740.2 Member Function Documentation

virtual bool wxSVGBitmapFileHandler::ProcessBitmap (const wxBitmap & *bitmap*, wxCoord *x*, wxCoord *y*, wxOutputStream & *stream*) const `[virtual]`

Writes the bitmap representation as SVG to the given stream.

The XML generated by this function will be inserted into the SVG file inline with the XML generated by the main wxSVGFileDC class so it is important that the XML is properly formed.

Parameters

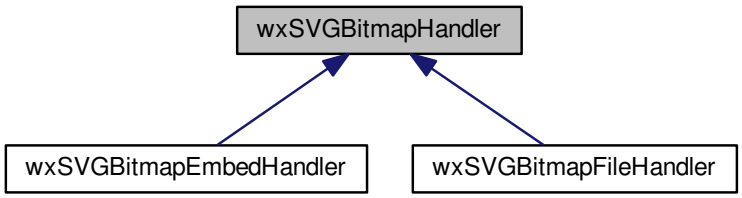
bitmap	A valid bitmap to add to SVG.
x	Horizontal position of the bitmap.
y	Vertical position of the bitmap.
stream	The stream to write SVG contents to.

Implements wxSVGBitmapHandler.

21.741 wxSVGBitmapHandler Class Reference

`#include <wx/dcsvg.h>`

Inheritance diagram for wxSVGBitmapHandler:



21.741.1 Detailed Description

Abstract base class for handling bitmaps inside a wxSVGFileDC.

To use it you need to derive a new class from it and override ProcessBitmap() to generate a properly a formed SVG image element (see http://www.w3.org/TR/SVG/struct.html#ImageElement).

Two example bitmap handlers are provided in wx/dcsvg.h. The first (default) handler will create PNG files in the same folder as the SVG file and uses links to them in the SVG. The second handler (wxSVGBitmapEmbedHandler) will embed the PNG image in the SVG file using base 64 encoding.

The handler can be changed by calling wxSVGFileDC::SetBitmapHandler().

Library: wxCore

Category: Device Contexts

Since

3.1.0

Public Member Functions

- virtual bool ProcessBitmap (const wxBitmap &bitmap, wxCoord x, wxCoord y, wxOutputStream &stream) const =0

  Writes the bitmap representation as SVG to the given stream.

21.741.2 Member Function Documentation

virtual bool wxSVGBitmapHandler::ProcessBitmap (const wxBitmap & *bitmap,* wxCoord *x,* wxCoord *y,* wxOutputStream & *stream*) const [pure virtual]

Writes the bitmap representation as SVG to the given stream.

The XML generated by this function will be inserted into the SVG file inline with the XML generated by the main wxSVGFileDC class so it is important that the XML is properly formed.

Parameters

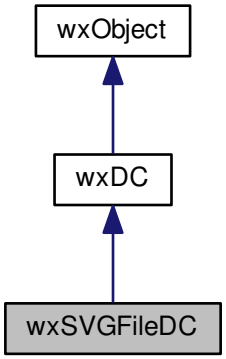
bitmap	A valid bitmap to add to SVG.
x	Horizontal position of the bitmap.
y	Vertical position of the bitmap.
stream	The stream to write SVG contents to.

Implemented in wxSVGBitmapFileHandler, and wxSVGBitmapEmbedHandler.

21.742 wxSVGFileDC Class Reference

```
#include <wx/dcsvg.h>
```

Inheritance diagram for wxSVGFileDC:



21.742.1 Detailed Description

A wxSVGFileDC is a device context onto which graphics and text can be drawn, and the output produced as a vector file, in SVG format.

This format can be read by a range of programs, including a Netscape plugin (Adobe) and the open source Inkscape program (http://inkscape.org/). Full details are given in the W3C SVG recommendation (http://www.w3.org/TR/SVG/).

The intention behind wxSVGFileDC is that it can be used to produce a file corresponding to the screen display context, wxSVGFileDC, by passing the wxSVGFileDC as a parameter instead of a wxDC. Thus the wxSVGFileDC is a write-only class.

As the wxSVGFileDC is a vector format, raster operations like GetPixel() are unlikely to be supported. However, the SVG specification allows for raster files to be embedded in the SVG, and so bitmaps, icons and blit operations in wxSVGFileDC are supported. By default only PNG format bitmaps are supported and these are saved as separate files in the same folder as the SVG file, however it is possible to change this behaviour by replacing the built in bitmap handler using wxSVGFileDC::SetBitmapHandler().

A more substantial SVG library (for reading and writing) is available at the wxArt2D website http://wxart2d.↩ sourceforge.net/.

Library: wxCore

Category: Device Contexts

Public Member Functions

- wxSVGFileDC (const wxString &filename, int width=320, int height=240, double dpi=72)

  Initializes a wxSVGFileDC with the given f filename with the given Width and Height at dpi resolution.
- virtual ~wxSVGFileDC ()

  Destructor.
- void EndDoc ()

  Does nothing.

- void EndPage ()

  Does nothing.
- void Clear ()

  This makes no sense in wxSVGFileDC and does nothing.
- void SetBitmapHandler (wxSVGBitmapHandler *handler)

  Replaces the default bitmap handler with handler.
- void SetLogicalFunction (wxRasterOperationMode function)

  Does the same as wxDC::SetLogicalFunction(), except that only wxCOPY is available.
- void SetClippingRegion (wxCoord x, wxCoord y, wxCoord width, wxCoord height)

  Sets the clipping region for this device context to the intersection of the given region described by the parameters of this method and the previously set clipping region.
- void SetClippingRegion (const wxPoint &pt, const wxSize &sz)

  This is an overloaded member function, provided for convenience.
- void SetClippingRegion (const wxRect &rect)

  This is an overloaded member function, provided for convenience.
- void SetClippingRegion (const wxRegion &region)

  This function is not implemented in this DC class.
- void DestroyClippingRegion ()

  Destroys the current clipping region so that none of the DC is clipped.

- void CrossHair (wxCoord x, wxCoord y)

  Functions not implemented in this DC class.
- bool FloodFill (wxCoord x, wxCoord y, const wxColour &colour, wxFloodFillStyle style=wxFLOOD_SURFACE)

  Functions not implemented in this DC class.
- void GetClippingBox (wxCoord *x, wxCoord *y, wxCoord *width, wxCoord *height) const

  Functions not implemented in this DC class.
- bool GetPixel (wxCoord x, wxCoord y, wxColour *colour) const

  Functions not implemented in this DC class.
- void SetPalette (const wxPalette &palette)

  Functions not implemented in this DC class.
- bool StartDoc (const wxString &message)

  Functions not implemented in this DC class.

Additional Inherited Members

21.742.2 Constructor & Destructor Documentation

wxSVGFileDC::wxSVGFileDC (const wxString & *filename*, int *width* = 320, int *height* = 240, double *dpi* = 72)

Initializes a wxSVGFileDC with the given *f* filename with the given *Width* and *Height* at *dpi* resolution.

virtual wxSVGFileDC::~wxSVGFileDC ( ) [virtual]

Destructor.

21.742.3 Member Function Documentation

void wxSVGFileDC::Clear ( )

This makes no sense in wxSVGFileDC and does nothing.

void wxSVGFileDC::CrossHair (wxCoord *x,* wxCoord *y*)

Functions not implemented in this DC class.

void wxSVGFileDC::DestroyClippingRegion ( )

Destroys the current clipping region so that none of the DC is clipped.

Since intersections arising from sequential calls to SetClippingRegion are represented with nested SVG group elements (<g>), all such groups are closed when DestroyClippingRegion is called.

void wxSVGFileDC::EndDoc ( )

Does nothing.

void wxSVGFileDC::EndPage ( )

Does nothing.

bool wxSVGFileDC::FloodFill (wxCoord *x,* wxCoord *y,* const wxColour & *colour,* wxFloodFillStyle *style* = wxFLOOD_SURFACE)

Functions not implemented in this DC class.

void wxSVGFileDC::GetClippingBox (wxCoord * *x,* wxCoord * *y,* wxCoord * *width,* wxCoord * *height*) const

Functions not implemented in this DC class.

bool wxSVGFileDC::GetPixel (wxCoord *x,* wxCoord *y,* wxColour * *colour*) const

Functions not implemented in this DC class.

void wxSVGFileDC::SetBitmapHandler (wxSVGBitmapHandler * *handler*)

Replaces the default bitmap handler with *handler*.

By default, an object of wxSVGBitmapFileHandler class is used as bitmap handler. You may want to replace it with an object of predefined wxSVGBitmapEmbedHandler class to embed the bitmaps in the generated SVG instead of storing them in separate files like this:

```
mySVGFileDC->SetBitmapHandler(new wxSVGBitmapEmbedHandler());
```

or derive your own bitmap handler class and use it if you need to customize the bitmap handling further.

Parameters

handler	The new bitmap handler. If non-NULL, this object takes ownership of this handler and will delete it when it is not needed any more.

Since

3.1.0

void wxSVGFileDC::SetClippingRegion (wxCoord *x,* wxCoord *y,* wxCoord *width,* wxCoord *height*)

Sets the clipping region for this device context to the intersection of the given region described by the parameters of this method and the previously set clipping region.

Clipping is implemented in the SVG output using SVG group elements (<g>), with nested group elements being used to represent clipping region intersections when two or more calls are made to SetClippingRegion().

void wxSVGFileDC::SetClippingRegion (const wxPoint & *pt,* const wxSize & *sz*)

This is an overloaded member function, provided for convenience.

It differs from the above function only in what argument(s) it accepts.

void wxSVGFileDC::SetClippingRegion (const wxRect & *rect*)

This is an overloaded member function, provided for convenience.

It differs from the above function only in what argument(s) it accepts.

void wxSVGFileDC::SetClippingRegion (const wxRegion & *region*)

This function is not implemented in this DC class.

It could be implemented in future if a GetPoints() function were made available on wxRegion.

void wxSVGFileDC::SetLogicalFunction (wxRasterOperationMode *function*)

Does the same as wxDC::SetLogicalFunction(), except that only wxCOPY is available.

Trying to set one of the other values will fail.

void wxSVGFileDC::SetPalette (const wxPalette & *palette*)

Functions not implemented in this DC class.

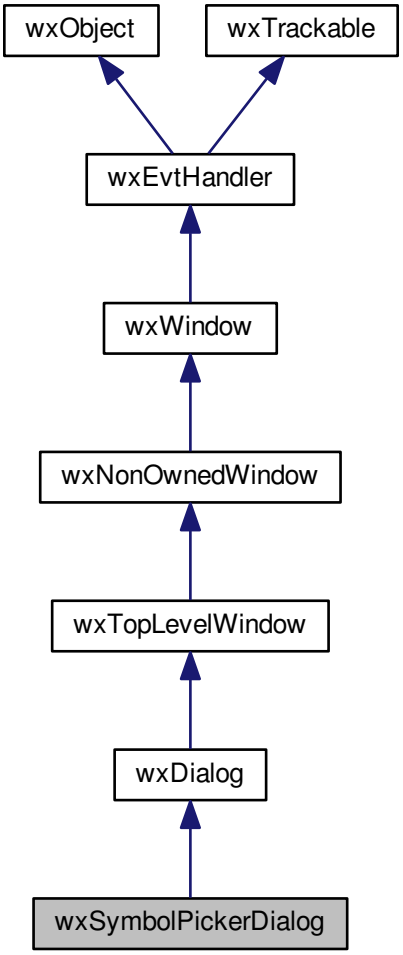
bool wxSVGFileDC::StartDoc (const wxString & *message*)

Functions not implemented in this DC class.

21.743 wxSymbolPickerDialog Class Reference

`#include <wx/richtext/richtextsymboldlg.h>`

Inheritance diagram for wxSymbolPickerDialog:



21.743.1 Detailed Description

wxSymbolPickerDialog presents the user with a choice of fonts and a grid of available characters.

This modal dialog provides the application with a selected symbol and optional font selection.

Although this dialog is contained in the rich text library, the dialog is generic and can be used in other contexts.

To use the dialog, pass a default symbol specified as a string, an initial font name, and a current font name. The difference between the initial font and current font is that the initial font determines what the font control will be set to when the dialog shows - an empty string will show the selection *normal* text. The current font, on the other hand, is used by the dialog to determine what font to display the characters in, even when no initial font is selected. This allows the user (and application) to distinguish between inserting a symbol in the current font, and inserting it with a specified font.

When the dialog is dismissed, the application can get the selected symbol with wxSymbolPickerDialog::GetSymbol and test whether a font was specified with wxSymbolPickerDialog::UseNormalFont,fetching the specified font with wxSymbolPickerDialog::GetFontName.

Here's a realistic example, inserting the supplied symbol into a rich text control in either the current font or specified

font.

```
wxRichTextCtrl* ctrl = (wxRichTextCtrl*) FindWindow(ID_RICHTEXT_CTRL)
      ;

wxTextAttr attr;
attr.SetFlags(wxTEXT_ATTR_FONT);
ctrl->GetStyle(ctrl->GetInsertionPoint(), attr);

wxString currentFontName;
if (attr.HasFont() && attr.GetFont().IsOk())
    currentFontName = attr.GetFont().GetFaceName();

// Don't set the initial font in the dialog (so the user is choosing
// 'normal text', i.e. the current font) but do tell the dialog
// what 'normal text' is.

wxSymbolPickerDialog dlg("*", wxEmptyString, currentFontName, this);

if (dlg.ShowModal() == wxID_OK)
{
    if (dlg.HasSelection())
    {
        long insertionPoint = ctrl->GetInsertionPoint();

        ctrl->WriteText(dlg.GetSymbol());

        if (!dlg.UseNormalFont())
        {
            wxFont font(attr.GetFont());
            font.SetFaceName(dlg.GetFontName());
            attr.SetFont(font);
            ctrl->SetStyle(insertionPoint, insertionPoint+1, attr);
        }
    }
}
```

Library: wxRichText

Category: Common Dialogs

Public Member Functions

- wxSymbolPickerDialog ()

  Default ctor.

- wxSymbolPickerDialog (const wxString &symbol, const wxString &initialFont, const wxString &normalTextFont, wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &title=_("Symbols"), const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxCLOSE_BOX)

  Constructor.

- bool Create (const wxString &symbol, const wxString &initialFont, const wxString &normalTextFont, wxWindow *parent, wxWindowID id=wxID_ANY, const wxString &caption=wxGetTranslation("Symbols"), const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxSize(400, 300), long style=wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxCLOSE_BOX)

  Creation: see the constructor for details about the parameters.

- wxString GetFontName () const

  Returns the font name (the font reflected in the font list).

- bool GetFromUnicode () const

  Returns true if the dialog is showing the full range of Unicode characters.

- wxString GetNormalTextFontName () const

  Gets the font name used for displaying symbols in the absence of a selected font.

- wxString GetSymbol () const

  Gets the current or initial symbol as a string.

- int GetSymbolChar () const

Gets the selected symbol character as an integer.

- bool HasSelection () const

  Returns true if a symbol is selected.

- void SetFontName (wxString value)

  Sets the initial/selected font name.

- void SetFromUnicode (bool value)

  Sets the internal flag indicating that the full Unicode range should be displayed.

- void SetNormalTextFontName (wxString value)

  Sets the name of the font to be used in the absence of a selected font.

- void SetSymbol (wxString value)

  Sets the symbol as a one or zero character string.

- void SetUnicodeMode (bool unicodeMode)

  Sets Unicode display mode.

- bool UseNormalFont () const

  Returns true if the has specified normal text - that is, there is no selected font.

Additional Inherited Members

21.743.2 Constructor & Destructor Documentation

wxSymbolPickerDialog::wxSymbolPickerDialog ( )

Default ctor.

wxSymbolPickerDialog::wxSymbolPickerDialog (const wxString & *symbol*, const wxString & *initialFont*, const wxString & *normalTextFont*, wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *title* = _("Symbols"), const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxCLOSE_BOX)

Constructor.

Parameters

symbol	The initial symbol to show. Specify a single character in a string, or an empty string.
initialFont	The initial font to be displayed in the font list. If empty, the item normal text will be selected.
normalTextFont	The font the dialog will use to display the symbols if the initial font is empty.
parent	The dialog's parent.
id	The dialog's identifier.
title	The dialog's caption.
pos	The dialog's position.
size	The dialog's size.
style	The dialog's window style.

21.743.3 Member Function Documentation

bool wxSymbolPickerDialog::Create (const wxString & *symbol*, const wxString & *initialFont*, const wxString & *normalTextFont*, wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxString & *caption* = wxGetTranslation("Symbols"), const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxSize(400, 300), long *style* = wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER|wxCLOSE_BOX)

Creation: see the constructor for details about the parameters.

wxString wxSymbolPickerDialog::GetFontName ( ) const

Returns the font name (the font reflected in the font list).

bool wxSymbolPickerDialog::GetFromUnicode ( ) const

Returns true if the dialog is showing the full range of Unicode characters.

wxString wxSymbolPickerDialog::GetNormalTextFontName ( ) const

Gets the font name used for displaying symbols in the absence of a selected font.

wxString wxSymbolPickerDialog::GetSymbol ( ) const

Gets the current or initial symbol as a string.

int wxSymbolPickerDialog::GetSymbolChar ( ) const

Gets the selected symbol character as an integer.

bool wxSymbolPickerDialog::HasSelection ( ) const

Returns true if a symbol is selected.

void wxSymbolPickerDialog::SetFontName (wxString *value*)

Sets the initial/selected font name.

void wxSymbolPickerDialog::SetFromUnicode (bool *value*)

Sets the internal flag indicating that the full Unicode range should be displayed.

void wxSymbolPickerDialog::SetNormalTextFontName (wxString *value*)

Sets the name of the font to be used in the absence of a selected font.

void wxSymbolPickerDialog::SetSymbol (wxString *value*)

Sets the symbol as a one or zero character string.

void wxSymbolPickerDialog::SetUnicodeMode (bool *unicodeMode*)

Sets Unicode display mode.

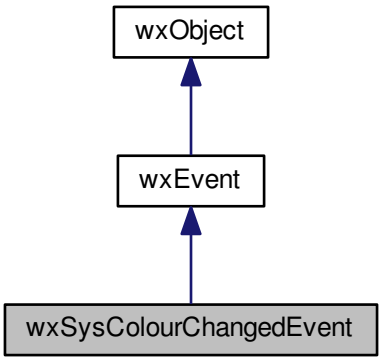
bool wxSymbolPickerDialog::UseNormalFont ( ) const

Returns true if the has specified normal text - that is, there is no selected font.

21.744 wxSysColourChangedEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxSysColourChangedEvent:



21.744.1 Detailed Description

This class is used for system colour change events, which are generated when the user changes the colour settings using the control panel.

This is only appropriate under Windows.

Remarks

The default event handler for this event propagates the event to child windows, since Windows only sends the events to top-level windows. If intercepting this event for a top-level window, remember to call the base class handler, or to pass the event on to the window's children explicitly.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxSysColourChangedEvent& event)

Event macros:

- EVT_SYS_COLOUR_CHANGED(func): Process a `wxEVT_SYS_COLOUR_CHANGED` event.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxSysColourChangedEvent ()

  Constructor.

Additional Inherited Members

21.744.2 Constructor & Destructor Documentation

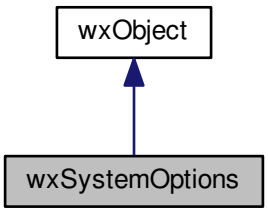
wxSysColourChangedEvent::wxSysColourChangedEvent ( )

Constructor.

21.745 wxSystemOptions Class Reference

`#include <wx/sysopt.h>`

Inheritance diagram for wxSystemOptions:



21.745.1 Detailed Description

wxSystemOptions stores option/value pairs that wxWidgets itself or applications can use to alter behaviour at run-time.

It can be used to optimize behaviour that doesn't deserve a distinct API, but is still important to be able to configure.

System options can be set by the program itself using SetOption() method and they also can be set from the program environment by defining an environment variable `wx_option` to set the given option for all wxWidgets applications or `wx_appname_option` to set it just for the application with the given name (as returned by wxApp::GetAppName()). Notice that any characters not allowed in the environment variables names, such as periods and dashes, should be replaced with underscores. E.g. to define a system option "foo-bar" you need to define the environment variable "wx_foo_bar".

The program may use system options for its own needs but they are mostly used to control the behaviour of wxWidgets library itself.

These options are currently recognised by wxWidgets:

21.745.2 All platforms

- exit-on-assert: If set to non-zero value, abort the program if an assertion fails. The default behaviour in case of assertion failure depends on the build mode and can be changed by overriding wxApp::OnAssertFailure() but setting this option allows to change it without modifying the program code and also applies to asserts which may happen before the wxApp object creation or after its destruction.

21.745.3 Windows

- no-maskblt: 1 to never use WIN32's MaskBlt function, 0 to allow it to be used where possible. Default: 0. In some circumstances the MaskBlt function can be slower than using the fallback code, especially if using DC caching. By default, MaskBlt will be used where it is implemented by the operating system and driver.

- msw.remap: If 1 (the default), wxToolBar bitmap colours will be remapped to the current theme's values. Set this to 0 to disable this functionality, for example if you're using more than 16 colours in your tool bitmaps.

- msw.window.no-clip-children: If 1, windows will not automatically get the WS_CLIPCHILDREN style. This restores the way windows are refreshed back to the method used in versions of wxWidgets earlier than 2.5.4, and for some complex window hierarchies it can reduce apparent refresh delays. You may still specify wxCLIP_CHILDREN for individual windows.

- msw.notebook.themed-background: If set to 0, globally disables themed backgrounds on notebook pages. Note that this won't disable the theme on the actual notebook background (noticeable only if there are no pages).

- msw.staticbox.optimized-paint: If set to 0, switches off optimized wxStaticBox painting. Setting this to 0 causes more flicker, but allows applications to paint graphics on the parent of a static box (the optimized refresh causes any such drawing to disappear).

- msw.display.directdraw: If set to 1, use DirectDraw-based implementation of wxDisplay. By default the standard Win32 functions are used.

- msw.font.no-proof-quality: If set to 1, use default fonts quality instead of proof quality when creating fonts. With proof quality the fonts have slightly better appearance but not all fonts are available in this quality, e.g. the Terminal font in small sizes is not and this option may be used if wider fonts selection is more important than higher quality.

- wince.dialog.real-ok-cancel: The PocketPC guidelines recommend for Ok/Cancel dialogs to use an OK button located inside the caption bar and implement Cancel functionality through Undo outside the dialog. wxDialog::CreateButtonSizer will follow the native behaviour on WinCE but it can be overridden with real wxButtons by setting the option below to 1.

21.745.4 GTK+

- gtk.tlw.can-set-transparent: wxTopLevelWindow::CanSetTransparent() method normally tries to detect automatically whether transparency for top level windows is currently supported, however this may sometimes fail and this option allows to override the automatic detection. Setting it to 1 makes the transparency be always available (setting it can still fail, of course) and setting it to 0 makes it always unavailable.

- gtk.desktop: This option can be set to override the default desktop environment determination. Supported values are GNOME and KDE.

- gtk.window.force-background-colour: If 1, the backgrounds of windows with the wxBG_STYLE_COLOUR background style are cleared forcibly instead of relying on the underlying GTK+ window colour. This works around a display problem when running applications under KDE with the gtk-qt theme installed (0.6 and below).

21.745.5 Mac

- mac.window-plain-transition: If 1, uses a plainer transition when showing a window. You can also use the symbol wxMAC_WINDOW_PLAIN_TRANSITION.
- window-default-variant: The default variant used by windows (cast to integer from the wxWindowVariant enum). Also known as wxWINDOW_DEFAULT_VARIANT.
- mac.listctrl.always_use_generic: Tells wxListCtrl to use the generic control even when it is capable of using the native control instead. Also known as wxMAC_ALWAYS_USE_GENERIC_LISTCTRL.
- mac.textcontrol-use-spell-checker: This option only has effect for Mac OS X 10.4 and higher. If 1 activates the spell checking in wxTextCtrl.
- osx.openfiledialog.always-show-types: Per default a wxFileDialog with wxFD_OPEN does not show a types-popup on OSX but allows the selection of files from any of the supported types. Setting this to 1 shows a wxChoice for selection (if there is more than one supported filetype).

21.745.6 Motif

- motif.largebuttons: If 1, uses a bigger default size for wxButtons.

The compile-time option to include or exclude this functionality is wxUSE_SYSTEM_OPTIONS.

Library: wxBase

Category: Application and System configuration

See also

wxSystemSettings

Public Member Functions

- wxSystemOptions ()

  Default constructor.

Static Public Member Functions

- static wxString GetOption (const wxString &name)

  Gets an option.
- static int GetOptionInt (const wxString &name)

  Gets an option as an integer.
- static bool HasOption (const wxString &name)

  Returns true if the given option is present.
- static bool IsFalse (const wxString &name)

  Returns true if the option with the given name had been set to 0 value.
- static void SetOption (const wxString &name, const wxString &value)

  Sets an option.
- static void SetOption (const wxString &name, int value)

  Sets an option.

Additional Inherited Members

21.745.7 Constructor & Destructor Documentation

wxSystemOptions::wxSystemOptions ( )

Default constructor.

You don't need to create an instance of wxSystemOptions since all of its functions are static.

21.745.8 Member Function Documentation

static wxString wxSystemOptions::GetOption (const wxString & *name*) `[static]`

Gets an option.

The function is case-insensitive to *name*. Returns empty string if the option hasn't been set.

See also

SetOption(), GetOptionInt(), HasOption()

static int wxSystemOptions::GetOptionInt (const wxString & *name*) `[static]`

Gets an option as an integer.

The function is case-insensitive to *name*. If the option hasn't been set, this function returns 0.

See also

SetOption(), GetOption(), HasOption()

static bool wxSystemOptions::HasOption (const wxString & *name*) `[static]`

Returns true if the given option is present.

The function is case-insensitive to *name*.

See also

SetOption(), GetOption(), GetOptionInt()

static bool wxSystemOptions::IsFalse (const wxString & *name*) `[static]`

Returns true if the option with the given *name* had been set to 0 value.

This is mostly useful for boolean options for which you can't use `GetOptionInt(name) == 0` as this would also be true if the option hadn't been set at all.

static void wxSystemOptions::SetOption (const wxString & *name*, const wxString & *value*) `[static]`

Sets an option.

The function is case-insensitive to *name*.

static void wxSystemOptions::SetOption (const wxString & *name,* int *value*) `[static]`

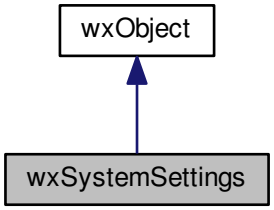
Sets an option.

The function is case-insensitive to *name*.

21.746 wxSystemSettings Class Reference

`#include <wx/settings.h>`

Inheritance diagram for wxSystemSettings:



21.746.1 Detailed Description

wxSystemSettings allows the application to ask for details about the system.

This can include settings such as standard colours, fonts, and user interface element sizes.

Library: wxCore

Category: Application and System configuration

See also

wxFont, wxColour, wxSystemOptions

Public Member Functions

- wxSystemSettings ()
  Default constructor.

Static Public Member Functions

- static wxColour GetColour (wxSystemColour index)
  Returns a system colour.
- static wxFont GetFont (wxSystemFont index)
  Returns a system font.
- static int GetMetric (wxSystemMetric index, wxWindow *win=NULL)

Returns the value of a system metric, or -1 if the metric is not supported on the current system.

- static wxSystemScreenType GetScreenType ()

  Returns the screen type.

- static bool HasFeature (wxSystemFeature index)

  Returns true if the port has certain feature.

Additional Inherited Members

21.746.2 Constructor & Destructor Documentation

wxSystemSettings::wxSystemSettings ( )

Default constructor.

You don't need to create an instance of wxSystemSettings since all of its functions are static.

21.746.3 Member Function Documentation

static wxColour wxSystemSettings::GetColour (wxSystemColour *index*) `[static]`

Returns a system colour.

Parameters

index	Can be one of the wxSystemColour enum values.

Returns

The returned colour is always valid.

static wxFont wxSystemSettings::GetFont (wxSystemFont *index*) `[static]`

Returns a system font.

Parameters

index	Can be one of the wxSystemFont enum values.

Returns

The returned font is always valid.

static int wxSystemSettings::GetMetric (wxSystemMetric *index*, wxWindow * *win* = NULL) `[static]`

Returns the value of a system metric, or -1 if the metric is not supported on the current system.

The value of *win* determines if the metric returned is a global value or a wxWindow based value, in which case it might determine the widget, the display the window is on, or something similar. The window given should be as close to the metric as possible (e.g. a wxTopLevelWindow in case of the wxSYS_CAPTION_Y metric).

index can be one of the wxSystemMetric enum values.

win is a pointer to the window for which the metric is requested. Specifying the *win* parameter is encouraged, because some metrics on some ports are not supported without one,or they might be capable of reporting better values if given one. If a window does not make sense for a metric, one should still be given, as for example it might determine which displays cursor width is requested with wxSYS_CURSOR_X.

static wxSystemScreenType wxSystemSettings::GetScreenType ( ) [static]

Returns the screen type.

The return value is one of the wxSystemScreenType enum values.

static bool wxSystemSettings::HasFeature (wxSystemFeature *index*) [static]

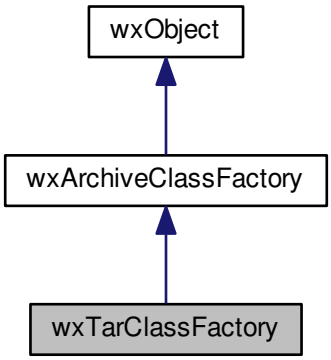
Returns true if the port has certain feature.

See the wxSystemFeature enum values.

21.747 wxTarClassFactory Class Reference

`#include <wx/tarstrm.h>`

Inheritance diagram for wxTarClassFactory:



21.747.1 Detailed Description

Class factory for the tar archive format.

See the base class for details.

Library: wxBase

Category: Archive support, Streams

See also

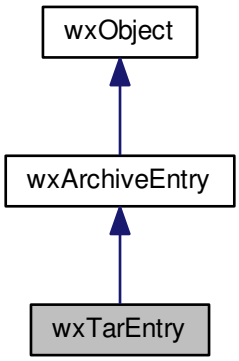
Archive Formats, Generic Archive Programming, wxTarEntry, wxTarInputStream, wxTarOutputStream

Additional Inherited Members

21.748 wxTarEntry Class Reference

`#include <wx/tarstrm.h>`

Inheritance diagram for wxTarEntry:



21.748.1 Detailed Description

Holds the meta-data for an entry in a tar.

21.748.2 Field availability

The tar format stores all the meta-data for an entry ahead of its data, therefore GetNextEntry() always returns a fully populated wxTarEntry object, both when reading from seekable and non-seekable streams.

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, wxTarInputStream, wxTarOutputStream

Public Member Functions

- wxTarEntry (const wxString &name=wxEmptyString, const wxDateTime &dt=wxDateTime::Now(), wxFileOffset size=wxInvalidOffset)

  Constructor.
- wxTarEntry (const wxTarEntry &entry)

  Copy constructor.
- wxString GetInternalName () const

  Returns the entry's filename in the internal format used within the archive.
- wxTarEntry & operator operator= (const wxTarEntry &entry)

Assignment operator.

- wxDateTime GetAccessTime () const

  Gets/sets the entry's access time stamp.
- void SetAccessTime (const wxDateTime &dt)

  Gets/sets the entry's access time stamp.

- wxDateTime GetCreateTime () const

  The entry's creation time stamp.
- void SetCreateTime (const wxDateTime &dt)

  The entry's creation time stamp.

- int GetDevMajor () const

  OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is wxTAR_CHRTYPE or wxTAR_BLKTYPE.
- int GetDevMinor () const

  OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is wxTAR_CHRTYPE or wxTAR_BLKTYPE.
- void SetDevMajor (int dev)

  OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is wxTAR_CHRTYPE or wxTAR_BLKTYPE.
- void SetDevMinor (int dev)

  OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is wxTAR_CHRTYPE or wxTAR_BLKTYPE.

- int GetGroupId () const

  The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.
- int GetUserId () const

  The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.
- void SetGroupId (int id)

  The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.
- void SetUserId (int id)

  The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.

- wxString GetGroupName () const

  The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.
- wxString GetUserName () const

  The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.
- void SetGroupName (const wxString &group)

  The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.
- void SetUserName (const wxString &user)

  The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.

- wxString GetLinkName () const

  The filename of a previous entry in the tar that this entry is a link to.
- void SetLinkName (const wxString &link)

  The filename of a previous entry in the tar that this entry is a link to.

- int GetMode () const

  UNIX permission bits for this entry.
- void SetMode (int mode)

UNIX permission bits for this entry.

- void SetSize (wxFileOffset size)

  The size of the entry's data in bytes.
- wxFileOffset GetSize () const

  The size of the entry's data in bytes.

- int GetTypeFlag () const

  Returns/Sets the type of the entry as a wxTarType value.
- void SetTypeFlag (int type)

  Returns/Sets the type of the entry as a wxTarType value.

Static Public Member Functions

- static wxString GetInternalName (const wxString &name, wxPathFormat format=wxPATH_NATIVE, bool *p↩ IsDir=NULL)

  A static member that translates a filename into the internal format used within the archive.

Additional Inherited Members

21.748.3 Constructor & Destructor Documentation

wxTarEntry::wxTarEntry (const wxString & *name* = wxEmptyString, const wxDateTime & *dt* = wxDateTime::Now ( ), wxFileOffset *size* = wxInvalidOffset)

Constructor.

The tar archive format stores the entry's size ahead of the entry's data. Therefore when creating an archive on a non-seekable stream it is necessary to supply the correct size when each entry is created.

wxTarEntry::wxTarEntry (const wxTarEntry & *entry*)

Copy constructor.

21.748.4 Member Function Documentation

wxDateTime wxTarEntry::GetAccessTime ( ) const

Gets/sets the entry's access time stamp.

See also wxArchiveEntry::GetDateTime() and wxArchiveEntry::SetDateTime().

wxDateTime wxTarEntry::GetCreateTime ( ) const

The entry's creation time stamp.

See also wxArchiveEntry::GetDateTime() and wxArchiveEntry::SetDateTime().

int wxTarEntry::GetDevMajor ( ) const

OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is *wxTAR_CHRTYPE* or *wxTAR_BLKTYPE*.

int wxTarEntry::GetDevMinor ( ) const

OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is *wxTAR_CHRTYPE* or *wxTAR_BLKTYPE*.

int wxTarEntry::GetGroupId ( ) const

The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.

These values aren't usually useful unless the file will only be restored to the same system it originated from. wxTarEntry::GetGroupName() and wxTarEntry::GetUserName() can be used instead.

wxString wxTarEntry::GetGroupName ( ) const

The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.

These are not present in very old tars.

wxString wxTarEntry::GetInternalName ( ) const [virtual]

Returns the entry's filename in the internal format used within the archive.

The name can include directory components, i.e. it can be a full path. The names of directory entries are returned without any trailing path separator. This gives a canonical name that can be used in comparisons.

Implements wxArchiveEntry.

static wxString wxTarEntry::GetInternalName (const wxString & *name*, wxPathFormat *format* = wxPATH_NATIVE, bool * *pIsDir* = NULL) [static]

A static member that translates a filename into the internal format used within the archive.

If the third parameter is provided, the bool pointed to is set to indicate whether the name looks like a directory name (i.e. has a trailing path separator).

wxString wxTarEntry::GetLinkName ( ) const

The filename of a previous entry in the tar that this entry is a link to.

Only meaningful when wxTarEntry::GetTypeFlag() is set to *wxTAR_LNKTYPE* or *wxTAR_SYMTYPE*.

int wxTarEntry::GetMode ( ) const

UNIX permission bits for this entry.

Giving read, write and execute permissions to the file's user and group (see GetGroupName() and GetUserName()) and to others.

The integer is one or more wxPosixPermissions flags OR-combined.

wxFileOffset wxTarEntry::GetSize ( ) const [virtual]

The size of the entry's data in bytes.

The tar archive format stores the entry's size ahead of the entry's data. Therefore when creating an archive on a non-seekable stream it is necessary to supply the correct size when each entry is created.

For seekable streams this is not necessary as wxTarOutputStream will attempt to seek back and fix the entry's header when the entry is closed, though it is still more efficient if the size is given beforehand.

Implements wxArchiveEntry.

int wxTarEntry::GetTypeFlag ( ) const

Returns/Sets the type of the entry as a wxTarType value.

When creating archives use only one of wxTarType values. When reading archives, GetTypeFlag() may return a value which does not match any value of wxTarType; in this case the returned value should be treated as *wxTAR_REGTYPE*.

int wxTarEntry::GetUserId ( ) const

The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.

These values aren't usually useful unless the file will only be restored to the same system it originated from. wxTarEntry::GetGroupName() and wxTarEntry::GetUserName() can be used instead.

wxString wxTarEntry::GetUserName ( ) const

The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.

These are not present in very old tars.

wxTarEntry& wxTarEntry::operator operator= (const wxTarEntry & *entry*)

Assignment operator.

void wxTarEntry::SetAccessTime (const wxDateTime & *dt*)

Gets/sets the entry's access time stamp.

See also wxArchiveEntry::GetDateTime() and wxArchiveEntry::SetDateTime().

void wxTarEntry::SetCreateTime (const wxDateTime & *dt*)

The entry's creation time stamp.

See also wxArchiveEntry::GetDateTime() and wxArchiveEntry::SetDateTime().

void wxTarEntry::SetDevMajor (int *dev*)

OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is *wxTAR_CHRTYPE* or *wxTAR_BLKTYPE*.

void wxTarEntry::SetDevMinor (int *dev*)

OS specific IDs defining a device; these are only meaningful when wxTarEntry::GetTypeFlag() is *wxTAR_CHRTYPE* or *wxTAR_BLKTYPE*.

void wxTarEntry::SetGroupId (int *id*)

The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.

These values aren't usually useful unless the file will only be restored to the same system it originated from. wxTarEntry::GetGroupName() and wxTarEntry::GetUserName() can be used instead.

void wxTarEntry::SetGroupName (const wxString & *group*)

The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.

These are not present in very old tars.

void wxTarEntry::SetLinkName (const wxString & *link*)

The filename of a previous entry in the tar that this entry is a link to.

Only meaningful when wxTarEntry::GetTypeFlag() is set to *wxTAR_LNKTYPE* or *wxTAR_SYMTYPE*.

void wxTarEntry::SetMode (int *mode*)

UNIX permission bits for this entry.

Giving read, write and execute permissions to the file's user and group (see GetGroupName() and GetUserName()) and to others.

The integer is one or more wxPosixPermissions flags OR-combined.

void wxTarEntry::SetSize (wxFileOffset *size*) `[virtual]`

The size of the entry's data in bytes.

The tar archive format stores the entry's size ahead of the entry's data. Therefore when creating an archive on a non-seekable stream it is necessary to supply the correct size when each entry is created.

For seekable streams this is not necessary as wxTarOutputStream will attempt to seek back and fix the entry's header when the entry is closed, though it is still more efficient if the size is given beforehand.

Implements wxArchiveEntry.

void wxTarEntry::SetTypeFlag (int *type*)

Returns/Sets the type of the entry as a wxTarType value.

When creating archives use only one of wxTarType values. When reading archives, GetTypeFlag() may return a value which does not match any value of wxTarType; in this case the returned value should be treated as *wxTAR_REGTYPE*.

void wxTarEntry::SetUserId (int *id*)

The user ID and group ID that has permissions (see wxTarEntry::GetMode()) over this entry.

These values aren't usually useful unless the file will only be restored to the same system it originated from. wxTarEntry::GetGroupName() and wxTarEntry::GetUserName() can be used instead.

void wxTarEntry::SetUserName (const wxString & *user*)

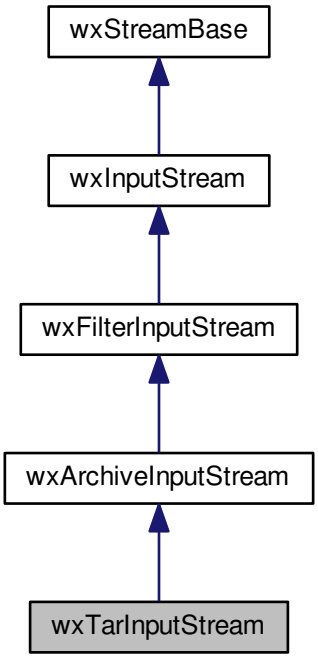
The names of the user and group that has permissions (see wxTarEntry::GetMode()) over this entry.

These are not present in very old tars.

21.749 wxTarInputStream Class Reference

`#include <wx/tarstrm.h>`

Inheritance diagram for wxTarInputStream:



21.749.1 Detailed Description

Input stream for reading tar files.

wxTarInputStream::GetNextEntry() returns a wxTarEntry object containing the meta-data for the next entry in the tar (and gives away ownership). Reading from the wxTarInputStream then returns the entry's data. wxTarInputStream::Eof() becomes true after an attempt has been made to read past the end of the entry's data.

When there are no more entries, wxTarInputStream::GetNextEntry() returns NULL and sets wxTarInputStream::Eof().

Tar entries are seekable if the parent stream is seekable. In practice this usually means they are only seekable if the tar is stored as a local file and is not compressed.

Library: wxBase

Category: Archive support, Streams

See also

Looking Up an Archive Entry by Name

Public Member Functions

- bool CloseEntry ()

  Closes the current entry.
- wxTarEntry * GetNextEntry ()

  Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a wxTarEntry object, giving away ownership.
- bool OpenEntry (wxTarEntry &entry)

  Closes the current entry if one is open, then opens the entry specified by the entry object.

- wxTarInputStream (wxInputStream &stream, wxMBConv &conv=wxConvLocal)

  Constructor.
- wxTarInputStream (wxInputStream *stream, wxMBConv &conv=wxConvLocal)

  Constructor.

Additional Inherited Members

21.749.2 Constructor & Destructor Documentation

wxTarInputStream::wxTarInputStream (wxInputStream & *stream*, wxMBConv & *conv* = `wxConvLocal`)

Constructor.

In a Unicode build the second parameter *conv* is used to translate fields from the standard tar header into Unicode.

It has no effect on the stream's data. *conv* is only used for the standard tar headers, any pax extended headers are always UTF-8 encoded.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

wxTarInputStream::wxTarInputStream (wxInputStream * *stream*, wxMBConv & *conv* = `wxConvLocal`)

Constructor.

In a Unicode build the second parameter *conv* is used to translate fields from the standard tar header into Unicode.

It has no effect on the stream's data. *conv* is only used for the standard tar headers, any pax extended headers are always UTF-8 encoded.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

21.749.3 Member Function Documentation

bool wxTarInputStream::CloseEntry ( ) `[virtual]`

Closes the current entry.

On a non-seekable stream reads to the end of the current entry first.

Implements wxArchiveInputStream.

wxTarEntry* wxTarInputStream::GetNextEntry ( )

Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a wxTarEntry object, giving away ownership.

The stream is then open and can be read.

bool wxTarInputStream::OpenEntry (wxTarEntry & *entry*)

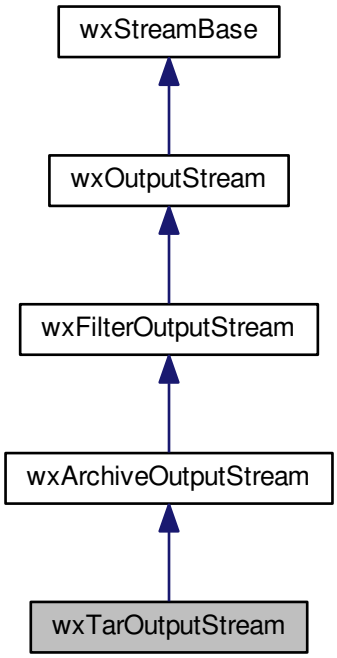
Closes the current entry if one is open, then opens the entry specified by the *entry* object.

entry should be from the same tar file, and the tar should be on a seekable stream.

21.750 wxTarOutputStream Class Reference

`#include <wx/tarstrm.h>`

Inheritance diagram for wxTarOutputStream:



21.750.1 Detailed Description

Output stream for writing tar files.

wxTarOutputStream::PutNextEntry() is used to create a new entry in the output tar, then the entry's data is written to the wxTarOutputStream. Another call to wxTarOutputStream::PutNextEntry() closes the current entry and begins the next.

Library: wxBase

Category: Streams

See also

Archive Formats, wxTarEntry, wxTarInputStream

Public Member Functions

- virtual ~wxTarOutputStream ()

  The destructor calls Close() to finish writing the tar if it has not been called already.
- bool Close ()

  Finishes writing the tar, returning true if successful.
- bool CloseEntry ()

  Close the current entry.
- bool CopyArchiveMetaData (wxTarInputStream &s)

  See wxArchiveOutputStream::CopyArchiveMetaData().
- bool CopyEntry (wxTarEntry *entry, wxTarInputStream &inputStream)

  Takes ownership of entry and uses it to create a new entry in the tar.
- bool PutNextDirEntry (const wxString &name, const wxDateTime &dt=wxDateTime::Now())

  Create a new directory entry (see wxArchiveEntry::IsDir()) with the given name and timestamp.
- bool PutNextEntry (wxTarEntry *entry)

  Takes ownership of entry and uses it to create a new entry in the tar.
- bool PutNextEntry (const wxString &name, const wxDateTime &dt=wxDateTime::Now(), wxFileOffset size=wxInvalidOffset)

  Create a new entry with the given name, timestamp and size.

- wxTarOutputStream (wxOutputStream &stream, wxTarFormat format=wxTAR_PAX, wxMBConv &conv=wxConvLocal)

  If the parent stream is passed as a pointer then the new filter stream takes ownership of it.
- wxTarOutputStream (wxOutputStream *stream, wxTarFormat format=wxTAR_PAX, wxMBConv &conv=wxConvLocal)

  If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

- int GetBlockingFactor () const

  *The tar is zero padded to round its size up to BlockingFactor * 512 bytes.*
- void SetBlockingFactor (int factor)

  *The tar is zero padded to round its size up to BlockingFactor * 512 bytes.*

Additional Inherited Members

21.750.2 Constructor & Destructor Documentation

wxTarOutputStream::wxTarOutputStream (wxOutputStream & *stream*, wxTarFormat *format* = wxTAR_PAX, wxMBConv & *conv* = `wxConvLocal`)

If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

If it is passed by reference then it does not.

In a Unicode build the third parameter *conv* is used to translate the headers fields into an 8-bit encoding. It has no effect on the stream's data.

When the *format* is *wxTAR_PAX*, pax extended headers are generated when any header field will not fit the standard tar header block or if it uses any non-ascii characters.

Extended headers are stored as extra 'files' within the tar, and will be extracted as such by any other tar program that does not understand them. The *conv* parameter only affect the standard tar headers, the extended headers are always UTF-8 encoded.

When the *format* is *wxTAR_USTAR*, no extended headers are generated, and instead a warning message is logged if any header field overflows.

wxTarOutputStream::wxTarOutputStream (wxOutputStream * *stream*, wxTarFormat *format* = wxTAR_PAX, wxMBConv & *conv* = `wxConvLocal`)

If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

If it is passed by reference then it does not.

In a Unicode build the third parameter *conv* is used to translate the headers fields into an 8-bit encoding. It has no effect on the stream's data.

When the *format* is *wxTAR_PAX*, pax extended headers are generated when any header field will not fit the standard tar header block or if it uses any non-ascii characters.

Extended headers are stored as extra 'files' within the tar, and will be extracted as such by any other tar program that does not understand them. The *conv* parameter only affect the standard tar headers, the extended headers are always UTF-8 encoded.

When the *format* is *wxTAR_USTAR*, no extended headers are generated, and instead a warning message is logged if any header field overflows.

virtual wxTarOutputStream::~wxTarOutputStream ( ) `[virtual]`

The destructor calls Close() to finish writing the tar if it has not been called already.

21.750.3 Member Function Documentation

bool wxTarOutputStream::Close ( ) `[virtual]`

Finishes writing the tar, returning true if successful.

Called by the destructor if not called explicitly.

Reimplemented from wxArchiveOutputStream.

bool wxTarOutputStream::CloseEntry ( ) `[virtual]`

Close the current entry.

It is called implicitly whenever another new entry is created with CopyEntry() or PutNextEntry(), or when the tar is closed.

Implements wxArchiveOutputStream.

bool wxTarOutputStream::CopyArchiveMetaData (wxTarInputStream & *s*)

See wxArchiveOutputStream::CopyArchiveMetaData().

For the tar format this function does nothing.

bool wxTarOutputStream::CopyEntry (wxTarEntry * *entry*, wxTarInputStream & *inputStream*)

Takes ownership of *entry* and uses it to create a new entry in the tar.

entry is then opened in *inputStream* and its contents copied to this stream.

For some other archive formats CopyEntry() is much more efficient than transferring the data using Read() and Write() since it will copy them without decompressing and recompressing them. For tar however it makes no difference.

For tars on seekable streams, *entry* must be from the same tar file as *inputStream*. For non-seekable streams, *entry* must also be the last thing read from *inputStream*.

int wxTarOutputStream::GetBlockingFactor ( ) const

The tar is zero padded to round its size up to *BlockingFactor* * 512 bytes.

The blocking factor defaults to 10 for *wxTAR_PAX* and 20 for *wxTAR_USTAR* (see wxTarOutputStream()), as specified in the POSIX standards.

bool wxTarOutputStream::PutNextDirEntry (const wxString & *name*, const wxDateTime & *dt* = wxDateTime::Now ()) [virtual]

Create a new directory entry (see wxArchiveEntry::IsDir()) with the given name and timestamp.

PutNextEntry() can also be used to create directory entries, by supplying a name with a trailing path separator.

Implements wxArchiveOutputStream.

bool wxTarOutputStream::PutNextEntry (wxTarEntry * *entry*)

Takes ownership of entry and uses it to create a new entry in the tar.

bool wxTarOutputStream::PutNextEntry (const wxString & *name*, const wxDateTime & *dt* = wxDateTime::Now (), wxFileOffset *size* = wxInvalidOffset) [virtual]

Create a new entry with the given name, timestamp and size.

Implements wxArchiveOutputStream.

void wxTarOutputStream::SetBlockingFactor (int *factor*)

The tar is zero padded to round its size up to *BlockingFactor* * 512 bytes.

The blocking factor defaults to 10 for *wxTAR_PAX* and 20 for *wxTAR_USTAR* (see wxTarOutputStream()), as specified in the POSIX standards.

21.751 wxTaskBarButton Class Reference

`#include <wx/taskbarbutton.h>`

21.751.1 Detailed Description

A taskbar button that associated with the window under Windows 7 or later.

It is used to access the functionality including thumbnail representations, thumbnail toolbars, notification and status overlays, and progress indicators.

Note

This class is only created and initialized in the internal implementation of wxFrame by design. You can only get the pointer of the instance which associated with the frame by calling wxFrame::MSWGetTaskBarButton().

Library: wxCore

Category: Miscellaneous

Implementations: native under wxMSW port; a generic implementation is used elsewhere. Availability: only available for the wxMSW port.

See also

wxFrame::MSWGetTaskBarButton()

Since

3.1.0

Public Member Functions

- virtual void SetProgressRange (int range)

  Starts showing a determinate progress indicator.
- virtual void SetProgressValue (int value)

  Update the progress indicator, setting the progress to the new value .
- virtual void PulseProgress ()

  Makes the progress indicator run in indeterminate mode.
- virtual void Show (bool show=true)

  Show in the taskbar.
- virtual void Hide ()

  Hide in the taskbar.
- virtual void SetThumbnailTooltip (const wxString &tooltip)

  Specifies or updates the text of the tooltip that is displayed when the mouse pointer rests on an individual preview thumbnail in a taskbar button flyout.
- virtual void SetProgressState (wxTaskBarButtonState state)

  Set the state of the progress indicator displayed on a taskbar button.
- virtual void SetOverlayIcon (const wxIcon &icon, const wxString &description=wxString())

  Set an overlay icon to indicate application status or a notification top the user.
- virtual void SetThumbnailClip (const wxRect &rect)

  Selects a portion of a window's client area to display as that window's thumbnail in the taskbar.
- virtual void SetThumbnailContents (const wxWindow *child)

  Selects the child window area to display as that window's thumbnail in the taskbar.
- virtual bool InsertThumbBarButton (size_t pos, wxThumbBarButton *button)

  Inserts the given button before the position pos to the taskbar thumbnail toolbar.
- virtual bool AppendThumbBarButton (wxThumbBarButton *button)

  Appends a button to the taskbar thumbnail toolbar.
- virtual bool AppendSeparatorInThumbBar ()

  Appends a separator to the taskbar thumbnail toolbar.
- virtual wxThumbBarButton * RemoveThumbBarButton (wxThumbBarButton *button)

  Removes the thumbnail toolbar button from the taskbar button but doesn't delete the associated c++ object.
- virtual wxThumbBarButton * RemoveThumbBarButton (int id)

  Removes the thumbnail toolbar button from the taskbar button but doesn't delete the associated c++ object.

21.751.2 Member Function Documentation

virtual bool wxTaskBarButton::AppendSeparatorInThumbBar ( ) [virtual]

Appends a separator to the taskbar thumbnail toolbar.

Note

The number of buttons and separators is limited to 7.

See also

AppendThumbBarButton(), InsertThumbBarButton()

virtual bool wxTaskBarButton::AppendThumbBarButton (wxThumbBarButton * *button*) [virtual]

Appends a button to the taskbar thumbnail toolbar.

Note

The number of buttons and separators is limited to 7.

See also

InsertThumbBarButton(), AppendSeparatorInThumbBar()

virtual void wxTaskBarButton::Hide ( ) [virtual]

Hide in the taskbar.

virtual bool wxTaskBarButton::InsertThumbBarButton (size_t *pos,* wxThumbBarButton * *button*) [virtual]

Inserts the given button before the position pos to the taskbar thumbnail toolbar.

Note

The number of buttons and separators is limited to 7.

See also

AppendThumbBarButton(), AppendSeparatorInThumbBar()

virtual wxThumbBarButton* wxTaskBarButton::RemoveThumbBarButton (wxThumbBarButton * *button*) [virtual]

Removes the thumbnail toolbar button from the taskbar button but doesn't delete the associated c++ object.

Parameters

button	The thumbnail toolbar button to remove.

Returns

A pointer to the button which was detached from the taskbar button.

virtual wxThumbBarButton* wxTaskBarButton::RemoveThumbBarButton (int *id*) [virtual]

Removes the thumbnail toolbar button from the taskbar button but doesn't delete the associated c++ object.

Parameters

id	The identifier of the thumbnail toolbar button to remove.

Returns

A pointer to the button which was detached from the taskbar button.

virtual void wxTaskBarButton::SetOverlayIcon (const wxIcon & *icon*, const wxString & *description* = wxString()) [virtual]

Set an overlay icon to indicate application status or a notification top the user.

Parameters

icon	This should be a small icon, measuring 16x16 pixels at 96 dpi. If an overlay icon is already applied to the taskbar button, that existing overlay is replaced. Setting with wxNullIcon to remove.
description	The property holds the description of the overlay for accessibility purposes.

virtual void wxTaskBarButton::SetProgressRange (int *range*) [virtual]

Starts showing a determinate progress indicator.

Call SetProgressValue() after this call to update the progress indicator.

If *range* is 0, the progress indicator is dismissed.

virtual void wxTaskBarButton::SetProgressState (wxTaskBarButtonState *state*) [virtual]

Set the state of the progress indicator displayed on a taskbar button.

See also

wxTaskBarButtonState

virtual void wxTaskBarButton::SetProgressValue (int *value*) [virtual]

Update the progress indicator, setting the progress to the new value .

Parameters

value	Must be in the range from 0 to the argument to the last SetProgressRange() call. When it is equal to the range, the progress bar is dismissed.

virtual void wxTaskBarButton::SetThumbnailClip (const wxRect & *rect*) [virtual]

Selects a portion of a window's client area to display as that window's thumbnail in the taskbar.

Parameters

rect	The portion inside of the window. Setting with an empty wxRect will restore the default diaplay of the thumbnail.

virtual void wxTaskBarButton::SetThumbnailContents (const wxWindow * *child*) [virtual]

Selects the child window area to display as that window's thumbnail in the taskbar.

virtual void wxTaskBarButton::SetThumbnailTooltip (const wxString & *tooltip*) [virtual]

Specifies or updates the text of the tooltip that is displayed when the mouse pointer rests on an individual preview thumbnail in a taskbar button flyout.

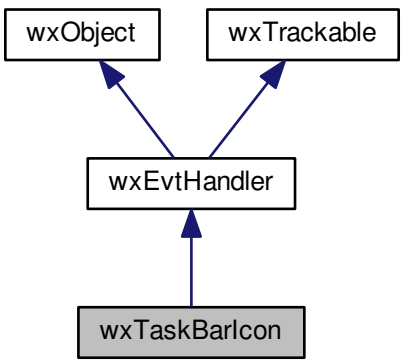
virtual void wxTaskBarButton::Show (bool *show* = true) [virtual]

Show in the taskbar.

21.752 wxTaskBarIcon Class Reference

`#include <wx/taskbar.h>`

Inheritance diagram for wxTaskBarIcon:



21.752.1 Detailed Description

This class represents a taskbar icon.

A taskbar icon is an icon that appears in the 'system tray' and responds to mouse clicks, optionally with a tooltip above it to help provide information.

21.752.2 X Window System Note

Under X Window System, the window manager must support either the "System Tray Protocol" (see http://freedesktop.org/wiki/Specifications/systemtray-spec) by freedesktop.org (WMs used by modern desktop environments such as GNOME >= 2, KDE >= 3 and XFCE >= 4 all do) or the older methods used in GNOME 1.2 and KDE 1 and 2.

If it doesn't, the icon will appear as a toplevel window on user's desktop. Because not all window managers have system tray, there's no guarantee that wxTaskBarIcon will work correctly under X Window System and so the applications should use it only as an optional component of their user interface. The user should be required to explicitly enable the taskbar icon on Unix, it shouldn't be on by default.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxTaskBarIconEvent& event)

Event macros for events emitted by this class: Note that not all ports are required to send these events and so it's better to override wxTaskBarIcon::CreatePopupMenu() if all that the application does is that it shows a popup menu in reaction to mouse click.

- EVT_TASKBAR_MOVE(func): Process a `wxEVT_TASKBAR_MOVE` event.
- EVT_TASKBAR_LEFT_DOWN(func): Process a `wxEVT_TASKBAR_LEFT_DOWN` event.
- EVT_TASKBAR_LEFT_UP(func): Process a `wxEVT_TASKBAR_LEFT_UP` event.
- EVT_TASKBAR_RIGHT_DOWN(func): Process a `wxEVT_TASKBAR_RIGHT_DOWN` event.
- EVT_TASKBAR_RIGHT_UP(func): Process a `wxEVT_TASKBAR_RIGHT_UP` event.
- EVT_TASKBAR_LEFT_DCLICK(func): Process a `wxEVT_TASKBAR_LEFT_DCLICK` event.
- EVT_TASKBAR_RIGHT_DCLICK(func): Process a `wxEVT_TASKBAR_RIGHT_DCLICK` event.
- EVT_TASKBAR_CLICK(func): This is a synonym for either EVT_TASKBAR_RIGHT_DOWN or UP depending on the platform, use this event macro to catch the event which should result in the menu being displayed on the current platform.

Library: wxAdvanced

Category: Miscellaneous

Public Member Functions

- wxTaskBarIcon (wxTaskBarIconType iconType=wxTBI_DEFAULT_TYPE)

  Default constructor.
- virtual ~wxTaskBarIcon ()

  Destroys the wxTaskBarIcon object, removing the icon if not already removed.
- void Destroy ()

This method is similar to wxWindow::Destroy and can be used to schedule the task bar icon object for the delayed destruction: it will be deleted during the next event loop iteration, which allows the task bar icon to process any pending events for it before being destroyed.

- bool IsIconInstalled () const

  Returns true if SetIcon() was called with no subsequent RemoveIcon().

- bool IsOk () const

  Returns true if the object initialized successfully.

- virtual bool PopupMenu (wxMenu *menu)

  Pops up a menu at the current mouse position.

- virtual bool RemoveIcon ()

  Removes the icon previously set with SetIcon().

- virtual bool SetIcon (const wxIcon &icon, const wxString &tooltip=wxEmptyString)

  Sets the icon, and optional tooltip text.

Static Public Member Functions

- static bool IsAvailable ()

  Returns true if system tray is available in the desktop environment the app runs under.

Protected Member Functions

- virtual wxMenu * CreatePopupMenu ()

  This method is called by the library when the user requests popup menu (on Windows and Unix platforms, this is when the user right-clicks the icon).

Additional Inherited Members

21.752.3 Constructor & Destructor Documentation

wxTaskBarIcon::wxTaskBarIcon (wxTaskBarIconType *iconType* = wxTBI_DEFAULT_TYPE)

Default constructor.

The iconType is only applicable on wxOSX_Cocoa.

virtual wxTaskBarIcon::~wxTaskBarIcon ( ) [virtual]

Destroys the wxTaskBarIcon object, removing the icon if not already removed.

21.752.4 Member Function Documentation

virtual wxMenu* wxTaskBarIcon::CreatePopupMenu ( ) [protected], [virtual]

This method is called by the library when the user requests popup menu (on Windows and Unix platforms, this is when the user right-clicks the icon).

Override this function in order to provide popup menu associated with the icon. If CreatePopupMenu() returns NULL (this happens by default), no menu is shown, otherwise the menu is displayed and then deleted by the library as soon as the user dismisses it.

The events can be handled by a class derived from wxTaskBarIcon.

void wxTaskBarIcon::Destroy ( )

This method is similar to wxWindow::Destroy and can be used to schedule the task bar icon object for the delayed destruction: it will be deleted during the next event loop iteration, which allows the task bar icon to process any pending events for it before being destroyed.

static bool wxTaskBarIcon::IsAvailable ( ) [static]

Returns true if system tray is available in the desktop environment the app runs under.

On Windows and Mac OS X, the tray is always available and this function simply returns true.

On Unix, X11 environment may or may not provide the tray, depending on user's desktop environment. Most modern desktops support the tray via the System Tray Protocol by freedesktop.org (http://freedesktop.org/wiki/Specifications/systemtray-spec).

Note

Tray availability may change during application's lifetime under X11 and so applications shouldn't cache the result.
wxTaskBarIcon supports older GNOME 1.2 and KDE 1/2 methods of adding icons to tray, but they are unreliable and this method doesn't detect them.

Since

2.9.0

bool wxTaskBarIcon::IsIconInstalled ( ) const

Returns true if SetIcon() was called with no subsequent RemoveIcon().

bool wxTaskBarIcon::IsOk ( ) const

Returns true if the object initialized successfully.

virtual bool wxTaskBarIcon::PopupMenu (wxMenu * *menu*) [virtual]

Pops up a menu at the current mouse position.

The events can be handled by a class derived from wxTaskBarIcon.

Note

It is recommended to override CreatePopupMenu() callback instead of calling this method from event handler, because some ports (e.g. wxCocoa) may not implement PopupMenu() and mouse click events at all.

virtual bool wxTaskBarIcon::RemoveIcon ( ) [virtual]

Removes the icon previously set with SetIcon().

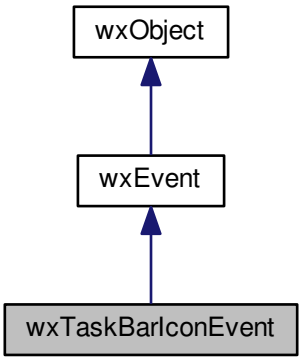
virtual bool wxTaskBarIcon::SetIcon (const wxIcon & *icon*, const wxString & *tooltip* = wxEmptyString) [virtual]

Sets the icon, and optional tooltip text.

21.753 wxTaskBarIconEvent Class Reference

`#include <wx/taskbar.h>`

Inheritance diagram for wxTaskBarIconEvent:



21.753.1 Detailed Description

The event class used by wxTaskBarIcon.

For a list of the event macros meant to be used with wxTaskBarIconEvent, please look at wxTaskBarIcon description.

Library: wxAdvanced

Category: Events

Public Member Functions

- wxTaskBarIconEvent (wxEventType evtType, wxTaskBarIcon *tbIcon)

  Constructor.

Additional Inherited Members

21.753.2 Constructor & Destructor Documentation

wxTaskBarIconEvent::wxTaskBarIconEvent (wxEventType *evtType,* wxTaskBarIcon * *tbIcon*)

Constructor.

21.754 wxTaskBarJumpList Class Reference

`#include <wx/taskbarbutton.h>`

21.754.1 Detailed Description

This class is an transparent wrapper around Windows Jump Lists.

Jump Lists, as an new feature since Windows 7, are lists of recently opened items, such as files, folders, or websites, which are organized by the program that the user use to open them. Jump Lists don't just show shortcuts to files. Sometimes they can also provide quick access to tasks. With this class, you can access the recent and frequent category in the jump lists. You can also set the tasks category of the Jump Lists of your application. What's more, you can add custom category to them.

Library: wxCore

Category: Miscellaneous

Availability: only available for the wxMSW port.

See also

wxTaskBarJumpListCategory, wxTaskBarJumpListItem

Since

3.1.0

Public Member Functions

- wxTaskBarJumpList (const wxString &appID=wxEmptyString)

  Constructs the jump list.
- virtual ~wxTaskBarJumpList ()
- void ShowRecentCategory (bool shown=true)

  Shows or hides the recent category.
- void HideRecentCategory ()

  Hides the recent category.
- void ShowFrequentCategory (bool shown=true)

  Shows or hides the frequent category.
- void HideFrequentCategory ()

  Hides the frequent category.
- wxTaskBarJumpListCategory & GetTasks () const

  Accesses the built in tasks category.
- const wxTaskBarJumpListCategory & GetFrequentCategory () const

  Gets the built in frequent category.
- const wxTaskBarJumpListCategory & GetRecentCategory () const

  Gets the built in recent category.
- const wxTaskBarJumpListCategories & GetCustomCategories () const

  Gets the custom categories.
- void AddCustomCategory (wxTaskBarJumpListCategory *category)

  Add an new custom category.
- wxTaskBarJumpListCategory * RemoveCustomCategory (const wxString &title)

  Removes the custom category from the jump lists but doesn't delete the associated C++ object.
- void DeleteCustomCategory (const wxString &title)

  Deletes the custom category from the jump lists.

21.754.2 Constructor & Destructor Documentation

wxTaskBarJumpList::wxTaskBarJumpList (const wxString & *appID* = wxEmptyString)

Constructs the jump list.

Parameters

appID	Specifies a unique identifier for the application jump list, can be empty by default.

See Application User Model IDs on MSDN for further details.

virtual wxTaskBarJumpList::~wxTaskBarJumpList ( ) [virtual]

21.754.3 Member Function Documentation

void wxTaskBarJumpList::AddCustomCategory (wxTaskBarJumpListCategory * *category*)

Add an new custom category.

Parameters

category	A wxTaskBarJumpListCategory object. It will be owned by the wxTaskBarJumpList object after this function is called, so do not delete it yourself.

void wxTaskBarJumpList::DeleteCustomCategory (const wxString & *title*)

Deletes the custom category from the jump lists.

Parameters

title	The title of the custom category.

const wxTaskBarJumpListCategories& wxTaskBarJumpList::GetCustomCategories ( ) const

Gets the custom categories.

const wxTaskBarJumpListCategory& wxTaskBarJumpList::GetFrequentCategory ( ) const

Gets the built in frequent category.

Note that the returned category is read-only.

const wxTaskBarJumpListCategory& wxTaskBarJumpList::GetRecentCategory ( ) const

Gets the built in recent category.

Note that the returned category is read-only.

wxTaskBarJumpListCategory& wxTaskBarJumpList::GetTasks ( ) const

Accesses the built in tasks category.

With the returned tasks category, you can append an new task, remove an existing task, modify the task item etc.

void wxTaskBarJumpList::HideFrequentCategory ( )

Hides the frequent category.

Equivalent to calling wxTaskBarJumpList::ShowFrequentCategory(false).

void wxTaskBarJumpList::HideRecentCategory ( )

Hides the recent category.

Equivalent to calling wxTaskBarJumpList::ShowFrequentCategory(false).

wxTaskBarJumpListCategory* wxTaskBarJumpList::RemoveCustomCategory (const wxString & *title*)

Removes the custom category from the jump lists but doesn't delete the associated C++ object.

Parameters

title	The title of the custom category.

void wxTaskBarJumpList::ShowFrequentCategory (bool *shown* = `true`)

Shows or hides the frequent category.

void wxTaskBarJumpList::ShowRecentCategory (bool *shown* = `true`)

Shows or hides the recent category.

21.755 wxTaskBarJumpListCategory Class Reference

`#include <wx/taskbarbutton.h>`

21.755.1 Detailed Description

This class represents a category of jump list in the taskbar button.

There are four kinds of categories in Windows: Recent, Frequent, Tasks and custom.

Library: wxCore

Category: Miscellaneous

Availability: only available for the wxMSW port.

See also

wxTaskBarJumpList, wxTaskBarJumpListItem

Since

3.1.0

Public Member Functions

- wxTaskBarJumpListCategory (wxTaskBarJumpList *parent=NULL, const wxString &title=wxEmptyString)

  Constructs the jump list category.
- virtual ~wxTaskBarJumpListCategory ()
- wxTaskBarJumpListItem * Append (wxTaskBarJumpListItem *item)

Appends a jump list item.

- void Delete (wxTaskBarJumpListItem *item)

  Deletes the jump list item from the category.
- wxTaskBarJumpListItem * Remove (wxTaskBarJumpListItem *item)

  Removes the jump list item from the category but doesn't delete the associated C++ object.
- wxTaskBarJumpListItem * FindItemByPosition (size_t pos) const

  Returns the wxTaskBarJumpListItem given a position in the category.
- wxTaskBarJumpListItem * Insert (size_t pos, wxTaskBarJumpListItem *item)

  Inserts the given item before the position pos.
- wxTaskBarJumpListItem * Prepend (wxTaskBarJumpListItem *item)

  Inserts the given item at position 0, i.e.
- void SetTitle (const wxString &title)

  Sets the title of the category.
- const wxString & GetTitle () const

  Gets the title of the category.
- const wxTaskBarJumpListItems & GetItems () const

  Gets the jump list items of the category.

21.755.2 Constructor & Destructor Documentation

wxTaskBarJumpListCategory::wxTaskBarJumpListCategory (wxTaskBarJumpList * *parent* = NULL, const wxString & *title* = wxEmptyString)

Constructs the jump list category.

Parameters

parent	Jump list that the jump list category belongs to. Can be NULL if the category is going to be added to the jump list later.
title	The title of the category.

virtual wxTaskBarJumpListCategory::~wxTaskBarJumpListCategory ( ) [virtual]

21.755.3 Member Function Documentation

wxTaskBarJumpListItem* wxTaskBarJumpListCategory::Append (wxTaskBarJumpListItem * *item*)

Appends a jump list item.

Parameters

item	The jump list item to be appended. It will be owned by the wxTaskBarJumpListCategory object after this function is called, so do not delete it yourself.

See also

Insert(), Prepend()

void wxTaskBarJumpListCategory::Delete (wxTaskBarJumpListItem * *item*)

Deletes the jump list item from the category.

Parameters

item	The jump list item to be deleted.

See also

Remove()

wxTaskBarJumpListItem* wxTaskBarJumpListCategory::FindItemByPosition (size_t *pos*) const

Returns the wxTaskBarJumpListItem given a position in the category.

const wxTaskBarJumpListItems& wxTaskBarJumpListCategory::GetItems ( ) const

Gets the jump list items of the category.

const wxString& wxTaskBarJumpListCategory::GetTitle ( ) const

Gets the title of the category.

wxTaskBarJumpListItem* wxTaskBarJumpListCategory::Insert (size_t *pos,* wxTaskBarJumpListItem * *item*)

Inserts the given item before the position pos.

See also

Append(), Prepend()

wxTaskBarJumpListItem* wxTaskBarJumpListCategory::Prepend (wxTaskBarJumpListItem * *item*)

Inserts the given item at position 0, i.e.

before all the other existing items.

See also

Append(), Insert();

wxTaskBarJumpListItem* wxTaskBarJumpListCategory::Remove (wxTaskBarJumpListItem * *item*)

Removes the jump list item from the category but doesn't delete the associated C++ object.

Parameters

item	The jump list item to be removed.

void wxTaskBarJumpListCategory::SetTitle (const wxString & *title*)

Sets the title of the category.

21.756 wxTaskBarJumpListItem Class Reference

`#include <wx/taskbarbutton.h>`

21.756.1 Detailed Description

A wxTaskBarJumpListItem represents an item in a jump list category.

Library: wxCore

Category: Miscellaneous

Availability: only available for the wxMSW port.

Since

3.1.0

Public Member Functions

- wxTaskBarJumpListItem (wxTaskBarJumpListCategory *parentCategory=NULL, wxTaskBarJumpListItemType type=wxTASKBAR_JUMP_LIST_SEPARATOR, const wxString &title=wxEmptyString, const wxString &filePath=wxEmptyString, const wxString &arguments=wxEmptyString, const wxString &tooltip=wxEmptyString, const wxString &iconPath=wxEmptyString, int iconIndex=0)

  Constructs a jump list item.
- wxTaskBarJumpListItemType GetType () const

  Returns the type of this item.
- void SetType (wxTaskBarJumpListItemType type)

  Sets the type of this item.
- const wxString & GetTitle () const

  Returns the title of this item.
- void SetTitle (const wxString &title)

  Sets the title of this item.
- const wxString & GetFilePath () const

  Returns the file path of this item.
- void SetFilePath (const wxString &filePath)

  Sets the file path of this item.
- const wxString & GetArguments () const

  Returns the command-line arguments of this item.
- void SetArguments (const wxString &arguments)

  Sets the command-line arguments of this item.
- const wxString & GetTooltip () const

  Returns the description tooltip of this item.
- void SetTooltip (const wxString &tooltip)

  Sets the description tooltip of this item.
- const wxString & GetIconPath () const

  Returns the icon path of this item.
- void SetIconPath (const wxString &iconPath)

  Sets the icon path of this item.
- int GetIconIndex () const

  Returns the icon index of icon in this item.

- void SetIconIndex (int iconIndex)

  Sets the icon index of icon in this item.
- wxTaskBarJumpListCategory * GetCategory () const

  Returns the category this jump list item is in, or NULL if this jump list item is not attached.
- void SetCategory (wxTaskBarJumpListCategory *category)

  Sets the parent category which will contain this jump list item.

21.756.2 Constructor & Destructor Documentation

wxTaskBarJumpListItem::wxTaskBarJumpListItem (wxTaskBarJumpListCategory * *parentCategory* = NULL, wxTaskBarJumpListItemType *type* = wxTASKBAR_JUMP_LIST_SEPARATOR, const wxString & *title* = wxEmptyString, const wxString & *filePath* = wxEmptyString, const wxString & *arguments* = wxEmptyString, const wxString & *tooltip* = wxEmptyString, const wxString & *iconPath* = wxEmptyString, int *iconIndex* = 0)

Constructs a jump list item.

Parameters

parentCategory	Category that the jump list item belongs to. Can be NULL if the item is going to be added to the category later.
type	The type for this item.
title	The title of this item.
filePath	The filePath of this item, the meaning of which depends on the type of this item: If the item type is wxTASKBAR_JUMP_LIST_DESTIONATION, filePath is the path to a file that can be opened by an application. If the item type is wxTASKBAR_JUMP_LIST_TASK, filePath is the path to an executable that is executed when this item is clicked by the user.
arguments	The command-line arguments of this item.
tooltip	The description tooltip of this item.
iconPath	The path to the file containing the icon.
iconIndex	The index of the icon, which is specified by iconPath.

21.756.3 Member Function Documentation

const wxString& wxTaskBarJumpListItem::GetArguments ( ) const

Returns the command-line arguments of this item.

wxTaskBarJumpListCategory* wxTaskBarJumpListItem::GetCategory ( ) const

Returns the category this jump list item is in, or NULL if this jump list item is not attached.

const wxString& wxTaskBarJumpListItem::GetFilePath ( ) const

Returns the file path of this item.

int wxTaskBarJumpListItem::GetIconIndex ( ) const

Returns the icon index of icon in this item.

const wxString& wxTaskBarJumpListItem::GetIconPath ( ) const

Returns the icon path of this item.

const wxString& wxTaskBarJumpListItem::GetTitle ( ) const

Returns the title of this item.

const wxString& wxTaskBarJumpListItem::GetTooltip ( ) const

Returns the description tooltip of this item.

wxTaskBarJumpListItemType wxTaskBarJumpListItem::GetType ( ) const

Returns the type of this item.

void wxTaskBarJumpListItem::SetArguments (const wxString & *arguments*)

Sets the command-line arguments of this item.

void wxTaskBarJumpListItem::SetCategory (wxTaskBarJumpListCategory * *category*)

Sets the parent category which will contain this jump list item.

void wxTaskBarJumpListItem::SetFilePath (const wxString & *filePath*)

Sets the file path of this item.

void wxTaskBarJumpListItem::SetIconIndex (int *iconIndex*)

Sets the icon index of icon in this item.

void wxTaskBarJumpListItem::SetIconPath (const wxString & *iconPath*)

Sets the icon path of this item.

void wxTaskBarJumpListItem::SetTitle (const wxString & *title*)

Sets the title of this item.

void wxTaskBarJumpListItem::SetTooltip (const wxString & *tooltip*)

Sets the description tooltip of this item.

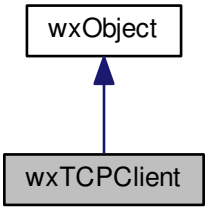
void wxTaskBarJumpListItem::SetType (wxTaskBarJumpListItemType *type*)

Sets the type of this item.

21.757 wxTCPClient Class Reference

```
#include <wx/sckipc.h>
```

Inheritance diagram for wxTCPClient:



21.757.1 Detailed Description

A wxTCPClient object represents the client part of a client-server conversation.

It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using wxDDEClient.

To create a client which can communicate with a suitable server, you need to derive a class from wxTCPConnection and another from wxTCPClient. The custom wxTCPConnection class will intercept communications in a 'conversation' with a server, and the custom wxTCPServer is required so that a user-overridden wxTCPClient::OnMake↩Connection() member can return a wxTCPConnection of the required class, when a connection is made.

Library: wxNet

Category: Networking

See also

wxTCPServer, wxTCPConnection, Interprocess Communication

Public Member Functions

- wxTCPClient ()

  Constructs a client object.

- virtual wxConnectionBase * MakeConnection (const wxString &host, const wxString &service, const wxString &topic)

  Tries to make a connection with a server specified by the host (a machine name under Unix), service name (must contain an integer port number under Unix), and a topic string.

- virtual wxConnectionBase * OnMakeConnection ()

  *The type of wxTCPConnection returned from a MakeConnection() call can be altered by deriving the **OnMake↩Connection** member to return your own derived connection object.*

- virtual bool ValidHost (const wxString &host)

  Returns true if this is a valid host name, false otherwise.

Additional Inherited Members

21.757.2 Constructor & Destructor Documentation

wxTCPClient::wxTCPClient ( )

Constructs a client object.

21.757.3 Member Function Documentation

virtual wxConnectionBase* wxTCPClient::MakeConnection (const wxString & *host*, const wxString & *service*, const wxString & *topic*) `[virtual]`

Tries to make a connection with a server specified by the host (a machine name under Unix), service name (must contain an integer port number under Unix), and a topic string.

If the server allows a connection, a wxTCPConnection object will be returned.

The type of wxTCPConnection returned can be altered by overriding the OnMakeConnection() member to return your own derived connection object.

virtual wxConnectionBase* wxTCPClient::OnMakeConnection ( ) `[virtual]`

The type of wxTCPConnection returned from a MakeConnection() call can be altered by deriving the **OnMake↩ Connection** member to return your own derived connection object.

By default, a wxTCPConnection object is returned.

The advantage of deriving your own connection class is that it will enable you to intercept messages initiated by the server, such as wxTCPConnection::OnAdvise(). You may also want to store application-specific data in instances of the new class.

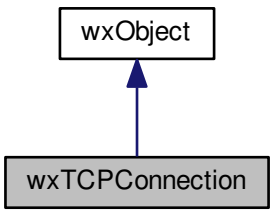
virtual bool wxTCPClient::ValidHost (const wxString & *host*) `[virtual]`

Returns true if this is a valid host name, false otherwise.

21.758 wxTCPConnection Class Reference

`#include <wx/sckipc.h>`

Inheritance diagram for wxTCPConnection:



21.758.1 Detailed Description

A wxTCPClient object represents the connection between a client and a server.

It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using wxDDEConnection.

A wxTCPConnection object can be created by making a connection using a wxTCPClient object, or by the acceptance of a connection by a wxTCPServer object. The bulk of a conversation is controlled by calling members in a **wxTCPConnection** object or by overriding its members.

An application should normally derive a new connection class from wxTCPConnection, in order to override the communication event handlers to do something interesting.

Library: wxNet

Category: Networking

See also

wxTCPClient, wxTCPServer, Interprocess Communication

Public Member Functions

- virtual bool Disconnect ()

  Called by the client or server application to disconnect from the other program; it causes the OnDisconnect() message to be sent to the corresponding connection object in the other program.
- virtual bool OnAdvise (const wxString &topic, const wxString &item, const void ∗data, size_t size, wxIPC↩ Format format)

  Message sent to the client application when the server notifies it of a change in the data associated with the given item.
- virtual bool OnDisconnect ()

  Message sent to the client or server application when the other application notifies it to delete the connection.
- virtual bool OnExecute (const wxString &topic, const void ∗data, size_t size, wxIPCFormat format)

  Message sent to the server application when the client notifies it to execute the given data.
- virtual bool OnPoke (const wxString &topic, const wxString &item, const void ∗data, size_t size, wxIPCFormat format)

  Message sent to the server application when the client notifies it to accept the given data.
- virtual const void ∗ OnRequest (const wxString &topic, const wxString &item, size_t ∗size, wxIPCFormat format)

  Message sent to the server application when the client calls Request().
- virtual bool OnStartAdvise (const wxString &topic, const wxString &item)

  Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item.
- virtual bool OnStopAdvise (const wxString &topic, const wxString &item)

  Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item.
- virtual const void ∗ Request (const wxString &item, size_t ∗size=0, wxIPCFormat format=wxIPC_TEXT)

  Called by the client application to request data from the server.
- virtual bool StartAdvise (const wxString &item)

  Called by the client application to ask if an advise loop can be started with the server.
- virtual bool StopAdvise (const wxString &item)

  Called by the client application to ask if an advise loop can be stopped.

- wxTCPConnection ()

  Constructs a connection object.
- wxTCPConnection (void ∗buffer, size_t size)

  Constructs a connection object.

- bool Advise (const wxString &item, const void ∗data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const char ∗data)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const wchar_t ∗data)

  Called by the server application to advise the client of a change in the data associated with the given item.
- bool Advise (const wxString &item, const wxString data)

  Called by the server application to advise the client of a change in the data associated with the given item.

- bool Execute (const void ∗data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the client application to execute a command on the server.
- bool Execute (const char ∗data)

  Called by the client application to execute a command on the server.
- bool Execute (const wchar_t ∗data)

  Called by the client application to execute a command on the server.
- bool Execute (const wxString data)

  Called by the client application to execute a command on the server.

- bool Poke (const wxString &item, const void ∗data, size_t size, wxIPCFormat format=wxIPC_PRIVATE)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const char ∗data)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const wchar_t ∗data)

  Called by the client application to poke data into the server.
- bool Poke (const wxString &item, const wxString data)

  Called by the client application to poke data into the server.

Additional Inherited Members

21.758.2 Constructor & Destructor Documentation

wxTCPConnection::wxTCPConnection ( )

Constructs a connection object.

If no user-defined connection object is to be derived from wxTCPConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection.

However, if the user defines his or her own derived connection object, the wxTCPServer::OnAcceptConnection and/or wxTCPClient::OnMakeConnection members should be replaced by functions which construct the new connection object.

If the arguments of the wxTCPConnection constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a buffer and size of the buffer for the connection object to use in transactions.

wxTCPConnection::wxTCPConnection (void * *buffer,* size_t *size*)

Constructs a connection object.

If no user-defined connection object is to be derived from wxTCPConnection, then the constructor should not be called directly, since the default connection object will be provided on requesting (or accepting) a connection.

However, if the user defines his or her own derived connection object, the wxTCPServer::OnAcceptConnection and/or wxTCPClient::OnMakeConnection members should be replaced by functions which construct the new connection object.

If the arguments of the wxTCPConnection constructor are void, then a default buffer is associated with the connection. Otherwise, the programmer must provide a buffer and size of the buffer for the connection object to use in transactions.

21.758.3 Member Function Documentation

bool wxTCPConnection::Advise (const wxString & *item,* const void * *data,* size_t *size,* wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns true if successful.

bool wxTCPConnection::Advise (const wxString & *item,* const char * *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns true if successful.

bool wxTCPConnection::Advise (const wxString & *item,* const wchar_t * *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns true if successful.

bool wxTCPConnection::Advise (const wxString & *item,* const wxString *data*)

Called by the server application to advise the client of a change in the data associated with the given item.

Causes the client connection's OnAdvise() member to be called.

Returns true if successful.

virtual bool wxTCPConnection::Disconnect ( ) [virtual]

Called by the client or server application to disconnect from the other program; it causes the OnDisconnect() message to be sent to the corresponding connection object in the other program.

The default behaviour of **OnDisconnect** is to delete the connection, but the calling application must explicitly delete its side of the connection having called **Disconnect**.

Returns true if successful.

bool wxTCPConnection::Execute (const void * *data*, size_t *size*, wxIPCFormat *format* = wxIPC_PRIVATE)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns true if successful.

bool wxTCPConnection::Execute (const char * *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns true if successful.

bool wxTCPConnection::Execute (const wchar_t * *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns true if successful.

bool wxTCPConnection::Execute (const wxString *data*)

Called by the client application to execute a command on the server.

Can also be used to transfer arbitrary data to the server (similar to Poke() in that respect). Causes the server connection's OnExecute() member to be called.

Returns true if successful.

virtual bool wxTCPConnection::OnAdvise (const wxString & *topic*, const wxString & *item*, const void * *data*, size_t *size*, wxIPCFormat *format*) `[virtual]`

Message sent to the client application when the server notifies it of a change in the data associated with the given item.

virtual bool wxTCPConnection::OnDisconnect ( ) `[virtual]`

Message sent to the client or server application when the other application notifies it to delete the connection.

Default behaviour is to delete the connection object.

virtual bool wxTCPConnection::OnExecute (const wxString & *topic*, const void * *data*, size_t *size*, wxIPCFormat *format*) `[virtual]`

Message sent to the server application when the client notifies it to execute the given data.

Note that there is no item associated with this message.

virtual bool wxTCPConnection::OnPoke (const wxString & *topic,* const wxString & *item,* const void * *data,* size_t *size,* wxIPCFormat *format*) [virtual]

Message sent to the server application when the client notifies it to accept the given data.

virtual const void* wxTCPConnection::OnRequest (const wxString & *topic,* const wxString & *item,* size_t * *size,* wxIPCFormat *format*) [virtual]

Message sent to the server application when the client calls Request().

The server should respond by returning a character string from **OnRequest**, or NULL to indicate no data.

virtual bool wxTCPConnection::OnStartAdvise (const wxString & *topic,* const wxString & *item*) [virtual]

Message sent to the server application by the client, when the client wishes to start an 'advise loop' for the given topic and item.

The server can refuse to participate by returning false.

virtual bool wxTCPConnection::OnStopAdvise (const wxString & *topic,* const wxString & *item*) [virtual]

Message sent to the server application by the client, when the client wishes to stop an 'advise loop' for the given topic and item.

The server can refuse to stop the advise loop by returning false, although this doesn't have much meaning in practice.

bool wxTCPConnection::Poke (const wxString & *item,* const void * *data,* size_t *size,* wxIPCFormat *format =* wxIPC_PRIVATE)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. Returns true if successful.

bool wxTCPConnection::Poke (const wxString & *item,* const char * *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. Returns true if successful.

bool wxTCPConnection::Poke (const wxString & *item,* const wchar_t * *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. Returns true if successful.

bool wxTCPConnection::Poke (const wxString & *item,* const wxString *data*)

Called by the client application to poke data into the server.

Can be used to transfer arbitrary data to the server. Causes the server connection's OnPoke() member to be called. Returns true if successful.

virtual const void* wxTCPConnection::Request (const wxString & *item*, size_t * *size* = 0, wxIPCFormat *format* = wxIPC_TEXT) [virtual]

Called by the client application to request data from the server.

Causes the server connection's OnRequest() member to be called.

Returns a character string (actually a pointer to the connection's buffer) if successful, NULL otherwise.

virtual bool wxTCPConnection::StartAdvise (const wxString & *item*) [virtual]

Called by the client application to ask if an advise loop can be started with the server.

Causes the server connection's OnStartAdvise() member to be called. Returns true if the server okays it, false otherwise.

virtual bool wxTCPConnection::StopAdvise (const wxString & *item*) [virtual]

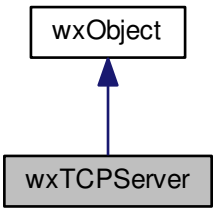
Called by the client application to ask if an advise loop can be stopped.

Causes the server connection's OnStopAdvise() member to be called. Returns true if the server okays it, false otherwise.

21.759 wxTCPServer Class Reference

`#include <wx/sckipc.h>`

Inheritance diagram for wxTCPServer:



21.759.1 Detailed Description

A wxTCPServer object represents the server part of a client-server conversation.

It emulates a DDE-style protocol, but uses TCP/IP which is available on most platforms.

A DDE-based implementation for Windows is available using wxDDEServer.

Library: wxNet

Category: Networking

See also

wxTCPClient, wxTCPConnection, Interprocess Communication

Public Member Functions

- wxTCPServer ()

  Constructs a server object.
- virtual bool Create (const wxString &service)

  Registers the server using the given service name.
- virtual wxConnectionBase * OnAcceptConnection (const wxString &topic)

  *When a client calls **MakeConnection**, the server receives the message and this member is called.*

Additional Inherited Members

21.759.2 Constructor & Destructor Documentation

wxTCPServer::wxTCPServer ( )

Constructs a server object.

21.759.3 Member Function Documentation

virtual bool wxTCPServer::Create (const wxString & *service*) `[virtual]`

Registers the server using the given service name.

Under Unix, the string must contain an integer id which is used as an Internet port number. false is returned if the call failed (for example, the port number is already in use).

virtual wxConnectionBase* wxTCPServer::OnAcceptConnection (const wxString & *topic*) `[virtual]`

When a client calls **MakeConnection**, the server receives the message and this member is called.

The application should derive a member to intercept this message and return a connection object of either the standard wxTCPConnection type, or of a user-derived type. If the topic is "STDIO", the application may wish to refuse the connection. Under Unix, when a server is created the OnAcceptConnection message is always sent for standard input and output.

21.760 wxTempFile Class Reference

`#include <wx/file.h>`

21.760.1 Detailed Description

wxTempFile provides a relatively safe way to replace the contents of the existing file.

The name is explained by the fact that it may be also used as just a temporary file if you don't replace the old file contents.

Usually, when a program replaces the contents of some file it first opens it for writing, thus losing all of the old data and then starts recreating it. This approach is not very safe because during the regeneration of the file bad things may happen: the program may find that there is an internal error preventing it from completing file generation, the

user may interrupt it (especially if file generation takes long time) and, finally, any other external interrupts (power supply failure or a disk error) will leave you without either the original file or the new one.

wxTempFile addresses this problem by creating a temporary file which is meant to replace the original file - but only after it is fully written. So, if the user interrupts the program during the file generation, the old file won't be lost. Also, if the program discovers itself that it doesn't want to replace the old file there is no problem - in fact, wxTempFile will **not** replace the old file by default, you should explicitly call wxTempFile::Commit() to do it. Calling wxTempFile::Discard() explicitly discards any modifications: it closes and deletes the temporary file and leaves the original file unchanged. If you call neither Commit() nor Discard(), the destructor will call Discard() automatically.

To summarize: if you want to replace another file, create an instance of wxTempFile passing the name of the file to be replaced to the constructor. (You may also use default constructor and pass the file name to wxTempFile::Open.) Then you can write to wxTempFile using wxFile-like functions and later call wxTempFile::Commit() to replace the old file (and close this one) or call wxTempFile::Discard() to cancel the modifications.

Library: wxBase

Category: File Handling

Public Member Functions

- wxTempFile (const wxString &strName)

  Associates wxTempFile with the file to be replaced and opens it.
- ~wxTempFile ()

  Destructor calls Discard() if temporary file is still open.
- bool Commit ()

  Validate changes: deletes the old file of name m_strName and renames the new file to the old name.
- void Discard ()

  Discard changes: the old file contents are not changed, the temporary file is deleted.
- bool Flush ()

  Flush the data written to the file to disk.
- bool IsOpened () const

  Returns true if the file was successfully opened.
- wxFileOffset Length () const

  Returns the length of the file.
- bool Open (const wxString &strName)

  Open the temporary file, returns true on success, false if an error occurred.
- wxFileOffset Seek (wxFileOffset ofs, wxSeekMode mode=wxFromStart)

  Seeks to the specified position.
- wxFileOffset Tell () const

  Returns the current position or wxInvalidOffset if file is not opened or if another error occurred.
- bool Write (const wxString &str, const wxMBConv &conv=wxConvUTF8)

  Write to the file, return true on success, false on failure.

21.760.2 Constructor & Destructor Documentation

wxTempFile::wxTempFile (const wxString & *strName*)

Associates wxTempFile with the file to be replaced and opens it.

Warning

You should use IsOpened() to verify that the constructor succeeded.

wxTempFile::~wxTempFile ( )

Destructor calls Discard() if temporary file is still open.

21.760.3 Member Function Documentation

bool wxTempFile::Commit ( )

Validate changes: deletes the old file of name m_strName and renames the new file to the old name.

Returns true if both actions succeeded.

If false is returned it may unfortunately mean two quite different things: either that the old file couldn't be deleted or that the new file couldn't be renamed to the old name.

void wxTempFile::Discard ( )

Discard changes: the old file contents are not changed, the temporary file is deleted.

bool wxTempFile::Flush ( )

Flush the data written to the file to disk.

This simply calls wxFile::Flush() for the underlying file and may be necessary with file systems such as XFS and Ext4 under Linux. Calling this function may however have serious performance implications and also is not necessary with many other file systems so it is not done by default – but you can call it before calling Commit() to absolutely ensure that the data was indeed written to the disk correctly.

bool wxTempFile::IsOpened ( ) const

Returns true if the file was successfully opened.

wxFileOffset wxTempFile::Length ( ) const

Returns the length of the file.

This method may return wxInvalidOffset if the length couldn't be determined or 0 even for non-empty files if the file is not seekable.

In general, the only way to determine if the file for which this function returns 0 is really empty or not is to try reading from it.

bool wxTempFile::Open (const wxString & *strName*)

Open the temporary file, returns true on success, false if an error occurred.

strName is the name of file to be replaced. The temporary file is always created in the directory where *strName* is. In particular, if *strName* doesn't include the path, it is created in the current directory and the program should have write access to it for the function to succeed.

wxFileOffset wxTempFile::Seek (wxFileOffset *ofs*, wxSeekMode *mode* = wxFromStart)

Seeks to the specified position.

wxFileOffset wxTempFile::Tell ( ) const

Returns the current position or wxInvalidOffset if file is not opened or if another error occurred.

bool wxTempFile::Write (const wxString & *str,* const wxMBConv & *conv* = `wxConvUTF8`)

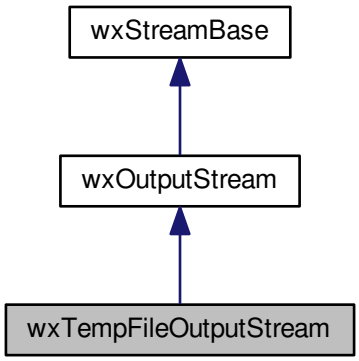
Write to the file, return true on success, false on failure.

The second argument is only meaningful in Unicode build of wxWidgets when *conv* is used to convert *str* to multibyte representation.

21.761 wxTempFileOutputStream Class Reference

`#include <wx/wfstream.h>`

Inheritance diagram for wxTempFileOutputStream:



21.761.1 Detailed Description

wxTempFileOutputStream is an output stream based on wxTempFile.

It provides a relatively safe way to replace the contents of the existing file.

Library: wxBase

Category: Streams

See also

wxTempFile

Public Member Functions

- wxTempFileOutputStream (const wxString &fileName)

Associates wxTempFileOutputStream with the file to be replaced and opens it.

- virtual bool Commit ()

  Validate changes: deletes the old file of the given name and renames the new file to the old name.

- virtual void Discard ()

  Discard changes: the old file contents are not changed, the temporary file is deleted.

Additional Inherited Members

21.761.2 Constructor & Destructor Documentation

wxTempFileOutputStream::wxTempFileOutputStream (const wxString & *fileName*)

Associates wxTempFileOutputStream with the file to be replaced and opens it.

Warning

You should use wxStreamBase::IsOk() to verify if the constructor succeeded.

Call Commit() or wxOutputStream::Close() to replace the old file and close this one. Calling Discard() (or allowing the destructor to do it) will discard the changes.

21.761.3 Member Function Documentation

virtual bool wxTempFileOutputStream::Commit ( ) [virtual]

Validate changes: deletes the old file of the given name and renames the new file to the old name.

Returns true if both actions succeeded.

If false is returned it may unfortunately mean two quite different things: either that either the old file couldn't be deleted or that the new file couldn't be renamed to the old name.

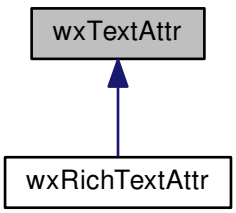
virtual void wxTempFileOutputStream::Discard ( ) [virtual]

Discard changes: the old file contents are not changed, the temporary file is deleted.

21.762 wxTextAttr Class Reference

`#include <wx/textctrl.h>`

Inheritance diagram for wxTextAttr:



21.762.1 Detailed Description

wxTextAttr represents the character and paragraph attributes, or style, for a range of text in a wxTextCtrl or wxRichTextCtrl.

When setting up a wxTextAttr object, pass a bitlist mask to wxTextAttr::SetFlags() to indicate which style elements should be changed. As a convenience, when you call a setter such as SetFont, the relevant bit will be set.

Library: wxCore

Category: Rich Text

See also

wxTextCtrl, wxRichTextCtrl

Public Member Functions

- bool Apply (const wxTextAttr &style, const wxTextAttr *compareWith=NULL)

  Applies the attributes in style to the original object, but not those attributes from style that are the same as those in compareWith (if passed).
- void Merge (const wxTextAttr &overlay)

  Copies all defined/valid properties from overlay to current object.
- bool EqPartial (const wxTextAttr &attr, bool weakTest=true) const

  Partial equality test.
- void operator= (const wxTextAttr &attr)

  Assignment from a wxTextAttr object.

- wxTextAttr ()

  Constructors.
- wxTextAttr (const wxColour &colText, const wxColour &colBack=wxNullColour, const wxFont &font=wxNullFont, wxTextAttrAlignment alignment=wxTEXT_ALIGNMENT_DEFAULT)

  Constructors.
- wxTextAttr (const wxTextAttr &attr)

  Constructors.

GetXXX functions

- wxTextAttrAlignment GetAlignment () const

  Returns the alignment flags.
- const wxColour & GetBackgroundColour () const

  Returns the background colour.
- const wxString & GetBulletFont () const

  Returns a string containing the name of the font associated with the bullet symbol.
- const wxString & GetBulletName () const

  Returns the standard bullet name, applicable if the bullet style is wxTEXT_ATTR_BULLET_STYLE_STANDARD.
- int GetBulletNumber () const

  Returns the bullet number.
- int GetBulletStyle () const

  Returns the bullet style.
- const wxString & GetBulletText () const

  Returns the bullet text, which could be a symbol, or (for example) cached outline text.
- const wxString & GetCharacterStyleName () const

  Returns the name of the character style.

- long GetFlags () const
  Returns flags indicating which attributes are applicable.
- wxFont GetFont () const
  Creates and returns a font specified by the font attributes in the wxTextAttr object.
- bool GetFontAttributes (const wxFont &font, int flags=wxTEXT_ATTR_FONT)
  Gets the font attributes from the given font, using only the attributes specified by flags.
- wxFontEncoding GetFontEncoding () const
  Returns the font encoding.
- const wxString & GetFontFaceName () const
  Returns the font face name.
- wxFontFamily GetFontFamily () const
  Returns the font family.
- int GetFontSize () const
  Returns the font size in points.
- wxFontStyle GetFontStyle () const
  Returns the font style.
- bool GetFontUnderlined () const
  Returns true if the font is underlined.
- wxFontWeight GetFontWeight () const
  Returns the font weight.
- long GetLeftIndent () const
  Returns the left indent in tenths of a millimetre.
- long GetLeftSubIndent () const
  Returns the left sub-indent in tenths of a millimetre.
- int GetLineSpacing () const
  Returns the line spacing value, one of wxTextAttrLineSpacing values.
- const wxString & GetListStyleName () const
  Returns the name of the list style.
- int GetOutlineLevel () const
  Returns the outline level.
- int GetParagraphSpacingAfter () const
  Returns the space in tenths of a millimeter after the paragraph.
- int GetParagraphSpacingBefore () const
  Returns the space in tenths of a millimeter before the paragraph.
- const wxString & GetParagraphStyleName () const
  Returns the name of the paragraph style.
- long GetRightIndent () const
  Returns the right indent in tenths of a millimeter.
- const wxArrayInt & GetTabs () const
  Returns an array of tab stops, each expressed in tenths of a millimeter.
- const wxColour & GetTextColour () const
  Returns the text foreground colour.
- int GetTextEffectFlags () const
  Returns the text effect bits of interest.
- int GetTextEffects () const
  Returns the text effects, a bit list of styles.
- const wxString & GetURL () const
  Returns the URL for the content.

HasXXX and IsXXX functions

- bool HasAlignment () const
  Returns true if the attribute object specifies alignment.
- bool HasBackgroundColour () const
  Returns true if the attribute object specifies a background colour.
- bool HasBulletName () const
  Returns true if the attribute object specifies a standard bullet name.
- bool HasBulletNumber () const
  Returns true if the attribute object specifies a bullet number.

- bool HasBulletStyle () const

  Returns true if the attribute object specifies a bullet style.
- bool HasBulletText () const

  Returns true if the attribute object specifies bullet text (usually specifying a symbol).
- bool HasCharacterStyleName () const

  Returns true if the attribute object specifies a character style name.
- bool HasFlag (long flag) const

  Returns true if the flag is present in the attribute object's flag bitlist.
- bool HasFont () const

  Returns true if the attribute object specifies any font attributes.
- bool HasFontEncoding () const

  Returns true if the attribute object specifies an encoding.
- bool HasFontFaceName () const

  Returns true if the attribute object specifies a font face name.
- bool HasFontFamily () const

  Returns true if the attribute object specifies a font family.
- bool HasFontItalic () const

  Returns true if the attribute object specifies italic style.
- bool HasFontSize () const

  Returns true if the attribute object specifies a font point or pixel size.
- bool HasFontPointSize () const

  Returns true if the attribute object specifies a font point size.
- bool HasFontPixelSize () const

  Returns true if the attribute object specifies a font pixel size.
- bool HasFontUnderlined () const

  Returns true if the attribute object specifies either underlining or no underlining.
- bool HasFontWeight () const

  Returns true if the attribute object specifies font weight (bold, light or normal).
- bool HasLeftIndent () const

  Returns true if the attribute object specifies a left indent.
- bool HasLineSpacing () const

  Returns true if the attribute object specifies line spacing.
- bool HasListStyleName () const

  Returns true if the attribute object specifies a list style name.
- bool HasOutlineLevel () const

  Returns true if the attribute object specifies an outline level.
- bool HasPageBreak () const

  Returns true if the attribute object specifies a page break before this paragraph.
- bool HasParagraphSpacingAfter () const

  Returns true if the attribute object specifies spacing after a paragraph.
- bool HasParagraphSpacingBefore () const

  Returns true if the attribute object specifies spacing before a paragraph.
- bool HasParagraphStyleName () const

  Returns true if the attribute object specifies a paragraph style name.
- bool HasRightIndent () const

  Returns true if the attribute object specifies a right indent.
- bool HasTabs () const

  Returns true if the attribute object specifies tab stops.
- bool HasTextColour () const

  Returns true if the attribute object specifies a text foreground colour.
- bool HasTextEffects () const

  Returns true if the attribute object specifies text effects.
- bool HasURL () const

  Returns true if the attribute object specifies a URL.
- bool IsCharacterStyle () const

  Returns true if the object represents a character style, that is, the flags specify a font or a text background or foreground colour.
- bool IsDefault () const

  Returns false if we have any attributes set, true otherwise.

- bool IsParagraphStyle () const

  Returns true if the object represents a paragraph style, that is, the flags specify alignment, indentation, tabs, paragraph spacing, or bullet style.

SetXXX functions

- void SetAlignment (wxTextAttrAlignment alignment)

  Sets the paragraph alignment.
- void SetBackgroundColour (const wxColour &colBack)

  Sets the background colour.
- void SetBulletFont (const wxString &font)

  Sets the name of the font associated with the bullet symbol.
- void SetBulletName (const wxString &name)

  Sets the standard bullet name, applicable if the bullet style is wxTEXT_ATTR_BULLET_STYLE_STANDARD.
- void SetBulletNumber (int n)

  Sets the bullet number.
- void SetBulletStyle (int style)

  Sets the bullet style.
- void SetBulletText (const wxString &text)

  Sets the bullet text, which could be a symbol, or (for example) cached outline text.
- void SetCharacterStyleName (const wxString &name)

  Sets the character style name.
- void SetFlags (long flags)

  Sets the flags determining which styles are being specified.
- void SetFont (const wxFont &font, int flags=wxTEXT_ATTR_FONT &∼wxTEXT_ATTR_FONT_PIXEL_↩ SIZE)

  Sets the attributes for the given font.
- void SetFontEncoding (wxFontEncoding encoding)

  Sets the font encoding.
- void SetFontFaceName (const wxString &faceName)

  Sets the font face name.
- void SetFontFamily (wxFontFamily family)

  Sets the font family.
- void SetFontSize (int pointSize)

  Sets the font size in points.
- void SetFontPointSize (int pointSize)

  Sets the font size in points.
- void SetFontPixelSize (int pixelSize)

  Sets the font size in pixels.
- void SetFontStyle (wxFontStyle fontStyle)

  Sets the font style (normal, italic or slanted).
- void SetFontUnderlined (bool underlined)

  Sets the font underlining.
- void SetFontWeight (wxFontWeight fontWeight)

  Sets the font weight.
- void SetLeftIndent (int indent, int subIndent=0)

  Sets the left indent and left subindent in tenths of a millimetre.
- void SetLineSpacing (int spacing)

  Sets the line spacing.
- void SetListStyleName (const wxString &name)

  Sets the list style name.
- void SetOutlineLevel (int level)

  Specifies the outline level.
- void SetPageBreak (bool pageBreak=true)

  Specifies a page break before this paragraph.
- void SetParagraphSpacingAfter (int spacing)

  Sets the spacing after a paragraph, in tenths of a millimetre.
- void SetParagraphSpacingBefore (int spacing)

  Sets the spacing before a paragraph, in tenths of a millimetre.

- void SetParagraphStyleName (const wxString &name)
  Sets the name of the paragraph style.
- void SetRightIndent (int indent)
  Sets the right indent in tenths of a millimetre.
- void SetTabs (const wxArrayInt &tabs)
  Sets the tab stops, expressed in tenths of a millimetre.
- void SetTextColour (const wxColour &colText)
  Sets the text foreground colour.
- void SetTextEffectFlags (int flags)
  Sets the text effect bits of interest.
- void SetTextEffects (int effects)
  Sets the text effects, a bit list of styles.
- void SetURL (const wxString &url)
  Sets the URL for the content.

Static Public Member Functions

- static wxTextAttr Merge (const wxTextAttr &base, const wxTextAttr &overlay)
  Creates a new `wxTextAttr` which is a merge of base and overlay.

21.762.2 Constructor & Destructor Documentation

wxTextAttr::wxTextAttr ( )

Constructors.

wxTextAttr::wxTextAttr (const wxColour & *colText*, const wxColour & *colBack* = wxNullColour, const wxFont & *font* = wxNullFont, wxTextAttrAlignment *alignment* = wxTEXT_ALIGNMENT_DEFAULT)

Constructors.

wxTextAttr::wxTextAttr (const wxTextAttr & *attr*)

Constructors.

21.762.3 Member Function Documentation

bool wxTextAttr::Apply (const wxTextAttr & *style*, const wxTextAttr * *compareWith* = `NULL`)

Applies the attributes in *style* to the original object, but not those attributes from *style* that are the same as those in *compareWith* (if passed).

bool wxTextAttr::EqPartial (const wxTextAttr & *attr*, bool *weakTest* = `true`) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *attr* are present. If *weakTest* is false, the function will fail if an attribute is present in *attr* but not in this object.

wxTextAttrAlignment wxTextAttr::GetAlignment ( ) const

Returns the alignment flags.

See wxTextAttrAlignment for a list of available styles.

const wxColour& wxTextAttr::GetBackgroundColour ( ) const

Returns the background colour.

const wxString& wxTextAttr::GetBulletFont ( ) const

Returns a string containing the name of the font associated with the bullet symbol.

Only valid for attributes with wxTEXT_ATTR_BULLET_SYMBOL.

const wxString& wxTextAttr::GetBulletName ( ) const

Returns the standard bullet name, applicable if the bullet style is wxTEXT_ATTR_BULLET_STYLE_STANDARD.

Currently the following standard bullet names are supported:

- `standard/circle`
- `standard/square`
- `standard/diamond`
- `standard/triangle`

Note

For wxRichTextCtrl users only: if you wish your rich text controls to support further bullet graphics, you can derive a class from wxRichTextRenderer or wxRichTextStdRenderer, override `DrawStandardBullet` and `EnumerateStandardBulletNames`, and set an instance of the class using wxRichTextBuffer::SetRenderer.

int wxTextAttr::GetBulletNumber ( ) const

Returns the bullet number.

int wxTextAttr::GetBulletStyle ( ) const

Returns the bullet style.

See wxTextAttrBulletStyle for a list of available styles.

const wxString& wxTextAttr::GetBulletText ( ) const

Returns the bullet text, which could be a symbol, or (for example) cached outline text.

const wxString& wxTextAttr::GetCharacterStyleName ( ) const

Returns the name of the character style.

long wxTextAttr::GetFlags ( ) const

Returns flags indicating which attributes are applicable.

See SetFlags() for a list of available flags.

wxFont wxTextAttr::GetFont ( ) const

Creates and returns a font specified by the font attributes in the wxTextAttr object.

Note that wxTextAttr does not store a wxFont object, so this is only a temporary font.

For greater efficiency, access the font attributes directly.

bool wxTextAttr::GetFontAttributes (const wxFont & *font,* int *flags* = wxTEXT_ATTR_FONT)

Gets the font attributes from the given font, using only the attributes specified by *flags*.

wxFontEncoding wxTextAttr::GetFontEncoding ( ) const

Returns the font encoding.

const wxString& wxTextAttr::GetFontFaceName ( ) const

Returns the font face name.

wxFontFamily wxTextAttr::GetFontFamily ( ) const

Returns the font family.

int wxTextAttr::GetFontSize ( ) const

Returns the font size in points.

wxFontStyle wxTextAttr::GetFontStyle ( ) const

Returns the font style.

bool wxTextAttr::GetFontUnderlined ( ) const

Returns true if the font is underlined.

wxFontWeight wxTextAttr::GetFontWeight ( ) const

Returns the font weight.

long wxTextAttr::GetLeftIndent ( ) const

Returns the left indent in tenths of a millimetre.

long wxTextAttr::GetLeftSubIndent ( ) const

Returns the left sub-indent in tenths of a millimetre.

int wxTextAttr::GetLineSpacing ( ) const

Returns the line spacing value, one of wxTextAttrLineSpacing values.

const wxString& wxTextAttr::GetListStyleName ( ) const

Returns the name of the list style.

int wxTextAttr::GetOutlineLevel ( ) const

Returns the outline level.

int wxTextAttr::GetParagraphSpacingAfter ( ) const

Returns the space in tenths of a millimeter after the paragraph.

int wxTextAttr::GetParagraphSpacingBefore ( ) const

Returns the space in tenths of a millimeter before the paragraph.

const wxString& wxTextAttr::GetParagraphStyleName ( ) const

Returns the name of the paragraph style.

long wxTextAttr::GetRightIndent ( ) const

Returns the right indent in tenths of a millimeter.

const wxArrayInt& wxTextAttr::GetTabs ( ) const

Returns an array of tab stops, each expressed in tenths of a millimeter.

Each stop is measured from the left margin and therefore each value must be larger than the last.

const wxColour& wxTextAttr::GetTextColour ( ) const

Returns the text foreground colour.

int wxTextAttr::GetTextEffectFlags ( ) const

Returns the text effect bits of interest.

See SetFlags() for further information.

int wxTextAttr::GetTextEffects ( ) const

Returns the text effects, a bit list of styles.

See SetTextEffects() for details.

const wxString& wxTextAttr::GetURL ( ) const

Returns the URL for the content.

Content with *wxTEXT_ATTR_URL* style causes wxRichTextCtrl to show a hand cursor over it, and wxRichTextCtrl generates a wxTextUrlEvent when the content is clicked.

bool wxTextAttr::HasAlignment ( ) const

Returns true if the attribute object specifies alignment.

bool wxTextAttr::HasBackgroundColour ( ) const

Returns true if the attribute object specifies a background colour.

bool wxTextAttr::HasBulletName ( ) const

Returns true if the attribute object specifies a standard bullet name.

bool wxTextAttr::HasBulletNumber ( ) const

Returns true if the attribute object specifies a bullet number.

bool wxTextAttr::HasBulletStyle ( ) const

Returns true if the attribute object specifies a bullet style.

bool wxTextAttr::HasBulletText ( ) const

Returns true if the attribute object specifies bullet text (usually specifying a symbol).

bool wxTextAttr::HasCharacterStyleName ( ) const

Returns true if the attribute object specifies a character style name.

bool wxTextAttr::HasFlag (long *flag*) const

Returns true if the *flag* is present in the attribute object's flag bitlist.

bool wxTextAttr::HasFont ( ) const

Returns true if the attribute object specifies any font attributes.

bool wxTextAttr::HasFontEncoding ( ) const

Returns true if the attribute object specifies an encoding.

bool wxTextAttr::HasFontFaceName ( ) const

Returns true if the attribute object specifies a font face name.

bool wxTextAttr::HasFontFamily ( ) const

Returns true if the attribute object specifies a font family.

bool wxTextAttr::HasFontItalic ( ) const

Returns true if the attribute object specifies italic style.

bool wxTextAttr::HasFontPixelSize ( ) const

Returns true if the attribute object specifies a font pixel size.

bool wxTextAttr::HasFontPointSize ( ) const

Returns true if the attribute object specifies a font point size.

bool wxTextAttr::HasFontSize ( ) const

Returns true if the attribute object specifies a font point or pixel size.

bool wxTextAttr::HasFontUnderlined ( ) const

Returns true if the attribute object specifies either underlining or no underlining.

bool wxTextAttr::HasFontWeight ( ) const

Returns true if the attribute object specifies font weight (bold, light or normal).

bool wxTextAttr::HasLeftIndent ( ) const

Returns true if the attribute object specifies a left indent.

bool wxTextAttr::HasLineSpacing ( ) const

Returns true if the attribute object specifies line spacing.

bool wxTextAttr::HasListStyleName ( ) const

Returns true if the attribute object specifies a list style name.

bool wxTextAttr::HasOutlineLevel ( ) const

Returns true if the attribute object specifies an outline level.

bool wxTextAttr::HasPageBreak ( ) const

Returns true if the attribute object specifies a page break before this paragraph.

bool wxTextAttr::HasParagraphSpacingAfter ( ) const

Returns true if the attribute object specifies spacing after a paragraph.

bool wxTextAttr::HasParagraphSpacingBefore ( ) const

Returns true if the attribute object specifies spacing before a paragraph.

bool wxTextAttr::HasParagraphStyleName ( ) const

Returns true if the attribute object specifies a paragraph style name.

bool wxTextAttr::HasRightIndent ( ) const

Returns true if the attribute object specifies a right indent.

bool wxTextAttr::HasTabs ( ) const

Returns true if the attribute object specifies tab stops.

bool wxTextAttr::HasTextColour ( ) const

Returns true if the attribute object specifies a text foreground colour.

bool wxTextAttr::HasTextEffects ( ) const

Returns true if the attribute object specifies text effects.

bool wxTextAttr::HasURL ( ) const

Returns true if the attribute object specifies a URL.

bool wxTextAttr::IsCharacterStyle ( ) const

Returns true if the object represents a character style, that is, the flags specify a font or a text background or foreground colour.

bool wxTextAttr::IsDefault ( ) const

Returns false if we have any attributes set, true otherwise.

bool wxTextAttr::IsParagraphStyle ( ) const

Returns true if the object represents a paragraph style, that is, the flags specify alignment, indentation, tabs, paragraph spacing, or bullet style.

void wxTextAttr::Merge (const wxTextAttr & *overlay*)

Copies all defined/valid properties from overlay to current object.

static wxTextAttr wxTextAttr::Merge (const wxTextAttr & *base*, const wxTextAttr & *overlay*) [static]

Creates a new `wxTextAttr` which is a merge of *base* and *overlay*.

Properties defined in *overlay* take precedence over those in *base*. Properties undefined/invalid in both are undefined in the result.

void wxTextAttr::operator= (const wxTextAttr & *attr*)

Assignment from a wxTextAttr object.

void wxTextAttr::SetAlignment (wxTextAttrAlignment *alignment*)

Sets the paragraph alignment.

See wxTextAttrAlignment enumeration values.

Of these, wxTEXT_ALIGNMENT_JUSTIFIED is unimplemented. In future justification may be supported when printing or previewing, only.

void wxTextAttr::SetBackgroundColour (const wxColour & *colBack*)

Sets the background colour.

void wxTextAttr::SetBulletFont (const wxString & *font*)

Sets the name of the font associated with the bullet symbol.

Only valid for attributes with wxTEXT_ATTR_BULLET_SYMBOL.

void wxTextAttr::SetBulletName (const wxString & *name*)

Sets the standard bullet name, applicable if the bullet style is wxTEXT_ATTR_BULLET_STYLE_STANDARD.

See GetBulletName() for a list of supported names, and how to expand the range of supported types.

void wxTextAttr::SetBulletNumber (int *n*)

Sets the bullet number.

void wxTextAttr::SetBulletStyle (int *style*)

Sets the bullet style.

The wxTextAttrBulletStyle enumeration values are all supported, except for wxTEXT_ATTR_BULLET_STYLE_BI↩TMAP.

void wxTextAttr::SetBulletText (const wxString & *text*)

Sets the bullet text, which could be a symbol, or (for example) cached outline text.

void wxTextAttr::SetCharacterStyleName (const wxString & *name*)

Sets the character style name.

void wxTextAttr::SetFlags (long *flags*)

Sets the flags determining which styles are being specified.

The wxTextAttrFlags values can be passed in a bitlist.

void wxTextAttr::SetFont (const wxFont & *font*, int *flags* = wxTEXT_ATTR_FONT &~wxTEXT_ATTR_FONT_PIXEL_SIZE)

Sets the attributes for the given font.

Note that wxTextAttr does not store an actual wxFont object.

void wxTextAttr::SetFontEncoding (wxFontEncoding *encoding*)

Sets the font encoding.

void wxTextAttr::SetFontFaceName (const wxString & *faceName*)

Sets the font face name.

void wxTextAttr::SetFontFamily (wxFontFamily *family*)

Sets the font family.

void wxTextAttr::SetFontPixelSize (int *pixelSize*)

Sets the font size in pixels.

void wxTextAttr::SetFontPointSize (int *pointSize*)

Sets the font size in points.

void wxTextAttr::SetFontSize (int *pointSize*)

Sets the font size in points.

void wxTextAttr::SetFontStyle (wxFontStyle *fontStyle*)

Sets the font style (normal, italic or slanted).

void wxTextAttr::SetFontUnderlined (bool *underlined*)

Sets the font underlining.

void wxTextAttr::SetFontWeight (wxFontWeight *fontWeight*)

Sets the font weight.

void wxTextAttr::SetLeftIndent (int *indent,* int *subIndent* = `0`)

Sets the left indent and left subindent in tenths of a millimetre.

The sub-indent is an offset from the left of the paragraph, and is used for all but the first line in a paragraph.

A positive value will cause the first line to appear to the left of the subsequent lines, and a negative value will cause the first line to be indented relative to the subsequent lines.

wxRichTextBuffer uses indentation to render a bulleted item. The left indent is the distance between the margin and the bullet. The content of the paragraph, including the first line, starts at leftMargin + leftSubIndent. So the distance between the left edge of the bullet and the left of the actual paragraph is leftSubIndent.

void wxTextAttr::SetLineSpacing (int *spacing*)

Sets the line spacing.

spacing is a multiple, where 10 means single-spacing, 15 means 1.5 spacing, and 20 means double spacing. The wxTextAttrLineSpacing values are defined for convenience.

void wxTextAttr::SetListStyleName (const wxString & *name*)

Sets the list style name.

void wxTextAttr::SetOutlineLevel (int *level*)

Specifies the outline level.

Zero represents normal text. At present, the outline level is not used, but may be used in future for determining list levels and for applications that need to store document structure information.

void wxTextAttr::SetPageBreak (bool *pageBreak* = `true`)

Specifies a page break before this paragraph.

void wxTextAttr::SetParagraphSpacingAfter (int *spacing*)

Sets the spacing after a paragraph, in tenths of a millimetre.

void wxTextAttr::SetParagraphSpacingBefore (int *spacing*)

Sets the spacing before a paragraph, in tenths of a millimetre.

void wxTextAttr::SetParagraphStyleName (const wxString & *name*)

Sets the name of the paragraph style.

void wxTextAttr::SetRightIndent (int *indent*)

Sets the right indent in tenths of a millimetre.

void wxTextAttr::SetTabs (const wxArrayInt & *tabs*)

Sets the tab stops, expressed in tenths of a millimetre.

Each stop is measured from the left margin and therefore each value must be larger than the last.

void wxTextAttr::SetTextColour (const wxColour & *colText*)

Sets the text foreground colour.

void wxTextAttr::SetTextEffectFlags (int *flags*)

Sets the text effect bits of interest.

You should also pass wxTEXT_ATTR_EFFECTS to SetFlags(). See SetFlags() for further information.

void wxTextAttr::SetTextEffects (int *effects*)

Sets the text effects, a bit list of styles.

The wxTextAttrEffects enumeration values can be used.

Of these, only wxTEXT_ATTR_EFFECT_CAPITALS, wxTEXT_ATTR_EFFECT_STRIKETHROUGH, wxTEXT_↩ ATTR_EFFECT_SUPERSCRIPT and wxTEXT_ATTR_EFFECT_SUBSCRIPT are implemented.

wxTEXT_ATTR_EFFECT_CAPITALS capitalises text when displayed (leaving the case of the actual buffer text unchanged), and wxTEXT_ATTR_EFFECT_STRIKETHROUGH draws a line through text.

To set effects, you should also pass wxTEXT_ATTR_EFFECTS to SetFlags(), and call SetTextEffectFlags() with the styles (taken from the above set) that you are interested in setting.

void wxTextAttr::SetURL (const wxString & *url*)

Sets the URL for the content.

Sets the wxTEXT_ATTR_URL style; content with this style causes wxRichTextCtrl to show a hand cursor over it, and wxRichTextCtrl generates a wxTextUrlEvent when the content is clicked.

21.763 wxTextAttrBorder Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.763.1 Detailed Description

A class representing a rich text object border.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl, wxRichTextAttrBorders

Public Member Functions

- wxTextAttrBorder ()

  Default constructor.
- bool operator== (const wxTextAttrBorder &border) const

  Equality operator.
- void Reset ()

  Resets the border style, colour, width and flags.
- bool EqPartial (const wxTextAttrBorder &border, bool weakTest=true) const

  Partial equality test.
- bool Apply (const wxTextAttrBorder &border, const wxTextAttrBorder *compareWith=NULL)

  Applies the border to this object, but not if the same as compareWith.
- bool RemoveStyle (const wxTextAttrBorder &attr)

  Removes the specified attributes from this object.
- void CollectCommonAttributes (const wxTextAttrBorder &attr, wxTextAttrBorder &clashingAttr, wxTextAttrBorder &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.
- void SetStyle (int style)

  Sets the border style.
- int GetStyle () const

  Gets the border style.
- void SetColour (unsigned long colour)

  Sets the border colour.
- void SetColour (const wxColour &colour)

  Sets the border colour.
- unsigned long GetColourLong () const

  Gets the colour as a long.
- wxColour GetColour () const

  Gets the colour.
- wxTextAttrDimension & GetWidth ()

  Gets the border width.
- const wxTextAttrDimension & GetWidth () const
- void SetWidth (const wxTextAttrDimension &width)

  Sets the border width.
- void SetWidth (int value, wxTextAttrUnits units=wxTEXT_ATTR_UNITS_TENTHS_MM)

  Sets the border width.
- bool HasStyle () const

  True if the border has a valid style.
- bool HasColour () const

  True if the border has a valid colour.
- bool HasWidth () const

  True if the border has a valid width.
- bool IsValid () const

  True if the border is valid.
- bool IsDefault () const

  True if the border has no attributes set.
- void MakeValid ()

  Set the valid flag for this border.
- int GetFlags () const

  Returns the border flags.

- void SetFlags (int flags)

  Sets the border flags.
- void AddFlag (int flag)

  Adds a border flag.
- void RemoveFlag (int flag)

  Removes a border flag.

Public Attributes

- int m_borderStyle
- unsigned long m_borderColour
- wxTextAttrDimension m_borderWidth
- int m_flags

21.763.2 Constructor & Destructor Documentation

wxTextAttrBorder::wxTextAttrBorder ( ) [inline]

Default constructor.

21.763.3 Member Function Documentation

void wxTextAttrBorder::AddFlag (int *flag*) [inline]

Adds a border flag.

bool wxTextAttrBorder::Apply (const wxTextAttrBorder & *border,* const wxTextAttrBorder * *compareWith* = NULL)

Applies the border to this object, but not if the same as *compareWith*.

void wxTextAttrBorder::CollectCommonAttributes (const wxTextAttrBorder & *attr,* wxTextAttrBorder & *clashingAttr,* wxTextAttrBorder & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextAttrBorder::EqPartial (const wxTextAttrBorder & *border,* bool *weakTest* = true) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *border* are present. If *weakTest* is false, the function will fail if an attribute is present in *border* but not in this object.

wxColour wxTextAttrBorder::GetColour ( ) const [inline]

Gets the colour.

unsigned long wxTextAttrBorder::GetColourLong ( ) const [inline]

Gets the colour as a long.

int wxTextAttrBorder::GetFlags ( ) const [inline]

Returns the border flags.

int wxTextAttrBorder::GetStyle ( ) const [inline]

Gets the border style.

wxTextAttrDimension& wxTextAttrBorder::GetWidth ( ) [inline]

Gets the border width.

const wxTextAttrDimension& wxTextAttrBorder::GetWidth ( ) const [inline]

bool wxTextAttrBorder::HasColour ( ) const [inline]

True if the border has a valid colour.

bool wxTextAttrBorder::HasStyle ( ) const [inline]

True if the border has a valid style.

bool wxTextAttrBorder::HasWidth ( ) const [inline]

True if the border has a valid width.

bool wxTextAttrBorder::IsDefault ( ) const [inline]

True if the border has no attributes set.

bool wxTextAttrBorder::IsValid ( ) const [inline]

True if the border is valid.

void wxTextAttrBorder::MakeValid ( ) [inline]

Set the valid flag for this border.

bool wxTextAttrBorder::operator== (const wxTextAttrBorder & *border*) const [inline]

Equality operator.

void wxTextAttrBorder::RemoveFlag (int *flag*) [inline]

Removes a border flag.

bool wxTextAttrBorder::RemoveStyle (const wxTextAttrBorder & *attr*)

Removes the specified attributes from this object.

void wxTextAttrBorder::Reset ( ) [inline]

Resets the border style, colour, width and flags.

void wxTextAttrBorder::SetColour (unsigned long *colour*) [inline]

Sets the border colour.

void wxTextAttrBorder::SetColour (const wxColour & *colour*) [inline]

Sets the border colour.

void wxTextAttrBorder::SetFlags (int *flags*) [inline]

Sets the border flags.

void wxTextAttrBorder::SetStyle (int *style*) [inline]

Sets the border style.

void wxTextAttrBorder::SetWidth (const wxTextAttrDimension & *width*) [inline]

Sets the border width.

void wxTextAttrBorder::SetWidth (int *value,* wxTextAttrUnits *units* = wxTEXT_ATTR_UNITS_TENTHS_MM) [inline]

Sets the border width.

21.763.4 Member Data Documentation

unsigned long wxTextAttrBorder::m_borderColour

int wxTextAttrBorder::m_borderStyle

wxTextAttrDimension wxTextAttrBorder::m_borderWidth

int wxTextAttrBorder::m_flags

21.764 wxTextAttrBorders Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.764.1 Detailed Description

A class representing a rich text object's borders.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl, wxRichTextAttrBorder

Public Member Functions

- wxTextAttrBorders ()

  Default constructor.
- bool operator== (const wxTextAttrBorders &borders) const

  Equality operator.
- void SetStyle (int style)

  Sets the style of all borders.
- void SetColour (unsigned long colour)

  Sets colour of all borders.
- void SetColour (const wxColour &colour)

  Sets the colour for all borders.
- void SetWidth (const wxTextAttrDimension &width)

  Sets the width of all borders.
- void SetWidth (int value, wxTextAttrUnits units=wxTEXT_ATTR_UNITS_TENTHS_MM)

  Sets the width of all borders.
- void Reset ()

  Resets all borders.
- bool EqPartial (const wxTextAttrBorders &borders, bool weakTest=true) const

  Partial equality test.
- bool Apply (const wxTextAttrBorders &borders, const wxTextAttrBorders *compareWith=NULL)

  Applies border to this object, but not if the same as compareWith.
- bool RemoveStyle (const wxTextAttrBorders &attr)

  Removes the specified attributes from this object.
- void CollectCommonAttributes (const wxTextAttrBorders &attr, wxTextAttrBorders &clashingAttr, wxTextAttrBorders &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.
- bool IsValid () const

  Returns true if at least one border is valid.
- const wxTextAttrBorder & GetLeft () const

  Returns the left border.
- wxTextAttrBorder & GetLeft ()
- const wxTextAttrBorder & GetRight () const

  Returns the right border.
- wxTextAttrBorder & GetRight ()
- const wxTextAttrBorder & GetTop () const

  Returns the top border.
- wxTextAttrBorder & GetTop ()
- const wxTextAttrBorder & GetBottom () const

  Returns the bottom border.
- wxTextAttrBorder & GetBottom ()

Public Attributes

- wxTextAttrBorder m_left
- wxTextAttrBorder m_right
- wxTextAttrBorder m_top
- wxTextAttrBorder m_bottom

21.764.2 Constructor & Destructor Documentation

wxTextAttrBorders::wxTextAttrBorders ( ) [inline]

Default constructor.

21.764.3 Member Function Documentation

bool wxTextAttrBorders::Apply (const wxTextAttrBorders & *borders,* const wxTextAttrBorders * *compareWith* = NULL)

Applies border to this object, but not if the same as *compareWith*.

void wxTextAttrBorders::CollectCommonAttributes (const wxTextAttrBorders & *attr,* wxTextAttrBorders & *clashingAttr,* wxTextAttrBorders & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextAttrBorders::EqPartial (const wxTextAttrBorders & *borders,* bool *weakTest* = true) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *borders* are present. If *weakTest* is false, the function will fail if an attribute is present in *borders* but not in this object.

const wxTextAttrBorder& wxTextAttrBorders::GetBottom ( ) const [inline]

Returns the bottom border.

wxTextAttrBorder& wxTextAttrBorders::GetBottom ( ) [inline]

const wxTextAttrBorder& wxTextAttrBorders::GetLeft ( ) const [inline]

Returns the left border.

wxTextAttrBorder& wxTextAttrBorders::GetLeft ( ) [inline]

const wxTextAttrBorder& wxTextAttrBorders::GetRight ( ) const [inline]

Returns the right border.

wxTextAttrBorder& wxTextAttrBorders::GetRight ( ) [inline]

const wxTextAttrBorder& wxTextAttrBorders::GetTop ( ) const [inline]

Returns the top border.

wxTextAttrBorder& wxTextAttrBorders::GetTop ( ) [inline]

bool wxTextAttrBorders::IsValid ( ) const [inline]

Returns true if at least one border is valid.

bool wxTextAttrBorders::operator== (const wxTextAttrBorders & *borders*) const [inline]

Equality operator.

bool wxTextAttrBorders::RemoveStyle (const wxTextAttrBorders & *attr*)

Removes the specified attributes from this object.

void wxTextAttrBorders::Reset ( ) [inline]

Resets all borders.

void wxTextAttrBorders::SetColour (unsigned long *colour*)

Sets colour of all borders.

void wxTextAttrBorders::SetColour (const wxColour & *colour*)

Sets the colour for all borders.

void wxTextAttrBorders::SetStyle (int *style*)

Sets the style of all borders.

void wxTextAttrBorders::SetWidth (const wxTextAttrDimension & *width*)

Sets the width of all borders.

void wxTextAttrBorders::SetWidth (int *value,* wxTextAttrUnits *units* = wxTEXT_ATTR_UNITS_TENTHS_MM) [inline]

Sets the width of all borders.

21.764.4 Member Data Documentation

wxTextAttrBorder wxTextAttrBorders::m_bottom

wxTextAttrBorder wxTextAttrBorders::m_left

wxTextAttrBorder wxTextAttrBorders::m_right

wxTextAttrBorder wxTextAttrBorders::m_top

21.765 wxTextAttrDimension Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.765.1 Detailed Description

A class representing a rich text dimension, including units and position.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl, wxTextAttrDimensions

Public Member Functions

- wxTextAttrDimension ()

  Default constructor.
- wxTextAttrDimension (int value, wxTextAttrUnits units=wxTEXT_ATTR_UNITS_TENTHS_MM)

  Constructor taking value and units flag.
- void Reset ()

  Resets the dimension value and flags.
- bool EqPartial (const wxTextAttrDimension &dim, bool weakTest=true) const

  Partial equality test.
- bool Apply (const wxTextAttrDimension &dim, const wxTextAttrDimension *compareWith=NULL)

  Apply the dimension, but not those identical to compareWith if present.
- void CollectCommonAttributes (const wxTextAttrDimension &attr, wxTextAttrDimension &clashingAttr, wxTextAttrDimension &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.
- bool operator== (const wxTextAttrDimension &dim) const

  Equality operator.
- int GetValue () const

  Returns the integer value of the dimension.
- float GetValueMM () const

  Returns the floating-pointing value of the dimension in mm.
- void SetValueMM (float value)

  Sets the value of the dimension in mm.
- void SetValue (int value)

  Sets the integer value of the dimension.
- void SetValue (int value, wxTextAttrDimensionFlags flags)

  Sets the integer value of the dimension, passing dimension flags.

- void SetValue (int value, wxTextAttrUnits units)

  Sets the integer value and units.
- void SetValue (const wxTextAttrDimension &dim)

  Sets the dimension.
- wxTextAttrUnits GetUnits () const

  Gets the units of the dimension.
- void SetUnits (wxTextAttrUnits units)

  Sets the units of the dimension.
- wxTextBoxAttrPosition GetPosition () const

  Gets the position flags.
- void SetPosition (wxTextBoxAttrPosition pos)

  Sets the position flags.
- bool IsValid () const

  Returns true if the dimension is valid.
- void SetValid (bool b)

  Sets the valid flag.
- wxTextAttrDimensionFlags GetFlags () const

  Gets the dimension flags.
- void SetFlags (wxTextAttrDimensionFlags flags)

  Sets the dimension flags.

Public Attributes

- int m_value
- wxTextAttrDimensionFlags m_flags

21.765.2 Constructor & Destructor Documentation

wxTextAttrDimension::wxTextAttrDimension ( ) [inline]

Default constructor.

wxTextAttrDimension::wxTextAttrDimension (int *value,* wxTextAttrUnits *units* = wxTEXT_ATTR_UNITS_TENTHS_MM) [inline]

Constructor taking value and units flag.

21.765.3 Member Function Documentation

bool wxTextAttrDimension::Apply (const wxTextAttrDimension & *dim,* const wxTextAttrDimension * *compareWith* = NULL)

Apply the dimension, but not those identical to *compareWith* if present.

void wxTextAttrDimension::CollectCommonAttributes (const wxTextAttrDimension & *attr,* wxTextAttrDimension & *clashingAttr,* wxTextAttrDimension & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextAttrDimension::EqPartial (const wxTextAttrDimension & *dim,* bool *weakTest* = `true`) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *dim* are present. If *weakTest* is false, the function will fail if an attribute is present in *dim* but not in this object.

wxTextAttrDimensionFlags wxTextAttrDimension::GetFlags ( ) const `[inline]`

Gets the dimension flags.

wxTextBoxAttrPosition wxTextAttrDimension::GetPosition ( ) const `[inline]`

Gets the position flags.

wxTextAttrUnits wxTextAttrDimension::GetUnits ( ) const `[inline]`

Gets the units of the dimension.

int wxTextAttrDimension::GetValue ( ) const `[inline]`

Returns the integer value of the dimension.

float wxTextAttrDimension::GetValueMM ( ) const `[inline]`

Returns the floating-pointing value of the dimension in mm.

bool wxTextAttrDimension::IsValid ( ) const `[inline]`

Returns true if the dimension is valid.

bool wxTextAttrDimension::operator== (const wxTextAttrDimension & *dim*) const `[inline]`

Equality operator.

void wxTextAttrDimension::Reset ( ) `[inline]`

Resets the dimension value and flags.

void wxTextAttrDimension::SetFlags (wxTextAttrDimensionFlags *flags*) `[inline]`

Sets the dimension flags.

void wxTextAttrDimension::SetPosition (wxTextBoxAttrPosition *pos*) `[inline]`

Sets the position flags.

void wxTextAttrDimension::SetUnits (wxTextAttrUnits *units*) `[inline]`

Sets the units of the dimension.

void wxTextAttrDimension::SetValid (bool *b*) `[inline]`

Sets the valid flag.

void wxTextAttrDimension::SetValue (int *value*) `[inline]`

Sets the integer value of the dimension.

void wxTextAttrDimension::SetValue (int *value,* wxTextAttrDimensionFlags *flags*) `[inline]`

Sets the integer value of the dimension, passing dimension flags.

void wxTextAttrDimension::SetValue (int *value,* wxTextAttrUnits *units*) `[inline]`

Sets the integer value and units.

void wxTextAttrDimension::SetValue (const wxTextAttrDimension & *dim*) `[inline]`

Sets the dimension.

void wxTextAttrDimension::SetValueMM (float *value*) `[inline]`

Sets the value of the dimension in mm.

21.765.4 Member Data Documentation

wxTextAttrDimensionFlags wxTextAttrDimension::m_flags

int wxTextAttrDimension::m_value

21.766 wxTextAttrDimensionConverter Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.766.1 Detailed Description

A class to make it easier to convert dimensions.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl, wxTextAttrDimension

Public Member Functions

- wxTextAttrDimensionConverter (wxDC &dc, double scale=1.0, const wxSize &parentSize=wxDefaultSize)

  Constructor.
- wxTextAttrDimensionConverter (int ppi, double scale=1.0, const wxSize &parentSize=wxDefaultSize)

  Constructor.
- int GetPixels (const wxTextAttrDimension &dim, int direction=wxHORIZONTAL) const

  Gets the pixel size for the given dimension.
- int GetTenthsMM (const wxTextAttrDimension &dim) const

  Gets the mm size for the given dimension.
- int ConvertTenthsMMToPixels (int units) const

  Converts tenths of a mm to pixels.
- int ConvertPixelsToTenthsMM (int pixels) const

  Converts pixels to tenths of a mm.

Public Attributes

- int m_ppi
- double m_scale
- wxSize m_parentSize

21.766.2 Constructor & Destructor Documentation

wxTextAttrDimensionConverter::wxTextAttrDimensionConverter (wxDC & *dc,* double *scale* = `1.0`, const wxSize & *parentSize* = wxDefaultSize)

Constructor.

wxTextAttrDimensionConverter::wxTextAttrDimensionConverter (int *ppi,* double *scale* = `1.0`, const wxSize & *parentSize* = wxDefaultSize)

Constructor.

21.766.3 Member Function Documentation

int wxTextAttrDimensionConverter::ConvertPixelsToTenthsMM (int *pixels*) const

Converts pixels to tenths of a mm.

int wxTextAttrDimensionConverter::ConvertTenthsMMToPixels (int *units*) const

Converts tenths of a mm to pixels.

int wxTextAttrDimensionConverter::GetPixels (const wxTextAttrDimension & *dim,* int *direction* = wxHORIZONTAL) const

Gets the pixel size for the given dimension.

int wxTextAttrDimensionConverter::GetTenthsMM (const wxTextAttrDimension & *dim*) const

Gets the mm size for the given dimension.

21.766.4 Member Data Documentation

wxSize wxTextAttrDimensionConverter::m_parentSize

int wxTextAttrDimensionConverter::m_ppi

double wxTextAttrDimensionConverter::m_scale

21.767 wxTextAttrDimensions Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.767.1 Detailed Description

A class for left, right, top and bottom dimensions.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl, wxTextAttrDimension

Public Member Functions

- wxTextAttrDimensions ()

  Default constructor.
- void Reset ()

  Resets the value and flags for all dimensions.
- bool operator== (const wxTextAttrDimensions &dims) const

  Equality operator.
- bool EqPartial (const wxTextAttrDimensions &dims, bool weakTest=true) const

  Partial equality test.
- bool Apply (const wxTextAttrDimensions &dims, const wxTextAttrDimensions *compareWith=NULL)

  Apply to 'this', but not if the same as compareWith.
- void CollectCommonAttributes (const wxTextAttrDimensions &attr, wxTextAttrDimensions &clashingAttr, wxTextAttrDimensions &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.
- bool RemoveStyle (const wxTextAttrDimensions &attr)

  Remove specified attributes from this object.
- const wxTextAttrDimension & GetLeft () const

  Gets the left dimension.
- wxTextAttrDimension & GetLeft ()
- const wxTextAttrDimension & GetRight () const

  Gets the right dimension.
- wxTextAttrDimension & GetRight ()
- const wxTextAttrDimension & GetTop () const

  Gets the top dimension.

- wxTextAttrDimension & GetTop ()
- const wxTextAttrDimension & GetBottom () const

  Gets the bottom dimension.
- wxTextAttrDimension & GetBottom ()
- bool IsValid () const

  Are all dimensions valid?

Public Attributes

- wxTextAttrDimension m_left
- wxTextAttrDimension m_top
- wxTextAttrDimension m_right
- wxTextAttrDimension m_bottom

21.767.2 Constructor & Destructor Documentation

wxTextAttrDimensions::wxTextAttrDimensions ( ) [inline]

Default constructor.

21.767.3 Member Function Documentation

bool wxTextAttrDimensions::Apply (const wxTextAttrDimensions & *dims,* const wxTextAttrDimensions * *compareWith* = NULL)

Apply to 'this', but not if the same as *compareWith*.

void wxTextAttrDimensions::CollectCommonAttributes (const wxTextAttrDimensions & *attr,* wxTextAttrDimensions & *clashingAttr,* wxTextAttrDimensions & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextAttrDimensions::EqPartial (const wxTextAttrDimensions & *dims,* bool *weakTest* = true) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *dim* sare present. If *weakTest* is false, the function will fail if an attribute is present in *dims* but not in this object.

const wxTextAttrDimension& wxTextAttrDimensions::GetBottom ( ) const [inline]

Gets the bottom dimension.

wxTextAttrDimension& wxTextAttrDimensions::GetBottom ( ) [inline]

const wxTextAttrDimension& wxTextAttrDimensions::GetLeft ( ) const [inline]

Gets the left dimension.

wxTextAttrDimension& wxTextAttrDimensions::GetLeft ( ) [inline]

const wxTextAttrDimension& wxTextAttrDimensions::GetRight ( ) const [inline]

Gets the right dimension.

wxTextAttrDimension& wxTextAttrDimensions::GetRight ( ) [inline]

const wxTextAttrDimension& wxTextAttrDimensions::GetTop ( ) const [inline]

Gets the top dimension.

wxTextAttrDimension& wxTextAttrDimensions::GetTop ( ) [inline]

bool wxTextAttrDimensions::IsValid ( ) const [inline]

Are all dimensions valid?

bool wxTextAttrDimensions::operator== (const wxTextAttrDimensions & *dims*) const [inline]

Equality operator.

bool wxTextAttrDimensions::RemoveStyle (const wxTextAttrDimensions & *attr*)

Remove specified attributes from this object.

void wxTextAttrDimensions::Reset ( ) [inline]

Resets the value and flags for all dimensions.

21.767.4 Member Data Documentation

wxTextAttrDimension wxTextAttrDimensions::m_bottom

wxTextAttrDimension wxTextAttrDimensions::m_left

wxTextAttrDimension wxTextAttrDimensions::m_right

wxTextAttrDimension wxTextAttrDimensions::m_top

21.768 wxTextAttrShadow Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.768.1 Detailed Description

A class representing a shadow.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl

Public Member Functions

- wxTextAttrShadow ()

  Default constructor.
- bool operator== (const wxTextAttrShadow &shadow) const

  Equality operator.
- void Reset ()

  Resets the shadow.
- bool EqPartial (const wxTextAttrShadow &shadow, bool weakTest=true) const

  Partial equality test.
- bool Apply (const wxTextAttrShadow &shadow, const wxTextAttrShadow *compareWith=NULL)

  Applies the border to this object, but not if the same as compareWith.
- bool RemoveStyle (const wxTextAttrShadow &attr)

  Removes the specified attributes from this object.
- void CollectCommonAttributes (const wxTextAttrShadow &attr, wxTextAttrShadow &clashingAttr, wxTextAttrShadow &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.
- void SetColour (unsigned long colour)

  Sets the shadow colour.
- void SetColour (const wxColour &colour)

  Sets the shadow colour.
- unsigned long GetColourLong () const

  Gets the colour as a long.
- wxColour GetColour () const

  Gets the colour.
- bool HasColour () const

  True if the shadow has a valid colour.
- wxTextAttrDimension & GetOffsetX ()

  Gets the shadow horizontal offset.
- const wxTextAttrDimension & GetOffsetX () const
- void SetOffsetX (const wxTextAttrDimension &offset)

  Sets the shadow horizontal offset.
- wxTextAttrDimension & GetOffsetY ()

  Gets the shadow vertical offset.
- const wxTextAttrDimension & GetOffsetY () const
- void SetOffsetY (const wxTextAttrDimension &offset)

  Sets the shadow vertical offset.
- wxTextAttrDimension & GetSpread ()

  Gets the shadow spread size.
- const wxTextAttrDimension & GetSpread () const
- void SetSpread (const wxTextAttrDimension &spread)

  Sets the shadow spread size.

- wxTextAttrDimension & GetBlurDistance ()

  Gets the shadow blur distance.
- const wxTextAttrDimension & GetBlurDistance () const
- void SetBlurDistance (const wxTextAttrDimension &blur)

  Sets the shadow blur distance.
- wxTextAttrDimension & GetOpacity ()

  Gets the shadow opacity.
- const wxTextAttrDimension & GetOpacity () const
- bool IsValid () const

  Returns true if the dimension is valid.
- void SetValid (bool b)

  Sets the valid flag.
- int GetFlags () const

  Returns the border flags.
- void SetFlags (int flags)

  Sets the border flags.
- void AddFlag (int flag)

  Adds a border flag.
- void RemoveFlag (int flag)

  Removes a border flag.
- void SetOpacity (const wxTextAttrDimension &opacity)

  Sets the shadow opacity.
- bool IsDefault () const

  True if the shadow has no attributes set.

Public Attributes

- int m_flags
- unsigned long m_shadowColour
- wxTextAttrDimension m_offsetX
- wxTextAttrDimension m_offsetY
- wxTextAttrDimension m_spread
- wxTextAttrDimension m_blurDistance
- wxTextAttrDimension m_opacity

21.768.2 Constructor & Destructor Documentation

wxTextAttrShadow::wxTextAttrShadow ( ) [inline]

Default constructor.

21.768.3 Member Function Documentation

void wxTextAttrShadow::AddFlag (int *flag*) [inline]

Adds a border flag.

bool wxTextAttrShadow::Apply (const wxTextAttrShadow & *shadow*, const wxTextAttrShadow * *compareWith* = NULL)

Applies the border to this object, but not if the same as *compareWith*.

void wxTextAttrShadow::CollectCommonAttributes (const wxTextAttrShadow & *attr,* wxTextAttrShadow & *clashingAttr,* wxTextAttrShadow & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextAttrShadow::EqPartial (const wxTextAttrShadow & *shadow,* bool *weakTest* = `true`) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *border* are present. If *weakTest* is false, the function will fail if an attribute is present in *border* but not in this object.

wxTextAttrDimension& wxTextAttrShadow::GetBlurDistance ( ) `[inline]`

Gets the shadow blur distance.

const wxTextAttrDimension& wxTextAttrShadow::GetBlurDistance ( ) const `[inline]`

wxColour wxTextAttrShadow::GetColour ( ) const `[inline]`

Gets the colour.

unsigned long wxTextAttrShadow::GetColourLong ( ) const `[inline]`

Gets the colour as a long.

int wxTextAttrShadow::GetFlags ( ) const `[inline]`

Returns the border flags.

wxTextAttrDimension& wxTextAttrShadow::GetOffsetX ( ) `[inline]`

Gets the shadow horizontal offset.

const wxTextAttrDimension& wxTextAttrShadow::GetOffsetX ( ) const `[inline]`

wxTextAttrDimension& wxTextAttrShadow::GetOffsetY ( ) `[inline]`

Gets the shadow vertical offset.

const wxTextAttrDimension& wxTextAttrShadow::GetOffsetY ( ) const `[inline]`

wxTextAttrDimension& wxTextAttrShadow::GetOpacity ( ) `[inline]`

Gets the shadow opacity.

const wxTextAttrDimension& wxTextAttrShadow::GetOpacity ( ) const `[inline]`

wxTextAttrDimension& wxTextAttrShadow::GetSpread ( ) `[inline]`

Gets the shadow spread size.

const wxTextAttrDimension& wxTextAttrShadow::GetSpread ( ) const `[inline]`

bool wxTextAttrShadow::HasColour ( ) const `[inline]`

True if the shadow has a valid colour.

bool wxTextAttrShadow::IsDefault ( ) const `[inline]`

True if the shadow has no attributes set.

bool wxTextAttrShadow::IsValid ( ) const `[inline]`

Returns true if the dimension is valid.

bool wxTextAttrShadow::operator== (const wxTextAttrShadow & *shadow*) const

Equality operator.

void wxTextAttrShadow::RemoveFlag (int *flag*) `[inline]`

Removes a border flag.

bool wxTextAttrShadow::RemoveStyle (const wxTextAttrShadow & *attr*)

Removes the specified attributes from this object.

void wxTextAttrShadow::Reset ( )

Resets the shadow.

void wxTextAttrShadow::SetBlurDistance (const wxTextAttrDimension & *blur*) `[inline]`

Sets the shadow blur distance.

void wxTextAttrShadow::SetColour (unsigned long *colour*) `[inline]`

Sets the shadow colour.

void wxTextAttrShadow::SetColour (const wxColour & *colour*) `[inline]`

Sets the shadow colour.

void wxTextAttrShadow::SetFlags (int *flags*) `[inline]`

Sets the border flags.

void wxTextAttrShadow::SetOffsetX (const wxTextAttrDimension & *offset*) `[inline]`

Sets the shadow horizontal offset.

void wxTextAttrShadow::SetOffsetY (const wxTextAttrDimension & *offset*) `[inline]`

Sets the shadow vertical offset.

void wxTextAttrShadow::SetOpacity (const wxTextAttrDimension & *opacity*) `[inline]`

Sets the shadow opacity.

void wxTextAttrShadow::SetSpread (const wxTextAttrDimension & *spread*) `[inline]`

Sets the shadow spread size.

void wxTextAttrShadow::SetValid (bool *b*) `[inline]`

Sets the valid flag.

21.768.4 Member Data Documentation

wxTextAttrDimension wxTextAttrShadow::m_blurDistance

int wxTextAttrShadow::m_flags

wxTextAttrDimension wxTextAttrShadow::m_offsetX

wxTextAttrDimension wxTextAttrShadow::m_offsetY

wxTextAttrDimension wxTextAttrShadow::m_opacity

unsigned long wxTextAttrShadow::m_shadowColour

wxTextAttrDimension wxTextAttrShadow::m_spread

21.769 wxTextAttrSize Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.769.1 Detailed Description

A class for representing width and height.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl, wxTextAttrDimension

Public Member Functions

- wxTextAttrSize ()

  Default constructor.
- void Reset ()

  Resets the width and height dimensions.
- bool operator== (const wxTextAttrSize &size) const

  Equality operator.
- bool EqPartial (const wxTextAttrSize &size, bool weakTest=true) const

  Partial equality test.
- bool Apply (const wxTextAttrSize &dims, const wxTextAttrSize *compareWith=NULL)

  Apply to this object, but not if the same as compareWith.
- void CollectCommonAttributes (const wxTextAttrSize &attr, wxTextAttrSize &clashingAttr, wxTextAttrSize &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.
- bool RemoveStyle (const wxTextAttrSize &attr)

  Removes the specified attributes from this object.
- wxTextAttrDimension & GetWidth ()

  Returns the width.
- const wxTextAttrDimension & GetWidth () const
- void SetWidth (int value, wxTextAttrDimensionFlags flags)

  Sets the width.
- void SetWidth (int value, wxTextAttrUnits units)

  Sets the width.
- void SetWidth (const wxTextAttrDimension &dim)

  Sets the width.
- wxTextAttrDimension & GetHeight ()

  Gets the height.
- const wxTextAttrDimension & GetHeight () const
- void SetHeight (int value, wxTextAttrDimensionFlags flags)

  Sets the height.
- void SetHeight (int value, wxTextAttrUnits units)

  Sets the height.
- void SetHeight (const wxTextAttrDimension &dim)

  Sets the height.
- bool IsValid () const

  Is the size valid?

Public Attributes

- wxTextAttrDimension m_width
- wxTextAttrDimension m_height

21.769.2 Constructor & Destructor Documentation

wxTextAttrSize::wxTextAttrSize ( ) [inline]

Default constructor.

21.769.3 Member Function Documentation

bool wxTextAttrSize::Apply (const wxTextAttrSize & *dims,* const wxTextAttrSize * *compareWith* = NULL)

Apply to this object, but not if the same as *compareWith*.

void wxTextAttrSize::CollectCommonAttributes (const wxTextAttrSize & *attr,* wxTextAttrSize & *clashingAttr,* wxTextAttrSize & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextAttrSize::EqPartial (const wxTextAttrSize & *size,* bool *weakTest* = true) const

Partial equality test.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *size* are present. If *weakTest* is false, the function will fail if an attribute is present in *size* but not in this object.

wxTextAttrDimension& wxTextAttrSize::GetHeight ( ) [inline]

Gets the height.

const wxTextAttrDimension& wxTextAttrSize::GetHeight ( ) const [inline]

wxTextAttrDimension& wxTextAttrSize::GetWidth ( ) [inline]

Returns the width.

const wxTextAttrDimension& wxTextAttrSize::GetWidth ( ) const [inline]

bool wxTextAttrSize::IsValid ( ) const [inline]

Is the size valid?

bool wxTextAttrSize::operator== (const wxTextAttrSize & *size*) const [inline]

Equality operator.

bool wxTextAttrSize::RemoveStyle (const wxTextAttrSize & *attr*)

Removes the specified attributes from this object.

void wxTextAttrSize::Reset ( ) [inline]

Resets the width and height dimensions.

void wxTextAttrSize::SetHeight (int *value,* wxTextAttrDimensionFlags *flags*) [inline]

Sets the height.

void wxTextAttrSize::SetHeight (int *value,* wxTextAttrUnits *units*) `[inline]`

Sets the height.

void wxTextAttrSize::SetHeight (const wxTextAttrDimension & *dim*) `[inline]`

Sets the height.

void wxTextAttrSize::SetWidth (int *value,* wxTextAttrDimensionFlags *flags*) `[inline]`

Sets the width.

void wxTextAttrSize::SetWidth (int *value,* wxTextAttrUnits *units*) `[inline]`

Sets the width.

void wxTextAttrSize::SetWidth (const wxTextAttrDimension & *dim*) `[inline]`

Sets the width.

21.769.4 Member Data Documentation

wxTextAttrDimension wxTextAttrSize::m_height

wxTextAttrDimension wxTextAttrSize::m_width

21.770 wxTextBoxAttr Class Reference

`#include <wx/richtext/richtextbuffer.h>`

21.770.1 Detailed Description

A class representing the box attributes of a rich text object.

Library: wxRichText

Category: Rich Text

See also

wxRichTextAttr, wxRichTextCtrl

Public Member Functions

- wxTextBoxAttr ()

  Default constructor.
- wxTextBoxAttr (const wxTextBoxAttr &attr)

  Copy constructor.
- void Init ()

Initialises this object.

- void Reset ()

  Resets this object.

- bool operator== (const wxTextBoxAttr &attr) const

  Equality test.

- bool EqPartial (const wxTextBoxAttr &attr, bool weakTest=true) const

  Partial equality test, ignoring unset attributes.

- bool Apply (const wxTextBoxAttr &style, const wxTextBoxAttr *compareWith=NULL)

  Merges the given attributes.

- void CollectCommonAttributes (const wxTextBoxAttr &attr, wxTextBoxAttr &clashingAttr, wxTextBoxAttr &absentAttr)

  Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

- bool RemoveStyle (const wxTextBoxAttr &attr)

  Removes the specified attributes from this object.

- void SetFlags (int flags)

  Sets the flags.

- int GetFlags () const

  Returns the flags.

- bool HasFlag (wxTextBoxAttrFlags flag) const

  Is this flag present?

- void RemoveFlag (wxTextBoxAttrFlags flag)

  Removes this flag.

- void AddFlag (wxTextBoxAttrFlags flag)

  Adds this flag.

- bool IsDefault () const

  Returns true if no attributes are set.

- wxTextBoxAttrFloatStyle GetFloatMode () const

  Returns the float mode.

- void SetFloatMode (wxTextBoxAttrFloatStyle mode)

  Sets the float mode.

- bool HasFloatMode () const

  Returns true if float mode is active.

- bool IsFloating () const

  Returns true if this object is floating.

- wxTextBoxAttrClearStyle GetClearMode () const

  Returns the clear mode - whether to wrap text after object.

- void SetClearMode (wxTextBoxAttrClearStyle mode)

  Set the clear mode.

- bool HasClearMode () const

  Returns true if we have a clear flag.

- wxTextBoxAttrCollapseMode GetCollapseBorders () const

  Returns the collapse mode - whether to collapse borders.

- void SetCollapseBorders (wxTextBoxAttrCollapseMode collapse)

  Sets the collapse mode - whether to collapse borders.

- bool HasCollapseBorders () const

  Returns true if the collapse borders flag is present.

- wxTextBoxAttrWhitespaceMode GetWhitespaceMode () const

  Returns the whitespace mode.

- void SetWhitespaceMode (wxTextBoxAttrWhitespaceMode whitespace)

  Sets the whitespace mode.

- bool HasWhitespaceMode () const

  Returns true if the whitespace flag is present.
- bool HasCornerRadius () const

  Returns true if the corner radius flag is present.
- const wxTextAttrDimension & GetCornerRadius () const

  Returns the corner radius value.
- wxTextAttrDimension & GetCornerRadius ()
- void SetCornerRadius (const wxTextAttrDimension &dim)

  Sets the corner radius value.
- wxTextBoxAttrVerticalAlignment GetVerticalAlignment () const

  Returns the vertical alignment.
- void SetVerticalAlignment (wxTextBoxAttrVerticalAlignment verticalAlignment)

  Sets the vertical alignment.
- bool HasVerticalAlignment () const

  Returns true if a vertical alignment flag is present.
- wxTextAttrDimensions & GetMargins ()

  Returns the margin values.
- const wxTextAttrDimensions & GetMargins () const
- wxTextAttrDimension & GetLeftMargin ()

  Returns the left margin.
- const wxTextAttrDimension & GetLeftMargin () const
- wxTextAttrDimension & GetRightMargin ()

  Returns the right margin.
- const wxTextAttrDimension & GetRightMargin () const
- wxTextAttrDimension & GetTopMargin ()

  Returns the top margin.
- const wxTextAttrDimension & GetTopMargin () const
- wxTextAttrDimension & GetBottomMargin ()

  Returns the bottom margin.
- const wxTextAttrDimension & GetBottomMargin () const
- wxTextAttrDimensions & GetPosition ()

  Returns the position.
- const wxTextAttrDimensions & GetPosition () const
- wxTextAttrDimension & GetLeft ()

  Returns the left position.
- const wxTextAttrDimension & GetLeft () const
- wxTextAttrDimension & GetRight ()

  Returns the right position.
- const wxTextAttrDimension & GetRight () const
- wxTextAttrDimension & GetTop ()

  Returns the top position.
- const wxTextAttrDimension & GetTop () const
- wxTextAttrDimension & GetBottom ()

  Returns the bottom position.
- const wxTextAttrDimension & GetBottom () const
- wxTextAttrDimensions & GetPadding ()

  Returns the padding values.
- const wxTextAttrDimensions & GetPadding () const
- wxTextAttrDimension & GetLeftPadding ()

  Returns the left padding value.
- const wxTextAttrDimension & GetLeftPadding () const
- wxTextAttrDimension & GetRightPadding ()

Returns the right padding value.

- const wxTextAttrDimension & GetRightPadding () const
- wxTextAttrDimension & GetTopPadding ()

Returns the top padding value.

- const wxTextAttrDimension & GetTopPadding () const
- wxTextAttrDimension & GetBottomPadding ()

Returns the bottom padding value.

- const wxTextAttrDimension & GetBottomPadding () const
- wxTextAttrBorders & GetBorder ()

Returns the borders.

- const wxTextAttrBorders & GetBorder () const
- wxTextAttrBorder & GetLeftBorder ()

Returns the left border.

- const wxTextAttrBorder & GetLeftBorder () const
- wxTextAttrBorder & GetTopBorder ()

Returns the top border.

- const wxTextAttrBorder & GetTopBorder () const
- wxTextAttrBorder & GetRightBorder ()

Returns the right border.

- const wxTextAttrBorder & GetRightBorder () const
- wxTextAttrBorder & GetBottomBorder ()

Returns the bottom border.

- const wxTextAttrBorder & GetBottomBorder () const
- wxTextAttrBorders & GetOutline ()

Returns the outline.

- const wxTextAttrBorders & GetOutline () const
- wxTextAttrBorder & GetLeftOutline ()

Returns the left outline.

- const wxTextAttrBorder & GetLeftOutline () const
- wxTextAttrBorder & GetTopOutline ()

Returns the top outline.

- const wxTextAttrBorder & GetTopOutline () const
- wxTextAttrBorder & GetRightOutline ()

Returns the right outline.

- const wxTextAttrBorder & GetRightOutline () const
- wxTextAttrBorder & GetBottomOutline ()

Returns the bottom outline.

- const wxTextAttrBorder & GetBottomOutline () const
- wxTextAttrSize & GetSize ()

Returns the object size.

- const wxTextAttrSize & GetSize () const
- wxTextAttrSize & GetMinSize ()

Returns the object minimum size.

- const wxTextAttrSize & GetMinSize () const
- wxTextAttrSize & GetMaxSize ()

Returns the object maximum size.

- const wxTextAttrSize & GetMaxSize () const
- void SetSize (const wxTextAttrSize &sz)

Sets the object size.

- void SetMinSize (const wxTextAttrSize &sz)

Sets the object minimum size.

- void SetMaxSize (const wxTextAttrSize &sz)

Sets the object maximum size.

- wxTextAttrDimension & GetWidth ()

  Returns the object width.
- const wxTextAttrDimension & GetWidth () const
- wxTextAttrDimension & GetHeight ()

  Returns the object height.
- const wxTextAttrDimension & GetHeight () const
- const wxString & GetBoxStyleName () const

  Returns the box style name.
- void SetBoxStyleName (const wxString &name)

  Sets the box style name.
- bool HasBoxStyleName () const

  Returns true if the box style name is present.
- wxTextAttrShadow & GetShadow ()

  Returns the box shadow attributes.
- const wxTextAttrShadow & GetShadow () const

Public Attributes

- int m_flags
- wxTextAttrDimensions m_margins
- wxTextAttrDimensions m_padding
- wxTextAttrDimensions m_position
- wxTextAttrSize m_size
- wxTextAttrSize m_minSize
- wxTextAttrSize m_maxSize
- wxTextAttrBorders m_border
- wxTextAttrBorders m_outline
- wxTextBoxAttrFloatStyle m_floatMode
- wxTextBoxAttrClearStyle m_clearMode
- wxTextBoxAttrCollapseMode m_collapseMode
- wxTextBoxAttrVerticalAlignment m_verticalAlignment
- wxTextBoxAttrWhitespaceMode m_whitespaceMode
- wxTextAttrDimension m_cornerRadius
- wxString m_boxStyleName
- wxTextAttrShadow m_shadow

21.770.2 Constructor & Destructor Documentation

wxTextBoxAttr::wxTextBoxAttr ( ) [inline]

Default constructor.

wxTextBoxAttr::wxTextBoxAttr (const wxTextBoxAttr & *attr*) [inline]

Copy constructor.

21.770.3 Member Function Documentation

void wxTextBoxAttr::AddFlag (wxTextBoxAttrFlags *flag*) [inline]

Adds this flag.

bool wxTextBoxAttr::Apply (const wxTextBoxAttr & *style,* const wxTextBoxAttr * *compareWith* = NULL)

Merges the given attributes.

If *compareWith* is non-NULL, then it will be used to mask out those attributes that are the same in style and *compareWith*, for situations where we don't want to explicitly set inherited attributes.

void wxTextBoxAttr::CollectCommonAttributes (const wxTextBoxAttr & *attr,* wxTextBoxAttr & *clashingAttr,* wxTextBoxAttr & *absentAttr*)

Collects the attributes that are common to a range of content, building up a note of which attributes are absent in some objects and which clash in some objects.

bool wxTextBoxAttr::EqPartial (const wxTextBoxAttr & *attr,* bool *weakTest* = true) const

Partial equality test, ignoring unset attributes.

If *weakTest* is true, attributes of this object do not have to be present if those attributes of *attr* are present. If *weakTest* is false, the function will fail if an attribute is present in *attr* but not in this object.

wxTextAttrBorders& wxTextBoxAttr::GetBorder ( ) [inline]

Returns the borders.

const wxTextAttrBorders& wxTextBoxAttr::GetBorder ( ) const [inline]

wxTextAttrDimension& wxTextBoxAttr::GetBottom ( ) [inline]

Returns the bottom position.

const wxTextAttrDimension& wxTextBoxAttr::GetBottom ( ) const [inline]

wxTextAttrBorder& wxTextBoxAttr::GetBottomBorder ( ) [inline]

Returns the bottom border.

const wxTextAttrBorder& wxTextBoxAttr::GetBottomBorder ( ) const [inline]

wxTextAttrDimension& wxTextBoxAttr::GetBottomMargin ( ) [inline]

Returns the bottom margin.

const wxTextAttrDimension& wxTextBoxAttr::GetBottomMargin ( ) const [inline]

wxTextAttrBorder& wxTextBoxAttr::GetBottomOutline ( ) [inline]

Returns the bottom outline.

const wxTextAttrBorder& wxTextBoxAttr::GetBottomOutline ( ) const [inline]

wxTextAttrDimension& wxTextBoxAttr::GetBottomPadding ( ) [inline]

Returns the bottom padding value.

const wxTextAttrDimension& wxTextBoxAttr::GetBottomPadding ( ) const `[inline]`

const wxString& wxTextBoxAttr::GetBoxStyleName ( ) const `[inline]`

Returns the box style name.

wxTextBoxAttrClearStyle wxTextBoxAttr::GetClearMode ( ) const `[inline]`

Returns the clear mode - whether to wrap text after object.
Currently unimplemented.

wxTextBoxAttrCollapseMode wxTextBoxAttr::GetCollapseBorders ( ) const `[inline]`

Returns the collapse mode - whether to collapse borders.

const wxTextAttrDimension& wxTextBoxAttr::GetCornerRadius ( ) const `[inline]`

Returns the corner radius value.

wxTextAttrDimension& wxTextBoxAttr::GetCornerRadius ( ) `[inline]`

int wxTextBoxAttr::GetFlags ( ) const `[inline]`

Returns the flags.

wxTextBoxAttrFloatStyle wxTextBoxAttr::GetFloatMode ( ) const `[inline]`

Returns the float mode.

wxTextAttrDimension& wxTextBoxAttr::GetHeight ( ) `[inline]`

Returns the object height.

const wxTextAttrDimension& wxTextBoxAttr::GetHeight ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetLeft ( ) `[inline]`

Returns the left position.

const wxTextAttrDimension& wxTextBoxAttr::GetLeft ( ) const `[inline]`

wxTextAttrBorder& wxTextBoxAttr::GetLeftBorder ( ) `[inline]`

Returns the left border.

const wxTextAttrBorder& wxTextBoxAttr::GetLeftBorder ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetLeftMargin ( ) `[inline]`

Returns the left margin.

const wxTextAttrDimension& wxTextBoxAttr::GetLeftMargin ( ) const `[inline]`

wxTextAttrBorder& wxTextBoxAttr::GetLeftOutline ( ) `[inline]`

Returns the left outline.

const wxTextAttrBorder& wxTextBoxAttr::GetLeftOutline ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetLeftPadding ( ) `[inline]`

Returns the left padding value.

const wxTextAttrDimension& wxTextBoxAttr::GetLeftPadding ( ) const `[inline]`

wxTextAttrDimensions& wxTextBoxAttr::GetMargins ( ) `[inline]`

Returns the margin values.

const wxTextAttrDimensions& wxTextBoxAttr::GetMargins ( ) const `[inline]`

wxTextAttrSize& wxTextBoxAttr::GetMaxSize ( ) `[inline]`

Returns the object maximum size.

const wxTextAttrSize& wxTextBoxAttr::GetMaxSize ( ) const `[inline]`

wxTextAttrSize& wxTextBoxAttr::GetMinSize ( ) `[inline]`

Returns the object minimum size.

const wxTextAttrSize& wxTextBoxAttr::GetMinSize ( ) const `[inline]`

wxTextAttrBorders& wxTextBoxAttr::GetOutline ( ) `[inline]`

Returns the outline.

const wxTextAttrBorders& wxTextBoxAttr::GetOutline ( ) const `[inline]`

wxTextAttrDimensions& wxTextBoxAttr::GetPadding ( ) `[inline]`

Returns the padding values.

const wxTextAttrDimensions& wxTextBoxAttr::GetPadding ( ) const `[inline]`

wxTextAttrDimensions& wxTextBoxAttr::GetPosition ( ) `[inline]`

Returns the position.

const wxTextAttrDimensions& wxTextBoxAttr::GetPosition ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetRight ( ) `[inline]`

Returns the right position.

const wxTextAttrDimension& wxTextBoxAttr::GetRight ( ) const `[inline]`

wxTextAttrBorder& wxTextBoxAttr::GetRightBorder ( ) `[inline]`

Returns the right border.

const wxTextAttrBorder& wxTextBoxAttr::GetRightBorder ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetRightMargin ( ) `[inline]`

Returns the right margin.

const wxTextAttrDimension& wxTextBoxAttr::GetRightMargin ( ) const `[inline]`

wxTextAttrBorder& wxTextBoxAttr::GetRightOutline ( ) `[inline]`

Returns the right outline.

const wxTextAttrBorder& wxTextBoxAttr::GetRightOutline ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetRightPadding ( ) `[inline]`

Returns the right padding value.

const wxTextAttrDimension& wxTextBoxAttr::GetRightPadding ( ) const `[inline]`

wxTextAttrShadow& wxTextBoxAttr::GetShadow ( ) `[inline]`

Returns the box shadow attributes.

const wxTextAttrShadow& wxTextBoxAttr::GetShadow ( ) const `[inline]`

wxTextAttrSize& wxTextBoxAttr::GetSize ( ) `[inline]`

Returns the object size.

const wxTextAttrSize& wxTextBoxAttr::GetSize ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetTop ( ) `[inline]`

Returns the top position.

const wxTextAttrDimension& wxTextBoxAttr::GetTop ( ) const `[inline]`

wxTextAttrBorder& wxTextBoxAttr::GetTopBorder ( ) `[inline]`

Returns the top border.

const wxTextAttrBorder& wxTextBoxAttr::GetTopBorder ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetTopMargin ( ) `[inline]`

Returns the top margin.

const wxTextAttrDimension& wxTextBoxAttr::GetTopMargin ( ) const `[inline]`

wxTextAttrBorder& wxTextBoxAttr::GetTopOutline ( ) `[inline]`

Returns the top outline.

const wxTextAttrBorder& wxTextBoxAttr::GetTopOutline ( ) const `[inline]`

wxTextAttrDimension& wxTextBoxAttr::GetTopPadding ( ) `[inline]`

Returns the top padding value.

const wxTextAttrDimension& wxTextBoxAttr::GetTopPadding ( ) const `[inline]`

wxTextBoxAttrVerticalAlignment wxTextBoxAttr::GetVerticalAlignment ( ) const `[inline]`

Returns the vertical alignment.

wxTextBoxAttrWhitespaceMode wxTextBoxAttr::GetWhitespaceMode ( ) const `[inline]`

Returns the whitespace mode.

wxTextAttrDimension& wxTextBoxAttr::GetWidth ( ) `[inline]`

Returns the object width.

const wxTextAttrDimension& wxTextBoxAttr::GetWidth ( ) const `[inline]`

bool wxTextBoxAttr::HasBoxStyleName ( ) const `[inline]`

Returns true if the box style name is present.

bool wxTextBoxAttr::HasClearMode ( ) const `[inline]`

Returns true if we have a clear flag.

bool wxTextBoxAttr::HasCollapseBorders ( ) const `[inline]`

Returns true if the collapse borders flag is present.

bool wxTextBoxAttr::HasCornerRadius ( ) const `[inline]`

Returns true if the corner radius flag is present.

bool wxTextBoxAttr::HasFlag (wxTextBoxAttrFlags *flag*) const `[inline]`

Is this flag present?

bool wxTextBoxAttr::HasFloatMode ( ) const `[inline]`

Returns true if float mode is active.

bool wxTextBoxAttr::HasVerticalAlignment ( ) const `[inline]`

Returns true if a vertical alignment flag is present.

bool wxTextBoxAttr::HasWhitespaceMode ( ) const `[inline]`

Returns true if the whitespace flag is present.

void wxTextBoxAttr::Init ( ) `[inline]`

Initialises this object.

bool wxTextBoxAttr::IsDefault ( ) const

Returns true if no attributes are set.

bool wxTextBoxAttr::IsFloating ( ) const `[inline]`

Returns true if this object is floating.

bool wxTextBoxAttr::operator== (const wxTextBoxAttr & *attr*) const

Equality test.

void wxTextBoxAttr::RemoveFlag (wxTextBoxAttrFlags *flag*) `[inline]`

Removes this flag.

bool wxTextBoxAttr::RemoveStyle (const wxTextBoxAttr & *attr*)

Removes the specified attributes from this object.

void wxTextBoxAttr::Reset ( )

Resets this object.

void wxTextBoxAttr::SetBoxStyleName (const wxString & *name*) `[inline]`

Sets the box style name.

void wxTextBoxAttr::SetClearMode (wxTextBoxAttrClearStyle *mode*) `[inline]`

Set the clear mode.

Currently unimplemented.

void wxTextBoxAttr::SetCollapseBorders (wxTextBoxAttrCollapseMode *collapse*) `[inline]`

Sets the collapse mode - whether to collapse borders.

void wxTextBoxAttr::SetCornerRadius (const wxTextAttrDimension & *dim*) `[inline]`

Sets the corner radius value.

void wxTextBoxAttr::SetFlags (int *flags*) `[inline]`

Sets the flags.

void wxTextBoxAttr::SetFloatMode (wxTextBoxAttrFloatStyle *mode*) `[inline]`

Sets the float mode.

void wxTextBoxAttr::SetMaxSize (const wxTextAttrSize & *sz*) `[inline]`

Sets the object maximum size.

void wxTextBoxAttr::SetMinSize (const wxTextAttrSize & *sz*) `[inline]`

Sets the object minimum size.

void wxTextBoxAttr::SetSize (const wxTextAttrSize & *sz*) `[inline]`

Sets the object size.

void wxTextBoxAttr::SetVerticalAlignment (wxTextBoxAttrVerticalAlignment *verticalAlignment*) `[inline]`

Sets the vertical alignment.

void wxTextBoxAttr::SetWhitespaceMode (wxTextBoxAttrWhitespaceMode *whitespace*) `[inline]`

Sets the whitespace mode.

21.770.4 Member Data Documentation

wxTextAttrBorders wxTextBoxAttr::m_border

wxString wxTextBoxAttr::m_boxStyleName

wxTextBoxAttrClearStyle wxTextBoxAttr::m_clearMode

wxTextBoxAttrCollapseMode wxTextBoxAttr::m_collapseMode

wxTextAttrDimension wxTextBoxAttr::m_cornerRadius

int wxTextBoxAttr::m_flags

wxTextBoxAttrFloatStyle wxTextBoxAttr::m_floatMode

wxTextAttrDimensions wxTextBoxAttr::m_margins

wxTextAttrSize wxTextBoxAttr::m_maxSize

wxTextAttrSize wxTextBoxAttr::m_minSize

wxTextAttrBorders wxTextBoxAttr::m_outline

wxTextAttrDimensions wxTextBoxAttr::m_padding

wxTextAttrDimensions wxTextBoxAttr::m_position

wxTextAttrShadow wxTextBoxAttr::m_shadow

wxTextAttrSize wxTextBoxAttr::m_size

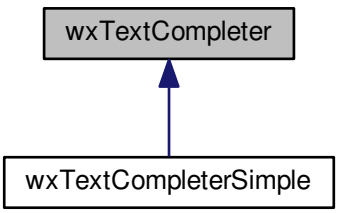
wxTextBoxAttrVerticalAlignment wxTextBoxAttr::m_verticalAlignment

wxTextBoxAttrWhitespaceMode wxTextBoxAttr::m_whitespaceMode

21.771 wxTextCompleter Class Reference

`#include <wx/textcompleter.h>`

Inheritance diagram for wxTextCompleter:



21.771.1 Detailed Description

Base class for custom text completer objects.

Custom completer objects used with wxTextEntry::AutoComplete() must derive from this class and implement its pure virtual method returning the completions. You would typically use a custom completer when the total number of completions is too big for performance to be acceptable if all of them need to be returned at once but if they can be generated hierarchically, i.e. only the first component initially, then the second one after the user finished entering the first one and so on.

When inheriting from this class you need to implement its two pure virtual methods. This allows to return the results incrementally and may or not be convenient depending on where do they come from. If you prefer to return all the completions at once, you should inherit from wxTextCompleterSimple instead.

Since

2.9.2

Public Member Functions

- virtual bool Start (const wxString &prefix)=0

  Function called to start iteration over the completions for the given prefix.
- virtual wxString GetNext ()=0

  Called to retrieve the next completion.

21.771.2 Member Function Documentation

virtual wxString wxTextCompleter::GetNext ( ) [pure virtual]

Called to retrieve the next completion.

All completions returned by this function should start with the prefix passed to the last call to Start().

Notice that, as Start(), this method is called from a worker thread context under MSW.

Returns

The next completion or an empty string to indicate that there are no more of them.

virtual bool wxTextCompleter::Start (const wxString & *prefix*) [pure virtual]

Function called to start iteration over the completions for the given prefix.

This function could start a database query, for example, if the results are read from a database.

Notice that under some platforms (currently MSW only) it is called from another thread context and so the appropriate synchronization mechanism should be used to access any data also used by the main UI thread.

Parameters

prefix	The prefix for which completions are to be generated.

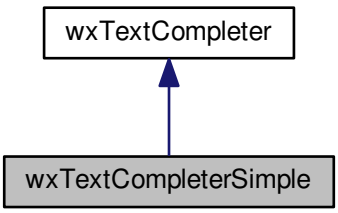
Returns

true to continue with calling GetNext() or false to indicate that there are no matches and GetNext() shouldn't be called at all.

21.772 wxTextCompleterSimple Class Reference

`#include <wx/textcompleter.h>`

Inheritance diagram for wxTextCompleterSimple:



21.772.1 Detailed Description

A simpler base class for custom completer objects.

This class may be simpler to use than the base wxTextCompleter as it allows to implement only a single virtual method instead of two of them (at the price of storing all completions in a temporary array).

Here is a simple example of a custom completer that completes the names of some chess pieces. Of course, as the total list here has only four items it would have been much simpler to just specify the array containing all the completions in this example but the same approach could be used when the total number of completions is much higher provided the number of possibilities for each word is still relatively small:

```
class MyTextCompleter : public wxTextCompleterSimple
{
public:
    virtual void GetCompletions(const wxString& prefix,
      wxArrayString& res)
    {
        const wxString firstWord = prefix.BeforeFirst(' ');
        if ( firstWord == "white" )
        {
            res.push_back("white pawn");
            res.push_back("white rook");
        }
        else if ( firstWord == "black" )
        {
            res.push_back("black king");
            res.push_back("black queen");
        }
        else
        {
            res.push_back("white");
            res.push_back("black");
        }
    }
};
...
wxTextCtrl *text = ...;
text->AutoComplete(new MyTextCompleter);
```

Library: wxCore

Since

2.9.2

Public Member Functions

- virtual void GetCompletions (const wxString &prefix, wxArrayString &res)=0

Pure virtual method returning all possible completions for the given prefix.

21.772.2 Member Function Documentation

virtual void wxTextCompleterSimple::GetCompletions (const wxString & *prefix,* wxArrayString & *res*) `[pure virtual]`

Pure virtual method returning all possible completions for the given prefix.

The custom completer should examine the provided prefix and return all the possible completions for it in the output array *res*.

Please notice that the returned values should start with the prefix, otherwise they will be simply ignored, making adding them to the array in the first place useless.

Notice that this function may be called from thread other than main one (this is currently always the case under MSW) so the appropriate synchronization mechanism should be used to protect the shared data.

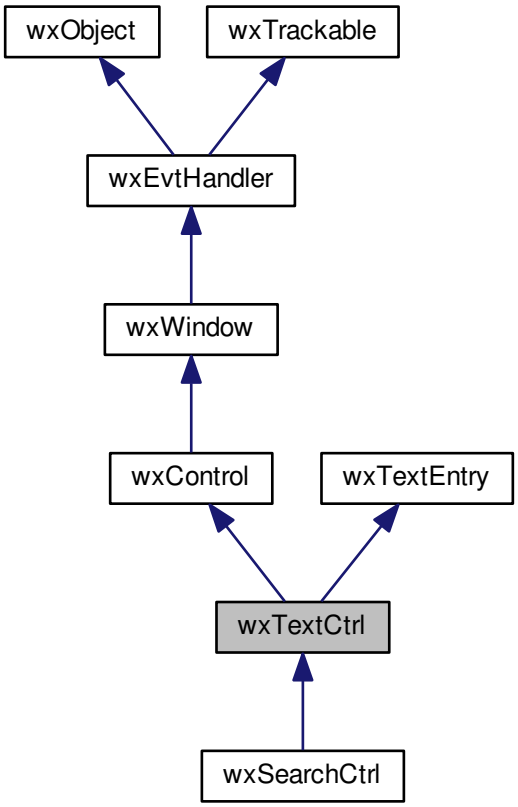
Parameters

prefix	The possibly empty prefix that the user had already entered.
res	Initially empty array that should be filled with all possible completions (possibly none if there are no valid possibilities starting with the given prefix).

21.773 wxTextCtrl Class Reference

`#include <wx/textctrl.h>`

Inheritance diagram for wxTextCtrl:



21.773.1 Detailed Description

A text control allows text to be displayed and edited.

It may be single line or multi-line. Notice that a lot of methods of the text controls are found in the base wxTextEntry class which is a common base class for wxTextCtrl and other controls using a single line text entry field (e.g. wxComboBox).

Styles

This class supports the following styles:

- wxTE_PROCESS_ENTER: The control will generate the event `wxEVT_TEXT_ENTER` (otherwise pressing Enter key is either processed internally by the control or used for navigation between dialog controls).
- wxTE_PROCESS_TAB: The control will receive `wxEVT_CHAR` events for TAB pressed - normally, TAB is used for passing to the next control in a dialog instead. For the control created with this style, you can still use Ctrl-Enter to pass to the next control from the keyboard.
- wxTE_MULTILINE: The text control allows multiple lines. If this style is not specified, line break characters should not be used in the controls value.
- wxTE_PASSWORD: The text will be echoed as asterisks.

- wxTE_READONLY: The text will not be user-editable.
- wxTE_RICH: Use rich text control under Win32, this allows to have more than 64KB of text in the control even under Win9x. This style is ignored under other platforms.
- wxTE_RICH2: Use rich text control version 2.0 or 3.0 under Win32, this style is ignored under other platforms
- wxTE_AUTO_URL: Highlight the URLs and generate the wxTextUrlEvents when mouse events occur over them. This style is only supported for wxTE_RICH Win32 and multi-line wxGTK2 text controls.
- wxTE_NOHIDESEL: By default, the Windows text control doesn't show the selection when it doesn't have focus - use this style to force it to always show it. It doesn't do anything under other platforms.
- wxHSCROLL: A horizontal scrollbar will be created and used, so that text won't be wrapped. No effect under wxGTK1.
- wxTE_NO_VSCROLL: For multiline controls only: vertical scrollbar will never be created. This limits the amount of text which can be entered into the control to what can be displayed in it under MSW but not under GTK2. Currently not implemented for the other platforms.
- wxTE_LEFT: The text in the control will be left-justified (default).
- wxTE_CENTRE: The text in the control will be centered (currently wxMSW and wxGTK2 only).
- wxTE_RIGHT: The text in the control will be right-justified (currently wxMSW and wxGTK2 only).
- wxTE_DONTWRAP: Same as wxHSCROLL style: don't wrap at all, show horizontal scrollbar instead.
- wxTE_CHARWRAP: Wrap the lines too long to be shown entirely at any position (wxUniv and wxGTK2 only).
- wxTE_WORDWRAP: Wrap the lines too long to be shown entirely at word boundaries (wxUniv and wxGTK2 only).
- wxTE_BESTWRAP: Wrap the lines at word boundaries or at any other character if there are words longer than the window width (this is the default).
- wxTE_CAPITALIZE: On PocketPC and Smartphone, causes the first letter to be capitalized.

Note that alignment styles (wxTE_LEFT, wxTE_CENTRE and wxTE_RIGHT) can be changed dynamically after control creation on wxMSW and wxGTK. wxTE_READONLY, wxTE_PASSWORD and wrapping styles can be dynamically changed under wxGTK but not wxMSW. The other styles can be only set during control creation.

21.773.2 wxTextCtrl Text Format

The multiline text controls always store the text as a sequence of lines separated by `'\n'` characters, i.e. in the Unix text format even on non-Unix platforms. This allows the user code to ignore the differences between the platforms but at a price: the indices in the control such as those returned by GetInsertionPoint() or GetSelection() can **not** be used as indices into the string returned by GetValue() as they're going to be slightly off for platforms using `"\\r\\n"` as separator (as Windows does).

Instead, if you need to obtain a substring between the 2 indices obtained from the control with the help of the functions mentioned above, you should use GetRange(). And the indices themselves can only be passed to other methods, for example SetInsertionPoint() or SetSelection().

To summarize: never use the indices returned by (multiline) wxTextCtrl as indices into the string it contains, but only as arguments to be passed back to the other wxTextCtrl methods. This problem doesn't arise for single-line platforms however where the indices in the control do correspond to the positions in the value string.

21.773.3 wxTextCtrl Styles.

Multi-line text controls support styling, i.e. provide a possibility to set colours and font for individual characters in it (note that under Windows `wxTE_RICH` style is required for style support). To use the styles you can either call SetDefaultStyle() before inserting the text or call SetStyle() later to change the style of the text already in the control (the first solution is much more efficient).

In either case, if the style doesn't specify some of the attributes (for example you only want to set the text colour but without changing the font nor the text background), the values of the default style will be used for them. If there is no default style, the attributes of the text control itself are used.

So the following code correctly describes what it does: the second call to SetDefaultStyle() doesn't change the text foreground colour (which stays red) while the last one doesn't change the background colour (which stays grey):

```
text->SetDefaultStyle(wxTextAttr(*wxRED));
text->AppendText("Red text\n");
text->SetDefaultStyle(wxTextAttr(wxNullColour, *
      wxLIGHT_GREY));
text->AppendText("Red on grey text\n");
text->SetDefaultStyle(wxTextAttr(*wxBLUE);
text->AppendText("Blue on grey text\n");
```

21.773.4 wxTextCtrl and C++ Streams

This class multiply-inherits from `std::streambuf` (except for some really old compilers using non-standard iostream library), allowing code such as the following:

```
wxTextCtrl *control = new wxTextCtrl(...);

ostream stream(control)

stream << 123.456 << " some text\n";
stream.flush();
```

Note that even if your build of wxWidgets doesn't support this (the symbol `wxHAS_TEXT_WINDOW_STREAM` has value of 0 then) you can still use wxTextCtrl itself in a stream-like manner:

```
wxTextCtrl *control = new wxTextCtrl(...);

*control << 123.456 << " some text\n";
```

However the possibility to create an ostream associated with wxTextCtrl may be useful if you need to redirect the output of a function taking an ostream as parameter to a text control.

Another commonly requested need is to redirect `std::cout` to the text control. This may be done in the following way:

```
#include <iostream>

wxTextCtrl *control = new wxTextCtrl(...);

std::streambuf *sbOld = std::cout.rdbuf();
std::cout.rdbuf(control);

// use cout as usual, the output appears in the text control
...

std::cout.rdbuf(sbOld);
```

But wxWidgets provides a convenient class to make it even simpler so instead you may just do

```
#include <iostream>

wxTextCtrl *control = new wxTextCtrl(...);

wxStreamToTextRedirector redirect(control);

// all output to cout goes into the text control until the exit from
// current scope
```

See wxStreamToTextRedirector for more details.

21.773.5 Event Handling.

The following commands are processed by default event handlers in wxTextCtrl: `wxID_CUT`, `wxID_COPY`, `wxID_PASTE`, `wxID_UNDO`, `wxID_REDO`. The associated UI update events are also processed automatically, when the control has the focus.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_TEXT(id, func): Respond to a `wxEVT_TEXT` event, generated when the text changes. Notice that this event will be sent when the text controls contents changes – whether this is due to user input or comes from the program itself (for example, if wxTextCtrl::SetValue() is called); see wxTextCtrl::ChangeValue() for a function which does not send this event. This event is however not sent during the control creation.
- EVT_TEXT_ENTER(id, func): Respond to a `wxEVT_TEXT_ENTER` event, generated when enter is pressed in a text control which must have wxTE_PROCESS_ENTER style for this event to be generated.
- EVT_TEXT_URL(id, func): A mouse event occurred over an URL in the text control (wxMSW and wxGTK2 only currently).
- EVT_TEXT_MAXLEN(id, func): This event is generated when the user tries to enter more text into the control than the limit set by wxTextCtrl::SetMaxLength(), see its description.

Library: wxCore

Category: Controls

See also

wxTextCtrl::Create, wxValidator

Public Member Functions

- wxTextCtrl ()

  Default ctor.
- wxTextCtrl (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxTextCtrlNameStr)

  Constructor, creating and showing a text control.
- virtual ~wxTextCtrl ()

  Destructor, destroying the text control.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &value=wxEmptyString, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxTextCtrlNameStr)

  Creates the text control for two-step construction.
- virtual void DiscardEdits ()

  Resets the internal modified flag as if the current changes had been saved.
- virtual bool EmulateKeyPress (const wxKeyEvent &event)

  This function inserts into the control the character which would have been inserted if the given key event had occurred in the text control.
- virtual const wxTextAttr & GetDefaultStyle () const

Returns the style currently used for the new text.

- virtual int GetLineLength (long lineNo) const

  Gets the length of the specified line, not including any trailing newline character(s).
- virtual wxString GetLineText (long lineNo) const

  Returns the contents of a given line in the text control, not including any trailing newline character(s).
- virtual int GetNumberOfLines () const

  Returns the number of lines in the text control buffer.
- virtual bool GetStyle (long position, wxTextAttr &style)

  Returns the style at this position in the text control.
- wxTextCtrlHitTestResult HitTest (const wxPoint &pt, long ∗pos) const

  Finds the position of the character at the specified point.
- wxTextCtrlHitTestResult HitTest (const wxPoint &pt, wxTextCoord ∗col, wxTextCoord ∗row) const

  Finds the row and column of the character at the specified point.
- virtual bool IsModified () const

  Returns true if the text has been modified by user.
- bool IsMultiLine () const

  Returns true if this is a multi line edit control and false otherwise.
- bool IsSingleLine () const

  Returns true if this is a single line edit control and false otherwise.
- bool LoadFile (const wxString &filename, int fileType=wxTEXT_TYPE_ANY)

  Loads and displays the named file, if it exists.
- virtual void MarkDirty ()

  Mark text as modified (dirty).
- void OnDropFiles (wxDropFilesEvent &event)

  This event handler function implements default drag and drop behaviour, which is to load the first dropped file into the control.
- virtual bool PositionToXY (long pos, long ∗x, long ∗y) const

  Converts given position to a zero-based column, line number pair.
- wxPoint PositionToCoords (long pos) const

  Converts given text position to client coordinates in pixels.
- bool SaveFile (const wxString &filename=wxEmptyString, int fileType=wxTEXT_TYPE_ANY)

  Saves the contents of the control in a text file.
- virtual bool SetDefaultStyle (const wxTextAttr &style)

  Changes the default style to use for the new text which is going to be added to the control using WriteText() or AppendText().
- void SetModified (bool modified)

  Marks the control as being modified by the user or not.
- virtual bool SetStyle (long start, long end, const wxTextAttr &style)

  Changes the style of the given range.
- virtual void ShowPosition (long pos)

  Makes the line containing the given position visible.
- virtual long XYToPosition (long x, long y) const

  Converts the given zero based column and line number to a position.

- wxTextCtrl & operator<< (const wxString &s)

  Operator definitions for appending to a text control.
- wxTextCtrl & operator<< (int i)

  Operator definitions for appending to a text control.
- wxTextCtrl & operator<< (long i)

  Operator definitions for appending to a text control.
- wxTextCtrl & operator<< (float f)

Operator definitions for appending to a text control.

- wxTextCtrl & operator<< (double d)

  Operator definitions for appending to a text control.
- wxTextCtrl & operator<< (char c)

  Operator definitions for appending to a text control.
- wxTextCtrl & operator<< (wchar_t c)

  Operator definitions for appending to a text control.

Additional Inherited Members

21.773.6 Constructor & Destructor Documentation

wxTextCtrl::wxTextCtrl ( )

Default ctor.

wxTextCtrl::wxTextCtrl (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxTextCtrlNameStr)

Constructor, creating and showing a text control.

Parameters

parent	Parent window. Should not be NULL.
id	Control identifier. A value of -1 denotes a default value.
value	Default text value.
pos	Text control position.
size	Text control size.
style	Window style. See wxTextCtrl.
validator	Window validator.
name	Window name.

Remarks

The horizontal scrollbar (wxHSCROLL style flag) will only be created for multi-line text controls. Without a horizontal scrollbar, text lines that don't fit in the control's size will be wrapped (but no newline character is inserted). Single line controls don't have a horizontal scrollbar, the text is automatically scrolled so that the insertion point is always visible.

See also

Create(), wxValidator

virtual wxTextCtrl::~wxTextCtrl ( ) [virtual]

Destructor, destroying the text control.

21.773.7 Member Function Documentation

bool wxTextCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *value* = wxEmptyString, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxTextCtrlNameStr)

Creates the text control for two-step construction.

This method should be called if the default constructor was used for the control creation. Its parameters have the same meaning as for the non-default constructor.

virtual void wxTextCtrl::DiscardEdits ( ) [virtual]

Resets the internal modified flag as if the current changes had been saved.

virtual bool wxTextCtrl::EmulateKeyPress (const wxKeyEvent & *event*) [virtual]

This function inserts into the control the character which would have been inserted if the given key event had occurred in the text control.

The *event* object should be the same as the one passed to `EVT_KEY_DOWN` handler previously by wxWidgets. Please note that this function doesn't currently work correctly for all keys under any platform but MSW.

Returns

true if the event resulted in a change to the control, false otherwise.

virtual const wxTextAttr& wxTextCtrl::GetDefaultStyle ( ) const [virtual]

Returns the style currently used for the new text.

See also

SetDefaultStyle()

virtual int wxTextCtrl::GetLineLength (long *lineNo*) const [virtual]

Gets the length of the specified line, not including any trailing newline character(s).

Parameters

lineNo	Line number (starting from zero).

Returns

The length of the line, or -1 if *lineNo* was invalid.

virtual wxString wxTextCtrl::GetLineText (long *lineNo*) const [virtual]

Returns the contents of a given line in the text control, not including any trailing newline character(s).

Parameters

lineNo	The line number, starting from zero.

Returns

The contents of the line.

virtual int wxTextCtrl::GetNumberOfLines ( ) const [virtual]

Returns the number of lines in the text control buffer.

The returned number is the number of logical lines, i.e. just the count of the number of newline characters in the control + 1, for wxGTK and wxOSX/Cocoa ports while it is the number of physical lines, i.e. the count of lines actually shown in the control, in wxMSW and wxOSX/Carbon. Because of this discrepancy, it is not recommended to use this function.

Remarks

Note that even empty text controls have one line (where the insertion point is), so GetNumberOfLines() never returns 0.

virtual bool wxTextCtrl::GetStyle (long *position,* wxTextAttr & *style*) [virtual]

Returns the style at this position in the text control.

Not all platforms support this function.

Returns

true on success, false if an error occurred (this may also mean that the styles are not supported under this platform).

See also

SetStyle(), wxTextAttr

wxTextCtrlHitTestResult wxTextCtrl::HitTest (const wxPoint & *pt,* long * *pos*) const

Finds the position of the character at the specified point.

If the return code is not wxTE_HT_UNKNOWN the row and column of the character closest to this position are returned, otherwise the output parameters are not modified.

Please note that this function is currently only implemented in wxUniv, wxMSW and wxGTK2 ports and always returns wxTE_HT_UNKNOWN in the other ports.

wxPerl Note: In wxPerl this function takes only the *pt* argument and returns a 3-element list (result, col, row).

Parameters

pt	The position of the point to check, in window device coordinates.
pos	Receives the position of the character at the given position. May be NULL.

See also

PositionToXY(), XYToPosition()

wxTextCtrlHitTestResult wxTextCtrl::HitTest (const wxPoint & *pt,* wxTextCoord * *col,* wxTextCoord * *row*) const

Finds the row and column of the character at the specified point.

If the return code is not wxTE_HT_UNKNOWN the row and column of the character closest to this position are returned, otherwise the output parameters are not modified.

Please note that this function is currently only implemented in wxUniv, wxMSW and wxGTK2 ports and always returns wxTE_HT_UNKNOWN in the other ports.

wxPerl Note: In wxPerl this function takes only the *pt* argument and returns a 3-element list (result, col, row).

Parameters

pt	The position of the point to check, in window device coordinates.
col	Receives the column of the character at the given position. May be NULL.
row	Receives the row of the character at the given position. May be NULL.

See also

PositionToXY(), XYToPosition()

virtual bool wxTextCtrl::IsModified ( ) const `[virtual]`

Returns true if the text has been modified by user.

Note that calling SetValue() doesn't make the control modified.

See also

MarkDirty()

bool wxTextCtrl::IsMultiLine ( ) const

Returns true if this is a multi line edit control and false otherwise.

See also

IsSingleLine()

bool wxTextCtrl::IsSingleLine ( ) const

Returns true if this is a single line edit control and false otherwise.

See also

IsSingleLine(), IsMultiLine()

bool wxTextCtrl::LoadFile (const wxString & *filename*, int *fileType* = wxTEXT_TYPE_ANY)

Loads and displays the named file, if it exists.

Parameters

filename	The filename of the file to load.
fileType	The type of file to load. This is currently ignored in wxTextCtrl.

Returns

true if successful, false otherwise.

virtual void wxTextCtrl::MarkDirty ( ) `[virtual]`

Mark text as modified (dirty).

See also

IsModified()

void wxTextCtrl::OnDropFiles (wxDropFilesEvent & *event*)

This event handler function implements default drag and drop behaviour, which is to load the first dropped file into the control.

Parameters

event	The drop files event.

Remarks

This is not implemented on non-Windows platforms.

See also

wxDropFilesEvent

wxTextCtrl& wxTextCtrl::operator<< (const wxString & *s*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);
(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxTextCtrl& wxTextCtrl::operator<< (int *i*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);
(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxTextCtrl& wxTextCtrl::operator<< (long *i*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);
(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxTextCtrl& wxTextCtrl::operator<< (float *f*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);
(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxTextCtrl& wxTextCtrl::operator<< (double *d*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);
(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxTextCtrl& wxTextCtrl::operator<< (char *c*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);

(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxTextCtrl& wxTextCtrl::operator<< (wchar_t *c*)

Operator definitions for appending to a text control.

These operators can be used as with the standard C++ streams, for example:

```
wxTextCtrl *wnd = new wxTextCtrl(my_frame);

(*wnd) << "Welcome to text control number " << 1 << ".\n";
```

wxPoint wxTextCtrl::PositionToCoords (long *pos*) const

Converts given text position to client coordinates in pixels.

This function allows to find where is the character at the given position displayed in the text control.

Availability: only available for the wxMSW, wxGTK ports.. Additionally, wxGTK only implements this method for multiline controls and wxDefaultPosition is always returned for the single line ones.

Parameters

pos	Text position in 0 to GetLastPosition() range (inclusive).

Returns

On success returns a wxPoint which contains client coordinates for the given position in pixels, otherwise returns wxDefaultPosition.

Since

2.9.3

See also

XYToPosition(), PositionToXY()

virtual bool wxTextCtrl::PositionToXY (long *pos*, long * *x*, long * *y*) const [virtual]

Converts given position to a zero-based column, line number pair.

Parameters

pos	Position.
x	Receives zero based column number.

y	Receives zero based line number.

Returns

true on success, false on failure (most likely due to a too large position parameter).

wxPerl Note: In wxPerl this function takes only the *pos* argument and returns a 2-element list (x, y).

See also

XYToPosition()

bool wxTextCtrl::SaveFile (const wxString & *filename* = wxEmptyString, int *fileType* = wxTEXT_TYPE_ANY)

Saves the contents of the control in a text file.

Parameters

filename	The name of the file in which to save the text.
fileType	The type of file to save. This is currently ignored in wxTextCtrl.

Returns

true if the operation was successful, false otherwise.

virtual bool wxTextCtrl::SetDefaultStyle (const wxTextAttr & *style*) [virtual]

Changes the default style to use for the new text which is going to be added to the control using WriteText() or AppendText().

If either of the font, foreground, or background colour is not set in *style*, the values of the previous default style are used for them. If the previous default style didn't set them neither, the global font or colours of the text control itself are used as fall back.

However if the *style* parameter is the default wxTextAttr, then the default style is just reset (instead of being combined with the new style which wouldn't change it at all).

Parameters

style	The style for the new text.

Returns

true on success, false if an error occurred (this may also mean that the styles are not supported under this platform).

See also

GetDefaultStyle()

void wxTextCtrl::SetModified (bool *modified*)

Marks the control as being modified by the user or not.

See also

MarkDirty(), DiscardEdits()

virtual bool wxTextCtrl::SetStyle (long *start,* long *end,* const wxTextAttr & *style*) `[virtual]`

Changes the style of the given range.

If any attribute within *style* is not set, the corresponding attribute from GetDefaultStyle() is used.

Parameters

start	The start of the range to change.
end	The end of the range to change.
style	The new style for the range.

Returns

true on success, false if an error occurred (this may also mean that the styles are not supported under this platform).

See also

GetStyle(), wxTextAttr

virtual void wxTextCtrl::ShowPosition (long *pos*) `[virtual]`

Makes the line containing the given position visible.

Parameters

pos	The position that should be visible.

virtual long wxTextCtrl::XYToPosition (long *x,* long *y*) const `[virtual]`

Converts the given zero based column and line number to a position.

Parameters

x	The column number.
y	The line number.

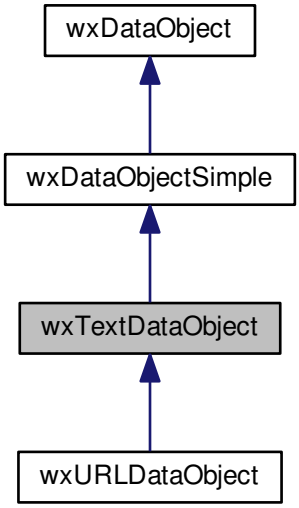
Returns

The position value, or -1 if x or y was invalid.

21.774 wxTextDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxTextDataObject:



21.774.1 Detailed Description

wxTextDataObject is a specialization of wxDataObjectSimple for text data.

It can be used without change to paste data into the wxClipboard or a wxDropSource. A user may wish to derive a new class from this class for providing text on-demand in order to minimize memory consumption when offering data in several formats, such as plain text and RTF because by default the text is stored in a string in this class, but it might as well be generated when requested. For this, GetTextLength() and GetText() will have to be overridden.

Note that if you already have the text inside a string, you will not achieve any efficiency gain by overriding these functions because copying wxStrings is already a very efficient operation (data is not actually copied because wx↩ Strings are reference counted).

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject, wxDataObjectSimple, wxFileDataObject, wxBitmapDataObject

Public Member Functions

- wxTextDataObject (const wxString &text=wxEmptyString)

  Constructor, may be used to initialise the text (otherwise SetText() should be used later).
- virtual wxString GetText () const

  Returns the text associated with the data object.
- virtual size_t GetTextLength () const

  Returns the data size.
- virtual size_t GetFormatCount (wxDataObject::Direction dir=wxDataObject::Get) const

  Returns 2 under wxMac and wxGTK, where text data coming from the clipboard may be provided as ANSI (wxDF_TEXT) or as Unicode text (wxDF_UNICODETEXT, but only when wxUSE_UNICODE==1).
- const wxDataFormat & GetFormat () const

  Returns the preferred format supported by this object.
- virtual void GetAllFormats (wxDataFormat *formats, wxDataObject::Direction dir=wxDataObject::Get) const

  Returns all the formats supported by wxTextDataObject.
- virtual void SetText (const wxString &strText)

  Sets the text associated with the data object.

Additional Inherited Members

21.774.2 Constructor & Destructor Documentation

wxTextDataObject::wxTextDataObject (const wxString & *text* = wxEmptyString)

Constructor, may be used to initialise the text (otherwise SetText() should be used later).

21.774.3 Member Function Documentation

virtual void wxTextDataObject::GetAllFormats (wxDataFormat * *formats*, wxDataObject::Direction *dir* = wxDataObject::Get) const `[virtual]`

Returns all the formats supported by wxTextDataObject.

Under wxMac and wxGTK they are `wxDF_TEXT` and `wxDF_UNICODETEXT`, under other ports returns only one of the two, depending on the build mode.

Implements wxDataObject.

const wxDataFormat& wxTextDataObject::GetFormat ( ) const

Returns the preferred format supported by this object.

This is `wxDF_TEXT` or `wxDF_UNICODETEXT` depending on the platform and from the build mode (i.e. from `wxUSE_UNICODE`).

virtual size_t wxTextDataObject::GetFormatCount (wxDataObject::Direction *dir* = wxDataObject::Get) const `[virtual]`

Returns 2 under wxMac and wxGTK, where text data coming from the clipboard may be provided as ANSI (`wxDF_TEXT`) or as Unicode text (`wxDF_UNICODETEXT`, but only when `wxUSE_UNICODE==1`).

Returns 1 under other platforms (e.g. wxMSW) or when building in ANSI mode (`wxUSE_UNICODE==0`).

Implements wxDataObject.

virtual wxString wxTextDataObject::GetText ( ) const `[virtual]`

Returns the text associated with the data object.

You may wish to override this method when offering data on-demand, but this is not required by wxWidgets' internals. Use this method to get data in text form from the wxClipboard.

virtual size_t wxTextDataObject::GetTextLength ( ) const `[virtual]`

Returns the data size.

By default, returns the size of the text data set in the constructor or using SetText(). This can be overridden to provide text size data on-demand. It is recommended to return the text length plus 1 for a trailing zero, but this is not strictly required.

virtual void wxTextDataObject::SetText (const wxString & *strText*) `[virtual]`

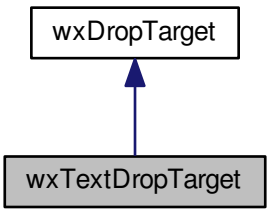
Sets the text associated with the data object.

This method is called when the data object receives the data and, by default, copies the text into the member variable. If you want to process the text on the fly you may wish to override this function.

21.775 wxTextDropTarget Class Reference

`#include <wx/dnd.h>`

Inheritance diagram for wxTextDropTarget:



21.775.1 Detailed Description

A predefined drop target for dealing with text data.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDropSource, wxDropTarget, wxFileDropTarget

Public Member Functions

- wxTextDropTarget ()

  Constructor.
- virtual bool OnDrop (wxCoord x, wxCoord y)

  See wxDropTarget::OnDrop().
- virtual bool OnDropText (wxCoord x, wxCoord y, const wxString &data)=0

  Override this function to receive dropped text.

21.775.2 Constructor & Destructor Documentation

wxTextDropTarget::wxTextDropTarget ( )

Constructor.

21.775.3 Member Function Documentation

virtual bool wxTextDropTarget::OnDrop (wxCoord *x*, wxCoord *y*) `[virtual]`

See wxDropTarget::OnDrop().

This function is implemented appropriately for text, and calls OnDropText().

Reimplemented from wxDropTarget.

virtual bool wxTextDropTarget::OnDropText (wxCoord *x*, wxCoord *y*, const wxString & *data*) `[pure virtual]`

Override this function to receive dropped text.

Parameters

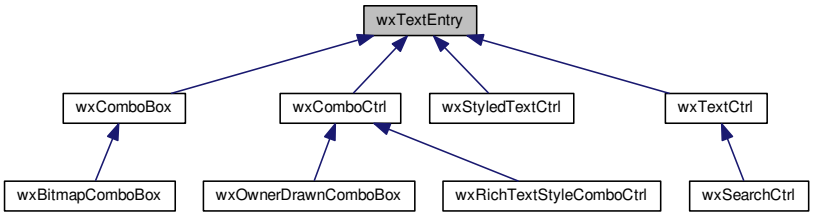
x	The x coordinate of the mouse.
y	The y coordinate of the mouse.
data	The data being dropped: a wxString.

Return true to accept the data, or false to veto the operation.

21.776 wxTextEntry Class Reference

`#include <wx/textentry.h>`

Inheritance diagram for wxTextEntry:



21.776.1 Detailed Description

Common base class for single line text entry fields.

This class is not a control itself, as it doesn't derive from wxWindow. Instead it is used as a base class by other controls, notably wxTextCtrl and wxComboBox and gathers the methods common to both of them.

Library: wxCore

Category: Controls

See also

wxTextCtrl, wxComboBox

Since

2.9.0

Public Member Functions

- virtual void AppendText (const wxString &text)

  Appends the text to the end of the text control.
- bool AutoComplete (const wxArrayString &choices)

  Call this function to enable auto-completion of the text typed in a single-line text control using the given choices.
- bool AutoComplete (wxTextCompleter *completer)

  Enable auto-completion using the provided completer object.
- bool AutoCompleteFileNames ()

  Call this function to enable auto-completion of the text typed in a single-line text control using all valid file system paths.
- bool AutoCompleteDirectories ()

  Call this function to enable auto-completion of the text using the file system directories.
- virtual bool CanCopy () const

  Returns true if the selection can be copied to the clipboard.
- virtual bool CanCut () const

  Returns true if the selection can be cut to the clipboard.
- virtual bool CanPaste () const

  Returns true if the contents of the clipboard can be pasted into the text control.
- virtual bool CanRedo () const

  Returns true if there is a redo facility available and the last operation can be redone.
- virtual bool CanUndo () const

  Returns true if there is an undo facility available and the last operation can be undone.
- virtual void ChangeValue (const wxString &value)

  Sets the new text control value.
- virtual void Clear ()

  Clears the text in the control.
- virtual void Copy ()

  Copies the selected text to the clipboard.
- virtual void Cut ()

  Copies the selected text to the clipboard and removes it from the control.
- virtual long GetInsertionPoint () const

Returns the insertion point, or cursor, position.

- virtual wxTextPos GetLastPosition () const

  Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.

- virtual wxString GetRange (long from, long to) const

  Returns the string containing the text starting in the positions from and up to to in the control.

- virtual void GetSelection (long *from, long *to) const

  Gets the current selection span.

- virtual wxString GetStringSelection () const

  Gets the text currently selected in the control.

- virtual wxString GetValue () const

  Gets the contents of the control.

- virtual bool IsEditable () const

  Returns true if the controls contents may be edited by user (note that it always can be changed by the program).

- virtual bool IsEmpty () const

  Returns true if the control is currently empty.

- virtual void Paste ()

  Pastes text from the clipboard to the text item.

- virtual void Redo ()

  If there is a redo facility and the last operation can be redone, redoes the last operation.

- virtual void Remove (long from, long to)

  Removes the text starting at the first given position up to (but not including) the character at the last position.

- virtual void Replace (long from, long to, const wxString &value)

  Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.

- virtual void SetEditable (bool editable)

  *Makes the text item editable or read-only, overriding the **wxTE_READONLY** flag.*

- virtual void SetInsertionPoint (long pos)

  Sets the insertion point at the given position.

- virtual void SetInsertionPointEnd ()

  Sets the insertion point at the end of the text control.

- virtual void SetMaxLength (unsigned long len)

  This function sets the maximum number of characters the user can enter into the control.

- virtual void SetSelection (long from, long to)

  Selects the text starting at the first position up to (but not including) the character at the last position.

- virtual void SelectAll ()

  Selects all text in the control.

- virtual void SelectNone ()

  Deselects selected text in the control.

- virtual bool SetHint (const wxString &hint)

  Sets a hint shown in an empty unfocused text control.

- virtual wxString GetHint () const

  Returns the current hint string.

- wxPoint GetMargins () const

  Returns the margins used by the control.

- virtual void SetValue (const wxString &value)

  Sets the new text control value.

- virtual void Undo ()

  If there is an undo facility and the last operation can be undone, undoes the last operation.

- virtual void WriteText (const wxString &text)

  Writes the text into the text control at the current insertion position.

- bool SetMargins (const wxPoint &pt)

  Attempts to set the control margins.
- bool SetMargins (wxCoord left, wxCoord top=-1)

  Attempts to set the control margins.

21.776.2 Member Function Documentation

virtual void wxTextEntry::AppendText (const wxString & *text*) `[virtual]`

Appends the text to the end of the text control.

Parameters

text	Text to write to the text control.

Remarks

After the text is appended, the insertion point will be at the end of the text control. If this behaviour is not desired, the programmer should use GetInsertionPoint() and SetInsertionPoint().

See also

WriteText()

Reimplemented in wxStyledTextCtrl.

bool wxTextEntry::AutoComplete (const wxArrayString & *choices*)

Call this function to enable auto-completion of the text typed in a single-line text control using the given *choices*.

Notice that currently this function is only implemented in wxGTK2, wxMSW and wxOSX/Cocoa (for wxTextCtrl only, but not for wxComboBox) ports and does nothing under the other platforms.

Since

2.9.0

Returns

true if the auto-completion was enabled or false if the operation failed, typically because auto-completion is not supported by the current platform.

See also

AutoCompleteFileNames()

bool wxTextEntry::AutoComplete (wxTextCompleter * *completer*)

Enable auto-completion using the provided completer object.

This method should be used instead of AutoComplete() overload taking the array of possible completions if the total number of strings is too big as it allows to return the completions dynamically, depending on the text already entered by user and so is more efficient.

The specified *completer* object will be used to retrieve the list of possible completions for the already entered text and will be deleted by wxTextEntry itself when it's not needed any longer.

Notice that you need to include wx/textcompleter.h in order to define your class inheriting from wxTextCompleter.

Currently this method is only implemented in wxMSW and wxOSX/Cocoa (for wxTextCtrl only, but not for wxComboBox).

Since

2.9.2

Parameters

completer	The object to be used for generating completions if non-NULL. If it is NULL, auto-completion is disabled. The wxTextEntry object takes ownership of this pointer and will delete it in any case (i.e. even if this method returns false).

Returns

true if the auto-completion was enabled or false if the operation failed, typically because auto-completion is not supported by the current platform.

See also

wxTextCompleter

bool wxTextEntry::AutoCompleteDirectories ( )

Call this function to enable auto-completion of the text using the file system directories.

Unlike AutoCompleteFileNames() which completes both file names and directories, this function only completes the directory names.

Notice that currently this function is only implemented in wxMSW port and does nothing under the other platforms.

Since

2.9.3

Returns

true if the auto-completion was enabled or false if the operation failed, typically because auto-completion is not supported by the current platform.

See also

AutoComplete()

bool wxTextEntry::AutoCompleteFileNames ( )

Call this function to enable auto-completion of the text typed in a single-line text control using all valid file system paths.

Notice that currently this function is only implemented in wxMSW port and does nothing under the other platforms.

Since

2.9.0

Returns

true if the auto-completion was enabled or false if the operation failed, typically because auto-completion is not supported by the current platform.

See also

AutoComplete()

virtual bool wxTextEntry::CanCopy ( ) const `[virtual]`

Returns true if the selection can be copied to the clipboard.

virtual bool wxTextEntry::CanCut ( ) const `[virtual]`

Returns true if the selection can be cut to the clipboard.

virtual bool wxTextEntry::CanPaste ( ) const `[virtual]`

Returns true if the contents of the clipboard can be pasted into the text control.

On some platforms (Motif, GTK) this is an approximation and returns true if the control is editable, false otherwise.

Reimplemented in wxStyledTextCtrl.

virtual bool wxTextEntry::CanRedo ( ) const `[virtual]`

Returns true if there is a redo facility available and the last operation can be redone.

Reimplemented in wxStyledTextCtrl.

virtual bool wxTextEntry::CanUndo ( ) const `[virtual]`

Returns true if there is an undo facility available and the last operation can be undone.

Reimplemented in wxStyledTextCtrl.

virtual void wxTextEntry::ChangeValue (const wxString & *value*) `[virtual]`

Sets the new text control value.

It also marks the control as not-modified which means that IsModified() would return false immediately after the call to ChangeValue().

The insertion point is set to the start of the control (i.e. position 0) by this function.

This functions does not generate the `wxEVT_TEXT` event but otherwise is identical to SetValue().

See User Generated Events vs Programmatically Generated Events for more information.

Since

2.7.1

Parameters

value	The new value to set. It may contain newline characters if the text control is multi-line.

virtual void wxTextEntry::Clear ( ) [virtual]

Clears the text in the control.

Note that this function will generate a `wxEVT_TEXT` event, i.e. its effect is identical to calling `SetValue("")`.

Reimplemented in wxStyledTextCtrl.

virtual void wxTextEntry::Copy ( ) [virtual]

Copies the selected text to the clipboard.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual void wxTextEntry::Cut ( ) [virtual]

Copies the selected text to the clipboard and removes it from the control.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual wxString wxTextEntry::GetHint ( ) const [virtual]

Returns the current hint string.

See SetHint() for more information about hints.

Since

2.9.0

Reimplemented in wxComboCtrl.

virtual long wxTextEntry::GetInsertionPoint ( ) const [virtual]

Returns the insertion point, or cursor, position.

This is defined as the zero based index of the character position to the right of the insertion point. For example, if the insertion point is at the end of the single-line text control, it is equal to GetLastPosition().

Notice that insertion position is, in general, different from the index of the character the cursor position at in the string returned by GetValue(). While this is always the case for the single line controls, multi-line controls can use two characters `"\\r\\n"` as line separator (this is notably the case under MSW) meaning that indices in the control and its string value are offset by 1 for every line.

Hence to correctly get the character at the current cursor position, taking into account that there can be none if the cursor is at the end of the string, you could do the following:

```
wxString GetCurrentChar(wxTextCtrl *tc)
{
    long pos = tc->GetInsertionPoint();
    if ( pos == tc->GetLastPosition() )
        return wxString();

    return tc->GetRange(pos, pos + 1);
}
```

Reimplemented in wxStyledTextCtrl, wxComboCtrl, and wxComboBox.

virtual **wxTextPos** **wxTextEntry::GetLastPosition** ( ) const [virtual]

Returns the zero based index of the last position in the text control, which is equal to the number of characters in the control.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

wxPoint **wxTextEntry::GetMargins** ( ) const

Returns the margins used by the control.

The `x` field of the returned point is the horizontal margin and the `y` field is the vertical one.

Remarks

If given margin cannot be accurately determined, its value will be set to -1. On some platforms you cannot obtain valid margin values until you have called SetMargins().

See also

SetMargins()

Since

2.9.1

virtual **wxString** **wxTextEntry::GetRange** (long *from*, long *to*) const [virtual]

Returns the string containing the text starting in the positions *from* and up to *to* in the control.

The positions must have been returned by another wxTextCtrl method. Please note that the positions in a multiline wxTextCtrl do **not** correspond to the indices in the string returned by GetValue() because of the different new line representations (`CR` or `CR` LF) and so this method should be used to obtain the correct results instead of extracting parts of the entire value. It may also be more efficient, especially if the control contains a lot of data.

virtual void **wxTextEntry::GetSelection** (long * *from*, long * *to*) const [virtual]

Gets the current selection span.

If the returned values are equal, there was no selection. Please note that the indices returned may be used with the other wxTextCtrl methods but don't necessarily represent the correct indices into the string returned by GetValue() for multiline controls under Windows (at least,) you should use GetStringSelection() to get the selected text.

Parameters

from	The returned first position.
to	The returned last position.

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (from, to).

Reimplemented in wxStyledTextCtrl, and wxComboBox.

virtual **wxString** **wxTextEntry::GetStringSelection** ( ) const [virtual]

Gets the text currently selected in the control.

If there is no selection, the returned string is empty.

Reimplemented in wxComboBox.

virtual wxString wxTextEntry::GetValue ( ) const [virtual]

Gets the contents of the control.

Notice that for a multiline text control, the lines will be separated by (Unix-style) \n characters, even under Windows where they are separated by a \r\n sequence in the native control.

Reimplemented in wxComboCtrl.

virtual bool wxTextEntry::IsEditable ( ) const [virtual]

Returns true if the controls contents may be edited by user (note that it always can be changed by the program).

In other words, this functions returns true if the control hasn't been put in read-only mode by a previous call to SetEditable().

Reimplemented in wxStyledTextCtrl.

virtual bool wxTextEntry::IsEmpty ( ) const [virtual]

Returns true if the control is currently empty.

This is the same as GetValue().empty() but can be much more efficient for the multiline controls containing big amounts of text.

Since

2.7.1

Reimplemented in wxComboBox, and wxOwnerDrawnComboBox.

virtual void wxTextEntry::Paste ( ) [virtual]

Pastes text from the clipboard to the text item.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual void wxTextEntry::Redo ( ) [virtual]

If there is a redo facility and the last operation can be redone, redoes the last operation.

Does nothing if there is no redo facility.

Reimplemented in wxStyledTextCtrl.

virtual void wxTextEntry::Remove (long *from*, long *to*) [virtual]

Removes the text starting at the first given position up to (but not including) the character at the last position.

This function puts the current insertion point position at *to* as a side effect.

Parameters

from	The first position.
to	The last position.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual void wxTextEntry::Replace (long *from,* long *to,* const wxString & *value*) `[virtual]`

Replaces the text starting at the first position up to (but not including) the character at the last position with the given text.

This function puts the current insertion point position at *to* as a side effect.

Parameters

from	The first position.
to	The last position.
value	The value to replace the existing text with.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual void wxTextEntry::SelectAll ( ) `[virtual]`

Selects all text in the control.

See also

SetSelection()

Reimplemented in wxStyledTextCtrl.

virtual void wxTextEntry::SelectNone ( ) `[virtual]`

Deselects selected text in the control.

Since

2.9.5

Reimplemented in wxStyledTextCtrl.

virtual void wxTextEntry::SetEditable (bool *editable*) `[virtual]`

Makes the text item editable or read-only, overriding the **wxTE_READONLY** flag.

Parameters

editable	If true, the control is editable. If false, the control is read-only.

See also

IsEditable()

Reimplemented in wxStyledTextCtrl.

virtual bool wxTextEntry::SetHint (const wxString & *hint*) `[virtual]`

Sets a hint shown in an empty unfocused text control.

The hints are usually used to indicate to the user what is supposed to be entered into the given entry field, e.g. a common use of them is to show an explanation of what can be entered in a wxSearchCtrl.

The hint is shown (usually greyed out) for an empty control until it gets focus and is shown again if the control loses it and remains empty. It won't be shown once the control has a non-empty value, although it will be shown again if the control contents is cleared. Because of this, it generally only makes sense to use hints with the controls which are initially empty.

Notice that hints are known as *cue banners* under MSW or *placeholder strings* under OS X.

Remarks

Currently implemented natively on Windows (Vista and later only), OS X and GTK+ (3.2 and later).

For the platforms without native hints support, the implementation has several known limitations. Notably, the hint display will not be properly updated if you change wxTextEntry contents programmatically when the hint is displayed using methods other than SetValue() or ChangeValue() or others which use them internally (e.g. Clear()). In other words, currently you should avoid calling methods such as WriteText() or Replace() when using hints and the text control is empty. If you bind to the control's focus and wxEVT_TEXT events, you must call wxEvent::Skip() on them so that the generic implementation works correctly.

Remarks

Hints can be used for single line text controls under all platforms, but only MSW and GTK+ 2 support them for multi-line text controls, they are ignored for them under the other platforms.

Since

2.9.0

Reimplemented in wxComboCtrl.

virtual void wxTextEntry::SetInsertionPoint (long *pos*) [virtual]

Sets the insertion point at the given position.

Parameters

pos	Position to set, in the range from 0 to GetLastPosition() inclusive.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual void wxTextEntry::SetInsertionPointEnd ( ) [virtual]

Sets the insertion point at the end of the text control.

This is equivalent to calling wxTextCtrl::SetInsertionPoint() with wxTextCtrl::GetLastPosition() argument.

Reimplemented in wxComboCtrl.

bool wxTextEntry::SetMargins (const wxPoint & *pt*)

Attempts to set the control margins.

When margins are given as wxPoint, x indicates the left and y the top margin. Use -1 to indicate that an existing value should be used.

Returns

true if setting of all requested margins was successful.

Since

2.9.1

bool wxTextEntry::SetMargins (wxCoord *left,* wxCoord *top* = -1)

Attempts to set the control margins.

When margins are given as wxPoint, x indicates the left and y the top margin. Use -1 to indicate that an existing value should be used.

Returns

true if setting of all requested margins was successful.

Since

2.9.1

virtual void wxTextEntry::SetMaxLength (unsigned long *len*) [virtual]

This function sets the maximum number of characters the user can enter into the control.

In other words, it allows to limit the text value length to *len* not counting the terminating `NUL` character.

If *len* is 0, the previously set max length limit, if any, is discarded and the user may enter as much text as the underlying native text control widget supports (typically at least 32Kb). If the user tries to enter more characters into the text control when it already is filled up to the maximal length, a `wxEVT_TEXT_MAXLEN` event is sent to notify the program about it (giving it the possibility to show an explanatory message, for example) and the extra input is discarded.

Note that in wxGTK this function may only be used with single line text controls.

virtual void wxTextEntry::SetSelection (long *from,* long *to*) [virtual]

Selects the text starting at the first position up to (but not including) the character at the last position.

If both parameters are equal to -1 all text in the control is selected.

Notice that the insertion point will be moved to *from* by this function.

Parameters

from	The first position.
to	The last position.

See also

SelectAll()

Reimplemented in wxStyledTextCtrl, wxComboCtrl, and wxComboBox.

virtual void wxTextEntry::SetValue (const wxString & *value*) [virtual]

Sets the new text control value.

It also marks the control as not-modified which means that IsModified() would return false immediately after the call to SetValue().

The insertion point is set to the start of the control (i.e. position 0) by this function.

Note that, unlike most other functions changing the controls values, this function generates a `wxEVT_TEXT` event. To avoid this you can use ChangeValue() instead.

Parameters

value	The new value to set. It may contain newline characters if the text control is multi-line.

Reimplemented in wxComboCtrl, and wxComboBox.

virtual void wxTextEntry::Undo ( ) [virtual]

If there is an undo facility and the last operation can be undone, undoes the last operation.

Does nothing if there is no undo facility.

Reimplemented in wxStyledTextCtrl, and wxComboCtrl.

virtual void wxTextEntry::WriteText (const wxString & *text*) [virtual]

Writes the text into the text control at the current insertion position.

Parameters

text	Text to write to the text control.

Remarks

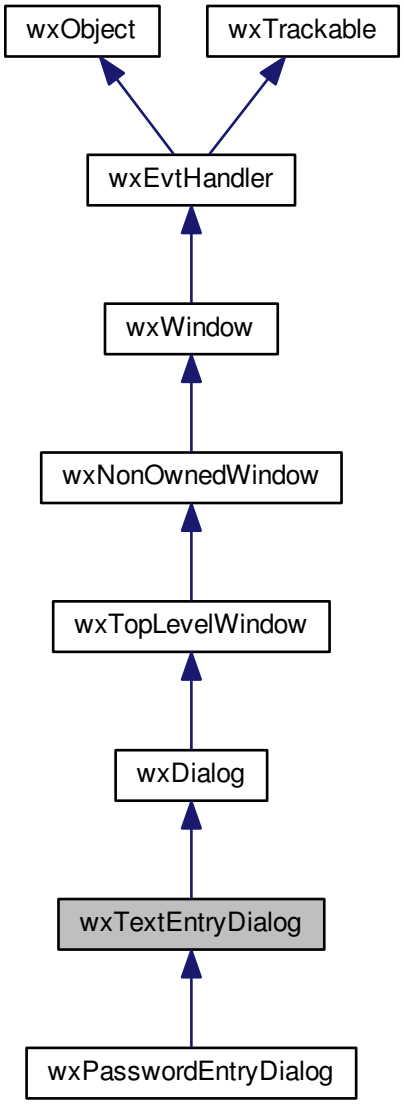
Newlines in the text string are the only control characters allowed, and they will cause appropriate line breaks. See operator<<() and AppendText() for more convenient ways of writing to the window. After the write operation, the insertion point will be at the end of the inserted text, so subsequent write operations will be appended. To append text after the user may have interacted with the control, call wxTextCtrl::SetInsertionPointEnd() before writing.

Reimplemented in wxStyledTextCtrl.

21.777 wxTextEntryDialog Class Reference

```
#include <wx/textdlg.h>
```

Inheritance diagram for wxTextEntryDialog:



21.777.1 Detailed Description

This class represents a dialog that requests a one-line text string from the user.

It is implemented as a generic wxWidgets dialog.

Library: wxCore

Category: Common Dialogs

See also

wxTextEntryDialog Overview

Public Member Functions

- wxTextEntryDialog ()

  Default constructor.
- wxTextEntryDialog (wxWindow *parent, const wxString &message, const wxString &caption=wxGetTextFromUserPromptStr, const wxString &value=wxEmptyString, long style=wxTextEntryDialogStyle, const wxPoint &pos=wxDefaultPosition)

  Constructor.
- bool Create (wxWindow *parent, const wxString &message, const wxString &caption=wxGetTextFromUserPromptStr, const wxString &value=wxEmptyString, long style=wxTextEntryDialogStyle, const wxPoint &pos=wxDefaultPosition)
- virtual ~wxTextEntryDialog ()

  Destructor.
- wxString GetValue () const

  Returns the text that the user has entered if the user has pressed OK, or the original value if the user has pressed Cancel.
- void SetMaxLength (unsigned long len)

  This function sets the maximum number of characters the user can enter into this dialog.
- void SetValue (const wxString &value)

  Sets the default text value.
- int ShowModal ()

  Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

- void SetTextValidator (const wxTextValidator &validator)

  Associate a validator with the text control used by the dialog.
- void SetTextValidator (wxTextValidatorStyle style=wxFILTER_NONE)

  Associate a validator with the text control used by the dialog.

Additional Inherited Members

21.777.2 Constructor & Destructor Documentation

wxTextEntryDialog::wxTextEntryDialog ( )

Default constructor.

Call Create() to really create the dialog later.

Since

2.9.5

wxTextEntryDialog::wxTextEntryDialog (wxWindow * *parent*, const wxString & *message*, const wxString & *caption* = wxGetTextFromUserPromptStr, const wxString & *value* = wxEmptyString, long *style* = wxTextEntryDialogStyle, const wxPoint & *pos* = wxDefaultPosition)

Constructor.

Use ShowModal() to show the dialog.

See Create() method for parameter description.

virtual wxTextEntryDialog::~wxTextEntryDialog ( ) [virtual]

Destructor.

21.777.3 Member Function Documentation

bool wxTextEntryDialog::Create (wxWindow * *parent*, const wxString & *message*, const wxString & *caption* = wxGetTextFromUserPromptStr, const wxString & *value* = wxEmptyString, long *style* = wxTextEntryDialogStyle, const wxPoint & *pos* = wxDefaultPosition)

Parameters

parent	Parent window.
message	Message to show on the dialog.
caption	The caption of the dialog.
value	The default value, which may be the empty string.
style	A dialog style, specifying the buttons (wxOK, wxCANCEL) and an optional wxCENTRE style. Additionally, wxTextCtrl styles (such as `wxTE_PASSWORD` or `wxTE_MULTILINE`) may be specified here.
pos	Dialog position.

Since

2.9.5

wxString wxTextEntryDialog::GetValue ( ) const

Returns the text that the user has entered if the user has pressed OK, or the original value if the user has pressed Cancel.

void wxTextEntryDialog::SetMaxLength (unsigned long *len*)

This function sets the maximum number of characters the user can enter into this dialog.

See also

wxTextEntry::SetMaxLength()

Since

2.9.5

void wxTextEntryDialog::SetTextValidator (const wxTextValidator & *validator*)

Associate a validator with the text control used by the dialog.

These methods can be used to limit the user entry to only some characters, e.g.

```
wxTextEntryDialog dlg(this, ...);
dlg.SetTextValidator(wxFILTER_ALPHA);
if ( dlg.ShowModal() == wxID_OK )
{
    // We can be certain that this string contains letters only.
    wxString value = dlg.GetValue();
}
```

The first overload uses the provided *validator* which can be of a custom class derived from wxTextValidator while the second one creates a wxTextValidator with the specified *style*.

void wxTextEntryDialog::SetTextValidator (wxTextValidatorStyle *style* = wxFILTER_NONE)

Associate a validator with the text control used by the dialog.

These methods can be used to limit the user entry to only some characters, e.g.

```
wxTextEntryDialog dlg(this, ...);
dlg.SetTextValidator(wxFILTER_ALPHA);
if ( dlg.ShowModal() == wxID_OK )
{
    // We can be certain that this string contains letters only.
    wxString value = dlg.GetValue();
}
```

The first overload uses the provided *validator* which can be of a custom class derived from wxTextValidator while the second one creates a wxTextValidator with the specified *style*.

void wxTextEntryDialog::SetValue (const wxString & *value*)

Sets the default text value.

int wxTextEntryDialog::ShowModal ( ) [virtual]

Shows the dialog, returning wxID_OK if the user pressed OK, and wxID_CANCEL otherwise.

Call GetValue() to retrieve the values of the string entered by the user after showing the dialog.

Reimplemented from wxDialog.

21.778 wxTextFile Class Reference

`#include <wx/textfile.h>`

21.778.1 Detailed Description

The wxTextFile is a simple class which allows to work with text files on line by line basis.

It also understands the differences in line termination characters under different platforms and will not do anything bad to files with "non native" line termination sequences - in fact, it can be also used to modify the text files and change the line termination characters from one type (say DOS) to another (say Unix).

One word of warning: the class is not at all optimized for big files and thus it will load the file entirely into memory when opened. Of course, you should not work in this way with large files (as an estimation, anything over 1 Megabyte is surely too big for this class). On the other hand, it is not a serious limitation for small files like configuration files or program sources which are well handled by wxTextFile.

The typical things you may do with wxTextFile in order are:

- Create and open it: this is done with either wxTextFile::Create or wxTextFile::Open function which opens the file (name may be specified either as the argument to these functions or in the constructor), reads its contents in memory (in the case of wxTextFile::Open()) and closes it.
- Work with the lines in the file: this may be done either with "direct access" functions like wxTextFile::GetLineCount and wxTextFile::GetLine (*operator*[] does exactly the same but looks more like array addressing) or with "sequential access" functions which include wxTextFile::GetFirstLine, wxTextFile::GetNextLine and also wxTextFile::GetLastLine, wxTextFile::GetPrevLine. For the sequential access functions the current line number is maintained: it is returned by wxTextFile::GetCurrentLine and may be changed with wxTextFile::GoToLine.
- Add/remove lines to the file: wxTextFile::AddLine and wxTextFile::InsertLine add new lines while wxTextFile::RemoveLine deletes the existing ones. wxTextFile::Clear resets the file to empty.

- Save your changes: notice that the changes you make to the file will **not** be saved automatically; calling wxTextFile::Close or doing nothing discards them! To save the changes you must explicitly call wxTextFile::Write - here, you may also change the line termination type if you wish.

Library: wxBase

Category: File Handling

See also

wxFile

Public Member Functions

- wxTextFile ()

  Default constructor, use Create() or Open() with a file name parameter to initialize the object.
- wxTextFile (const wxString &strFile)

  Constructor does not load the file into memory, use Open() to do it.
- virtual ~wxTextFile ()

  Destructor does nothing.
- void AddLine (const wxString &str, wxTextFileType type=typeDefault)

  Adds a line to the end of file.
- void Clear ()

  Delete all lines from the file, set current line number to 0.
- bool Close ()

  *Closes the file and frees memory, **"losing all changes"**.*
- bool Create ()

  Creates the file with the name which was given in the wxTextFile(const wxString&) constructor.
- bool Create (const wxString &strFile)

  Creates the file with the given name.
- bool Eof () const

  Returns true if the current line is the last one.
- bool Exists () const

  Return true if file exists - the name of the file should have been specified in the constructor before calling Exists().
- size_t GetCurrentLine () const

  Returns the current line: it has meaning only when you're using GetFirstLine()/GetNextLine() functions, it doesn't get updated when you're using "direct access" functions like GetLine().
- wxString & GetFirstLine ()

  This method together with GetNextLine() allows more "iterator-like" traversal of the list of lines, i.e.
- wxString & GetLastLine ()

  Gets the last line of the file.
- size_t GetLineCount () const

  Get the number of lines in the file.
- wxTextFileType GetLineType (size_t n) const

  Get the type of the line (see also wxTextFile::GetEOL).
- const wxString & GetName () const

  Get the name of the file.
- wxString & GetNextLine ()

  Gets the next line (see GetFirstLine() for the example).
- wxString & GetPrevLine ()

Gets the previous line in the file.

- void GoToLine (size_t n)

  Changes the value returned by GetCurrentLine() and used by GetFirstLine() and GetNextLine().
- wxTextFileType GuessType () const

  Guess the type of file (which is supposed to be opened).
- void InsertLine (const wxString &str, size_t n, wxTextFileType type=typeDefault)

  Insert a line before the line number n.
- bool IsOpened () const

  Returns true if the file is currently opened.
- bool Open (const wxMBConv &conv=wxConvAuto())

  Opens the file with the name which was given in the wxTextFile(const wxString&) constructor and also loads file in memory on success.
- bool Open (const wxString &strFile, const wxMBConv &conv=wxConvAuto())

  Opens the file with the given name and also loads file in memory on success.
- void RemoveLine (size_t n)

  Delete line number n from the file.
- bool Write (wxTextFileType typeNew=wxTextFileType_None, const wxMBConv &conv=wxConvAuto())

  Change the file on disk.
- wxString & operator[] (size_t n) const

  The same as GetLine().

- wxString & GetLine (size_t n)

  Retrieves the line number n from the file.
- const wxString & GetLine (size_t n) const

  Retrieves the line number n from the file.

Static Public Member Functions

- static const wxChar * GetEOL (wxTextFileType type=typeDefault)

  Get the line termination string corresponding to given constant.

Static Public Attributes

- static const wxTextFileType typeDefault

  Default type for current platform determined at compile time.

21.778.2 Constructor & Destructor Documentation

wxTextFile::wxTextFile ( )

Default constructor, use Create() or Open() with a file name parameter to initialize the object.

wxTextFile::wxTextFile (const wxString & *strFile*)

Constructor does not load the file into memory, use Open() to do it.

virtual wxTextFile::~wxTextFile ( ) `[virtual]`

Destructor does nothing.

21.778.3 Member Function Documentation

void wxTextFile::AddLine (const wxString & *str,* wxTextFileType *type* = typeDefault)

Adds a line to the end of file.

void wxTextFile::Clear ( )

Delete all lines from the file, set current line number to 0.

bool wxTextFile::Close ( )

Closes the file and frees memory, **"losing all changes"**.

Use Write() if you want to save them.

bool wxTextFile::Create ( )

Creates the file with the name which was given in the wxTextFile(const wxString&) constructor.

The array of file lines is initially empty.

It will fail if the file already exists, Open() should be used in this case.

bool wxTextFile::Create (const wxString & *strFile*)

Creates the file with the given name.

The array of file lines is initially empty.

It will fail if the file already exists, Open() should be used in this case.

bool wxTextFile::Eof ( ) const

Returns true if the current line is the last one.

bool wxTextFile::Exists ( ) const

Return true if file exists - the name of the file should have been specified in the constructor before calling Exists().

size_t wxTextFile::GetCurrentLine ( ) const

Returns the current line: it has meaning only when you're using GetFirstLine()/GetNextLine() functions, it doesn't get updated when you're using "direct access" functions like GetLine().

GetFirstLine() and GetLastLine() also change the value of the current line, as well as GoToLine().

static const wxChar* wxTextFile::GetEOL (wxTextFileType *type* = typeDefault) [static]

Get the line termination string corresponding to given constant.

typeDefault is the value defined during the compilation and corresponds to the native format of the platform, i.e. it will be `wxTextFileType_Dos` under Windows and `wxTextFileType_Unix` under Unix (including Mac OS X, the value `wxTextFileType_Mac` was only used for classic Mac OS versions).

wxString& wxTextFile::GetFirstLine ( )

This method together with GetNextLine() allows more "iterator-like" traversal of the list of lines, i.e.

you may write something like:

```
wxTextFile file;
...
for ( str = file.GetFirstLine(); !file.Eof(); str = file.
      GetNextLine() )
{
    // do something with the current line in str
}
// do something with the last line in str
```

wxString& wxTextFile::GetLastLine ( )

Gets the last line of the file.

Together with GetPrevLine() it allows to enumerate the lines in the file from the end to the beginning like this:

```
wxTextFile file;
...
for ( str = file.GetLastLine();
    file.GetCurrentLine() > 0;
    str = file.GetPrevLine() )
{
    // do something with the current line in str
}
// do something with the first line in str
```

wxString& wxTextFile::GetLine (size_t *n*)

Retrieves the line number *n* from the file.

The returned line may be modified when non-const method is used but you shouldn't add line terminator at the end – this will be done by wxTextFile itself.

const wxString& wxTextFile::GetLine (size_t *n*) const

Retrieves the line number *n* from the file.

The returned line may be modified when non-const method is used but you shouldn't add line terminator at the end – this will be done by wxTextFile itself.

size_t wxTextFile::GetLineCount ( ) const

Get the number of lines in the file.

wxTextFileType wxTextFile::GetLineType (size_t *n*) const

Get the type of the line (see also wxTextFile::GetEOL).

const wxString& wxTextFile::GetName ( ) const

Get the name of the file.

wxString& wxTextFile::GetNextLine ( )

Gets the next line (see GetFirstLine() for the example).

wxString& wxTextFile::GetPrevLine ( )

Gets the previous line in the file.

void wxTextFile::GoToLine (size_t *n*)

Changes the value returned by GetCurrentLine() and used by GetFirstLine() and GetNextLine().

wxTextFileType wxTextFile::GuessType ( ) const

Guess the type of file (which is supposed to be opened).

If sufficiently many lines of the file are in DOS/Unix/Mac format, the corresponding value will be returned. If the detection mechanism fails `wxTextFileType_None` is returned.

void wxTextFile::InsertLine (const wxString & *str,* size_t *n,* wxTextFileType *type* = typeDefault)

Insert a line before the line number *n*.

bool wxTextFile::IsOpened ( ) const

Returns true if the file is currently opened.

bool wxTextFile::Open (const wxMBConv & *conv* = wxConvAuto())

Opens the file with the name which was given in the wxTextFile(const wxString&) constructor and also loads file in memory on success.

It will fail if the file does not exist, Create() should be used in this case.

The *conv* argument is only meaningful in Unicode build of wxWidgets when it is used to convert the file to wide character representation.

bool wxTextFile::Open (const wxString & *strFile,* const wxMBConv & *conv* = wxConvAuto())

Opens the file with the given name and also loads file in memory on success.

It will fail if the file does not exist, Create() should be used in this case.

The *conv* argument is only meaningful in Unicode build of wxWidgets when it is used to convert the file to wide character representation.

wxString& wxTextFile::operator[] (size_t *n*) const

The same as GetLine().

void wxTextFile::RemoveLine (size_t *n*)

Delete line number *n* from the file.

bool wxTextFile::Write (wxTextFileType *typeNew* = wxTextFileType_None, const wxMBConv & *conv* = wxConvAuto())

Change the file on disk.

The *typeNew* parameter allows you to change the file format (default argument means "don't change type") and may be used to convert, for example, DOS files to Unix.

The *conv* argument is only meaningful in Unicode build of wxWidgets when it is used to convert all lines to multibyte representation before writing them to physical file.

Returns

true if operation succeeded, false if it failed.

21.778.4 Member Data Documentation

const wxTextFileType wxTextFile::typeDefault `[static]`

Default type for current platform determined at compile time.

21.779 wxTextInputStream Class Reference

`#include <wx/txtstrm.h>`

21.779.1 Detailed Description

This class provides functions that reads text data using an input stream, allowing you to read text, floats, and integers.

The wxTextInputStream correctly reads text files (or streams) in DOS, Macintosh and Unix formats and reports a single newline char as a line ending.

wxTextInputStream::operator>>() is overloaded and you can use this class like a standard C++ iostream. Note, however, that the arguments are the fixed size types wxUint32, wxInt32 etc and on a typical 32-bit computer, none of these match to the "long" type (wxInt32 is defined as int on 32-bit architectures) so that you cannot use long. To avoid problems (here and elsewhere), make use of wxInt32, wxUint32 and similar types.

If you're scanning through a file using wxTextInputStream, you should check for `EOF` **before** reading the next item (word / number), because otherwise the last item may get lost. You should however be prepared to receive an empty item (empty string / zero number) at the end of file, especially on Windows systems. This is unavoidable because most (but not all) files end with whitespace (i.e. usually a newline).

For example:

```
wxFileInputStream input( "mytext.txt" );
wxTextInputStream text( input );
wxUint8 i1;
float f2;
wxString line;

text >> i1;       // read a 8 bit integer.
text >> i1 >> f2; // read a 8 bit integer followed by float.
text >> line;     // read a text line
```

Library: wxBase

Category: Streams

See also

wxTextOutputStream

Public Member Functions

- wxTextInputStream (wxInputStream &stream, const wxString &sep=" \t", const wxMBConv &conv=wxConvAuto())

  Constructs a text stream associated to the given input stream.
- ~wxTextInputStream ()

  Destructor.
- const wxInputStream & GetInputStream () const

  Returns a pointer to the underlying input stream object.
- wxChar GetChar ()

  Reads a character, returns 0 if there are no more characters in the stream.
- wxUint16 Read16 (int base=10)

  Reads a unsigned 16 bit integer from the stream.
- wxInt16 Read16S (int base=10)

  Reads a signed 16 bit integer from the stream.
- wxUint32 Read32 (int base=10)

  Reads a 32 bit unsigned integer from the stream.
- wxInt32 Read32S (int base=10)

  Reads a 32 bit signed integer from the stream.
- wxUint64 Read64 (int base=10)

  Reads a 64 bit unsigned integer from the stream.
- wxInt64 Read64S (int base=10)

  Reads a 64 bit signed integer from the stream.
- wxUint8 Read8 (int base=10)

  Reads a single unsigned byte from the stream, given in base base.
- wxInt8 Read8S (int base=10)

  Reads a single signed byte from the stream.
- double ReadDouble ()

  Reads a double (IEEE encoded) from the stream.
- wxString ReadLine ()

  Reads a line from the input stream and returns it (without the end of line character).
- wxString ReadString ()
- wxString ReadWord ()

  Reads a word (a sequence of characters until the next separator) from the input stream.
- void SetStringSeparators (const wxString &sep)

  Sets the characters which are used to define the word boundaries in ReadWord().

21.779.2 Constructor & Destructor Documentation

wxTextInputStream::wxTextInputStream (wxInputStream & *stream*, const wxString & *sep* = " \t", const wxMBConv & *conv* = wxConvAuto())

Constructs a text stream associated to the given input stream.

Parameters

stream	The underlying input stream.
sep	The initial string separator characters.
conv	**In Unicode build only:** The encoding converter used to convert the bytes in the underlying input stream to characters.

wxTextInputStream::~wxTextInputStream ( )

Destructor.

21.779.3 Member Function Documentation

wxChar wxTextInputStream::GetChar ( )

Reads a character, returns 0 if there are no more characters in the stream.

const wxInputStream& wxTextInputStream::GetInputStream ( ) const

Returns a pointer to the underlying input stream object.

Since

2.9.2

wxUint16 wxTextInputStream::Read16 (int *base* = 10)

Reads a unsigned 16 bit integer from the stream.

See Read8() for the description of the *base* parameter.

wxInt16 wxTextInputStream::Read16S (int *base* = 10)

Reads a signed 16 bit integer from the stream.

See Read8() for the description of the *base* parameter.

wxUint32 wxTextInputStream::Read32 (int *base* = 10)

Reads a 32 bit unsigned integer from the stream.

See Read8() for the description of the *base* parameter.

wxInt32 wxTextInputStream::Read32S (int *base* = 10)

Reads a 32 bit signed integer from the stream.

See Read8() for the description of the *base* parameter.

wxUint64 wxTextInputStream::Read64 (int *base* = 10)

Reads a 64 bit unsigned integer from the stream.

See Read8() for the description of the *base* parameter.

Since

3.1.0

wxInt64 wxTextInputStream::Read64S (int *base* = `10`)

Reads a 64 bit signed integer from the stream.

See Read8() for the description of the *base* parameter.

Since

3.1.0

wxUint8 wxTextInputStream::Read8 (int *base* = `10`)

Reads a single unsigned byte from the stream, given in base *base*.

The value of *base* must be comprised between 2 and 36, inclusive, or be a special value 0 which means that the usual rules of C numbers are applied: if the number starts with `0x` it is considered to be in base 16, if it starts with 0 - in base 8 and in base 10 otherwise. Note that you may not want to specify the base 0 if you are parsing the numbers which may have leading zeroes as they can yield unexpected (to the user not familiar with C) results.

wxInt8 wxTextInputStream::Read8S (int *base* = `10`)

Reads a single signed byte from the stream.

See Read8() for the description of the *base* parameter.

double wxTextInputStream::ReadDouble ( )

Reads a double (IEEE encoded) from the stream.

wxString wxTextInputStream::ReadLine ( )

Reads a line from the input stream and returns it (without the end of line character).

wxString wxTextInputStream::ReadString ( )

Deprecated Use ReadLine() or ReadWord() instead.

Same as ReadLine().

wxString wxTextInputStream::ReadWord ( )

Reads a word (a sequence of characters until the next separator) from the input stream.

See also

SetStringSeparators()

void wxTextInputStream::SetStringSeparators (const wxString & *sep*)

Sets the characters which are used to define the word boundaries in ReadWord().

The default separators are the `space` and `TAB` characters.

21.780 wxTextOutputStream Class Reference

```
#include <wx/txtstrm.h>
```

21.780.1 Detailed Description

This class provides functions that write text data using an output stream, allowing you to write text, floats, and integers.

You can also simulate the C++ `std::cout` class:

```
wxFFileOutputStream output( stderr );
wxTextOutputStream cout( output );

cout << "This is a text line" << endl;
cout << 1234;
cout << 1.23456;
```

The wxTextOutputStream writes text files (or streams) on DOS, Macintosh and Unix in their native formats (concerning the line ending).

Library: wxBase

Category: Streams

See also

wxTextInputStream

Public Member Functions

- wxTextOutputStream (wxOutputStream &stream, wxEOL mode=wxEOL_NATIVE, const wxMBConv &conv=wxConvAuto())

  Constructs a text stream object associated to the given output stream.
- virtual ~wxTextOutputStream ()

  Destroys the wxTextOutputStream object.
- void Flush ()

  Flushes the stream.
- const wxOutputStream & GetOutputStream () const

  Returns a pointer to the underlying output stream object.
- wxEOL GetMode ()

  Returns the end-of-line mode.
- wxTextOutputStream & PutChar (wxChar c)

  Writes a character to the stream.
- void SetMode (wxEOL mode=wxEOL_NATIVE)

  Set the end-of-line mode.
- void Write64 (wxUint64 i64)

  Writes the 64 bit integer i64 to the stream.
- void Write32 (wxUint32 i32)

  Writes the 32 bit integer i32 to the stream.
- void Write16 (wxUint16 i16)

  Writes the 16 bit integer i16 to the stream.
- void Write8 (wxUint8 i8)

Writes the single byte i8 to the stream.

- virtual void WriteDouble (double f)

  Writes the double f to the stream using the IEEE format.

- virtual void WriteString (const wxString &string)

  Writes string as a line.

21.780.2 Constructor & Destructor Documentation

wxTextOutputStream::wxTextOutputStream (wxOutputStream & *stream*, wxEOL *mode* = wxEOL_NATIVE, const wxMBConv & *conv* = wxConvAuto ())

Constructs a text stream object associated to the given output stream.

Parameters

stream	The output stream.
mode	The end-of-line mode. One of wxEOL_NATIVE, wxEOL_DOS, wxEOL_MAC and wxEOL_UNIX.
conv	**In Unicode build only:** The object used to convert Unicode text into ASCII characters written to the output stream.

virtual wxTextOutputStream::~wxTextOutputStream ( ) [virtual]

Destroys the wxTextOutputStream object.

Also calls Flush().

21.780.3 Member Function Documentation

void wxTextOutputStream::Flush ( )

Flushes the stream.

This method should be called when using stateful encodings (currently the only example of such encoding in wxWidgets is wxMBConvUTF7) to write the end of the encoded data to the stream.

Since

2.9.0

wxEOL wxTextOutputStream::GetMode ( )

Returns the end-of-line mode.

One of wxEOL_DOS, wxEOL_MAC and wxEOL_UNIX.

const wxOutputStream& wxTextOutputStream::GetOutputStream ( ) const

Returns a pointer to the underlying output stream object.

Since

2.9.2

wxTextOutputStream& wxTextOutputStream::PutChar (wxChar *c*)

Writes a character to the stream.

void wxTextOutputStream::SetMode (wxEOL *mode* = wxEOL_NATIVE)

Set the end-of-line mode.

One of wxEOL_NATIVE, wxEOL_DOS, wxEOL_MAC and wxEOL_UNIX.

void wxTextOutputStream::Write16 (wxUint16 *i16*)

Writes the 16 bit integer *i16* to the stream.

void wxTextOutputStream::Write32 (wxUint32 *i32*)

Writes the 32 bit integer *i32* to the stream.

void wxTextOutputStream::Write64 (wxUint64 *i64*)

Writes the 64 bit integer *i64* to the stream.

Since

3.1.0

void wxTextOutputStream::Write8 (wxUint8 *i8*)

Writes the single byte *i8* to the stream.

virtual void wxTextOutputStream::WriteDouble (double *f*) [virtual]

Writes the double *f* to the stream using the IEEE format.

virtual void wxTextOutputStream::WriteString (const wxString & *string*) [virtual]

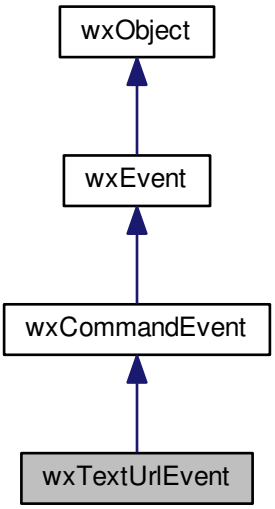
Writes *string* as a line.

Depending on the end-of-line mode the end of line ('\n') characters in the string are converted to the correct line ending terminator.

21.781 wxTextUrlEvent Class Reference

```
#include <wx/textctrl.h>
```

Inheritance diagram for wxTextUrlEvent:



Public Member Functions

- wxTextUrlEvent (int winid, const wxMouseEvent &evtMouse, long start, long end)
- wxTextUrlEvent (const wxTextUrlEvent &event)
- const wxMouseEvent & GetMouseEvent () const
- long GetURLStart () const
- long GetURLEnd () const
- virtual wxEvent * Clone () const

  Returns a copy of the event.

Additional Inherited Members

21.781.1 Constructor & Destructor Documentation

wxTextUrlEvent::wxTextUrlEvent (int *winid,* const wxMouseEvent & *evtMouse,* long *start,* long *end*)

wxTextUrlEvent::wxTextUrlEvent (const wxTextUrlEvent & *event*)

21.781.2 Member Function Documentation

virtual wxEvent* wxTextUrlEvent::Clone ( ) const `[virtual]`

Returns a copy of the event.

Any event that is posted to the wxWidgets event system for later action (via wxEvtHandler::AddPendingEvent, wxEvtHandler::QueueEvent or wxPostEvent()) must implement this method.

All wxWidgets events fully implement this method, but any derived events implemented by the user should also implement this method just in case they (or some event derived from them) are ever posted.

All wxWidgets events implement a copy constructor, so the easiest way of implementing the Clone function is to implement a copy constructor for a new event (call it MyEvent) and then define the Clone function like this:

```
wxEvent *Clone() const { return new MyEvent(*this); }
```

Implements wxEvent.

const wxMouseEvent& wxTextUrlEvent::GetMouseEvent ( ) const

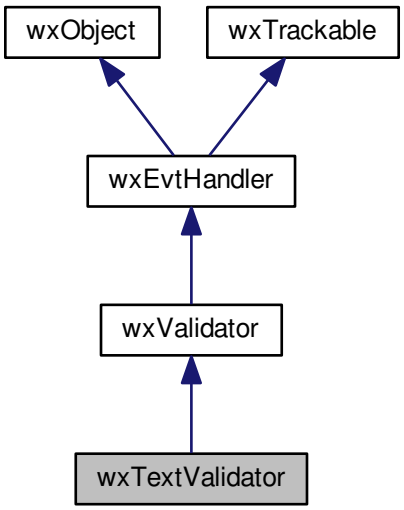
long wxTextUrlEvent::GetURLEnd ( ) const

long wxTextUrlEvent::GetURLStart ( ) const

21.782 wxTextValidator Class Reference

`#include <wx/valtext.h>`

Inheritance diagram for wxTextValidator:



21.782.1 Detailed Description

wxTextValidator validates text controls, providing a variety of filtering behaviours.

For more information, please see wxValidator Overview.

Library: wxCore

Category: Validators

See also

wxValidator Overview, wxValidator, wxGenericValidator, wxIntegerValidator, wxFloatingPointValidator

Public Member Functions

- wxTextValidator (const wxTextValidator &validator)

  Default constructor.
- wxTextValidator (long style=wxFILTER_NONE, wxString *valPtr=NULL)

  Constructor taking a style and optional pointer to a wxString variable.
- virtual wxObject * Clone () const

  Clones the text validator using the copy constructor.
- wxArrayString & GetExcludes ()

  Returns a reference to the exclude list (the list of invalid values).
- wxArrayString & GetIncludes ()

  Returns a reference to the include list (the list of valid values).
- long GetStyle () const

  Returns the validator style.
- bool HasFlag (wxTextValidatorStyle style) const

  Returns true if the given style bit is set in the current style.
- void OnChar (wxKeyEvent &event)

  Receives character input from the window and filters it according to the current validator style.
- void SetExcludes (const wxArrayString &stringList)

  Sets the exclude list (invalid values for the user input).
- void SetCharExcludes (const wxString &chars)

  Breaks the given chars strings in single characters and sets the internal wxArrayString used to store the "excluded" characters (see SetExcludes()).
- void SetIncludes (const wxArrayString &stringList)

  Sets the include list (valid values for the user input).
- void SetCharIncludes (const wxString &chars)

  Breaks the given chars strings in single characters and sets the internal wxArrayString used to store the "included" characters (see SetIncludes()).
- void SetStyle (long style)

  Sets the validator style which must be a combination of one or more of the wxTextValidatorStyle values.
- virtual bool TransferFromWindow ()

  Transfers the value in the text control to the string.
- virtual bool TransferToWindow ()

  Transfers the string value to the text control.
- virtual bool Validate (wxWindow *parent)

  Validates the window contents against the include or exclude lists, depending on the validator style.

Protected Member Functions

- bool ContainsOnlyIncludedCharacters (const wxString &val) const

  Returns true if all the characters of the given val string are present in the include list (set by SetIncludes() or SetCharIncludes()).
- bool ContainsExcludedCharacters (const wxString &val) const

  Returns true if at least one character of the given val string is present in the exclude list (set by SetExcludes() or SetCharExcludes()).
- virtual wxString IsValid (const wxString &val) const

  Returns the error message if the contents of val are invalid or the empty string if val is valid.

Additional Inherited Members

21.782.2 Constructor & Destructor Documentation

wxTextValidator::wxTextValidator (const wxTextValidator & *validator*)

Default constructor.

wxTextValidator::wxTextValidator (long *style* = wxFILTER_NONE, wxString * *valPtr* = NULL)

Constructor taking a style and optional pointer to a wxString variable.

Parameters

style	One or more of the wxTextValidatorStyle styles. See SetStyle().
valPtr	A pointer to a wxString variable that contains the value. This variable should have a lifetime equal to or longer than the validator lifetime (which is usually determined by the lifetime of the window).

21.782.3 Member Function Documentation

virtual wxObject* wxTextValidator::Clone ( ) const [virtual]

Clones the text validator using the copy constructor.

Reimplemented from wxValidator.

bool wxTextValidator::ContainsExcludedCharacters (const wxString & *val*) const [protected]

Returns true if at least one character of the given *val* string is present in the exclude list (set by SetExcludes() or SetCharExcludes()).

bool wxTextValidator::ContainsOnlyIncludedCharacters (const wxString & *val*) const [protected]

Returns true if all the characters of the given *val* string are present in the include list (set by SetIncludes() or SetCharIncludes()).

wxArrayString& wxTextValidator::GetExcludes ( )

Returns a reference to the exclude list (the list of invalid values).

wxArrayString& wxTextValidator::GetIncludes ( )

Returns a reference to the include list (the list of valid values).

long wxTextValidator::GetStyle ( ) const

Returns the validator style.

See also

HasFlag()

bool wxTextValidator::HasFlag (wxTextValidatorStyle *style*) const

Returns true if the given *style* bit is set in the current style.

virtual wxString wxTextValidator::IsValid (const wxString & *val*) const `[protected], [virtual]`

Returns the error message if the contents of *val* are invalid or the empty string if *val* is valid.

void wxTextValidator::OnChar (wxKeyEvent & *event*)

Receives character input from the window and filters it according to the current validator style.

void wxTextValidator::SetCharExcludes (const wxString & *chars*)

Breaks the given *chars* strings in single characters and sets the internal wxArrayString used to store the "excluded" characters (see SetExcludes()).

This function is mostly useful when `wxFILTER_EXCLUDE_CHAR_LIST` was used.

void wxTextValidator::SetCharIncludes (const wxString & *chars*)

Breaks the given *chars* strings in single characters and sets the internal wxArrayString used to store the "included" characters (see SetIncludes()).

This function is mostly useful when `wxFILTER_INCLUDE_CHAR_LIST` was used.

void wxTextValidator::SetExcludes (const wxArrayString & *stringList*)

Sets the exclude list (invalid values for the user input).

void wxTextValidator::SetIncludes (const wxArrayString & *stringList*)

Sets the include list (valid values for the user input).

void wxTextValidator::SetStyle (long *style*)

Sets the validator style which must be a combination of one or more of the wxTextValidatorStyle values.

Note that not all possible combinations make sense! Also note that the order in which the checks are performed is important, in case you specify more than a single style. wxTextValidator will perform the checks in the same definition order used in the wxTextValidatorStyle enumeration.

virtual bool wxTextValidator::TransferFromWindow ( ) `[virtual]`

Transfers the value in the text control to the string.

Reimplemented from wxValidator.

virtual bool wxTextValidator::TransferToWindow ( ) `[virtual]`

Transfers the string value to the text control.

Reimplemented from wxValidator.

virtual bool wxTextValidator::Validate (**wxWindow** * *parent*) [virtual]

Validates the window contents against the include or exclude lists, depending on the validator style.

Reimplemented from wxValidator.

21.783 wxTextWrapper Class Reference

`#include <wx/textwrapper.h>`

21.783.1 Detailed Description

Helps wrap lines of text to given width.

This is a generic purpose class which can be used to wrap lines of text to the specified width. It doesn't do anything by itself but simply calls its virtual OnOutputLine() and OnNewLine() methods for each wrapped line of text, you need to implement them in your derived class to actually do something useful.

Here is an example function using this class which inserts hard line breaks into a string of text at the positions where it would be wrapped:

```
wxString WrapText(wxWindow *win, const wxString& text, int widthMax)
{
    class HardBreakWrapper : public wxTextWrapper
    {
    public:
        HardBreakWrapper(wxWindow *win, const wxString& text, int widthMax)
        {
            Wrap(win, text, widthMax);
        }

        wxString const& GetWrapped() const { return m_wrapped; }

    protected:
        virtual void OnOutputLine(const wxString& line)
        {
            m_wrapped += line;
        }

        virtual void OnNewLine()
        {
            m_wrapped += '\n';
        }

    private:
        wxString m_wrapped;
    };

    HardBreakWrapper wrapper(win, text, widthMax);
    return wrapper.GetWrapped();
}
```

Library: None; this class implementation is entirely header-based.

Category: Graphics Device Interface (GDI)

Public Member Functions

- wxTextWrapper ()

  Trivial default constructor.

- void Wrap (wxWindow *win, const wxString &text, int widthMax)

  Wrap the given text.

Protected Member Functions

- virtual void OnOutputLine (const wxString &line)=0

  Called by Wrap() for each wrapped line of text.
- virtual void OnNewLine ()

  Called at the start of each subsequent line of text by Wrap().

21.783.2 Constructor & Destructor Documentation

wxTextWrapper::wxTextWrapper ( )

Trivial default constructor.

21.783.3 Member Function Documentation

virtual void wxTextWrapper::OnNewLine ( ) `[protected],[virtual]`

Called at the start of each subsequent line of text by Wrap().

This method may not be called at all if the entire text passed to Wrap() fits into the specified width.

virtual void wxTextWrapper::OnOutputLine (const wxString & *line*) `[protected],[pure virtual]`

Called by Wrap() for each wrapped line of text.

This method will always be called at least once by Wrap(). Notice that *line* may be empty if the text passed to Wrap() was empty itself.

void wxTextWrapper::Wrap (wxWindow * *win,* const wxString & *text,* int *widthMax*)

Wrap the given text.

This method will call OnOutputLine() for every line of wrapped text and OnNewLine() before the beginning of every new line after the first one (so it might be never called at all if the width of entire *text* is less than *widthMax*).

Parameters

win	A non-NULL window used for measuring the text extents.
text	The text to wrap.
widthMax	Maximal width of each line of text or `-1` to disable wrapping.

21.784 wxThread Class Reference

`#include <wx/thread.h>`

21.784.1 Detailed Description

A thread is basically a path of execution through a program.

Threads are sometimes called *light-weight* processes, but the fundamental difference between threads and processes is that memory spaces of different processes are separated while all threads share the same address space.

While it makes it much easier to share common data between several threads, it also makes it much easier to shoot oneself in the foot, so careful use of synchronization objects such as mutexes (see wxMutex) or critical sections (see

wxCriticalSection) is recommended. In addition, don't create global thread objects because they allocate memory in their constructor, which will cause problems for the memory checking system.

21.784.2 Types of wxThreads

There are two types of threads in wxWidgets: *detached* and *joinable*, modeled after the POSIX thread API. This is different from the Win32 API where all threads are joinable.

By default wxThreads in wxWidgets use the **detached** behaviour. Detached threads delete themselves once they have completed, either by themselves when they complete processing or through a call to Delete(), and thus **must** be created on the heap (through the new operator, for example).

Typically you'll want to store the instances of the detached wxThreads you allocate, so that you can call functions on them. Because of their nature however you'll need to always use a critical section when accessing them:

```
// declare a new type of event, to be used by our MyThread class:
wxDECLARE_EVENT(wxEVT_COMMAND_MYTHREAD_COMPLETED, wxThreadEvent);
wxDECLARE_EVENT(wxEVT_COMMAND_MYTHREAD_UPDATE, wxThreadEvent);
class MyFrame;

class MyThread : public wxThread
{
public:
    MyThread(MyFrame *handler)
        : wxThread(wxTHREAD_DETACHED)
        { m_pHandler = handler }
    ~MyThread();

protected:
    virtual ExitCode Entry();
    MyFrame *m_pHandler;
};

class MyFrame : public wxFrame
{
public:
    ...
    ~MyFrame()
    {
        // it's better to do any thread cleanup in the OnClose()
        // event handler, rather than in the destructor.
        // This is because the event loop for a top-level window is not
        // active anymore when its destructor is called and if the thread
        // sends events when ending, they won't be processed unless
        // you ended the thread from OnClose.
        // See @ref overview_windowdeletion for more info.
    }
    ...
    void DoStartThread();
    void DoPauseThread();

    // a resume routine would be nearly identic to DoPauseThread()
    void DoResumeThread() { ... }

    void OnThreadUpdate(wxThreadEvent&);
    void OnThreadCompletion(wxThreadEvent&);
    void OnClose(wxCloseEvent&);

protected:
    MyThread *m_pThread;
    wxCriticalSection m_pThreadCS;    // protects the m_pThread pointer

    friend class MyThread;            // allow it to access our m_pThread

    wxDECLARE_EVENT_TABLE();
};

wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_CLOSE(MyFrame::OnClose)
    EVT_MENU(Minimal_Start,  MyFrame::DoStartThread)
    EVT_COMMAND(wxID_ANY, wxEVT_COMMAND_MYTHREAD_UPDATE, MyFrame::OnThreadUpdate)
    EVT_COMMAND(wxID_ANY, wxEVT_COMMAND_MYTHREAD_COMPLETED, MyFrame::OnThreadCompletion)
wxEND_EVENT_TABLE()

wxDEFINE_EVENT(wxEVT_COMMAND_MYTHREAD_COMPLETED, wxThreadEvent);
wxDEFINE_EVENT(wxEVT_COMMAND_MYTHREAD_UPDATE, wxThreadEvent);

void MyFrame::DoStartThread()
{
    m_pThread = new MyThread(this);
```

```
    if ( m_pThread->Run() != wxTHREAD_NO_ERROR )
    {
        wxLogError("Can't create the thread!");
        delete m_pThread;
        m_pThread = NULL;
    }

    // after the call to wxThread::Run(), the m_pThread pointer is "unsafe":
    // at any moment the thread may cease to exist (because it completes its work).
    // To avoid dangling pointers OnThreadExit() will set m_pThread
    // to NULL when the thread dies.
}

wxThread::ExitCode MyThread::Entry()
{
    while (!TestDestroy())
    {
        // ... do a bit of work...

        wxQueueEvent(m_pHandler, new wxThreadEvent(wxEVT_COMMAND_MYTHREAD_UPDATE));
    }

    // signal the event handler that this thread is going to be destroyed
    // NOTE: here we assume that using the m_pHandler pointer is safe,
    //       (in this case this is assured by the MyFrame destructor)
    wxQueueEvent(m_pHandler, new wxThreadEvent(wxEVT_COMMAND_MYTHREAD_COMPLETED));

    return (wxThread::ExitCode)0;     // success
}

MyThread::~MyThread()
{
    wxCriticalSectionLocker enter(m_pHandler->m_pThreadCS);

    // the thread is being destroyed; make sure not to leave dangling pointers around
    m_pHandler->m_pThread = NULL;
}

void MyFrame::OnThreadCompletion(wxThreadEvent&)
{
    wxMessageOutputDebug().Printf("MYFRAME: MyThread exited!\n");
}

void MyFrame::OnThreadUpdate(wxThreadEvent&)
{
    wxMessageOutputDebug().Printf("MYFRAME: MyThread update...\n");
}

void MyFrame::DoPauseThread()
{
    // anytime we access the m_pThread pointer we must ensure that it won't
    // be modified in the meanwhile; since only a single thread may be
    // inside a given critical section at a given time, the following code
    // is safe:
    wxCriticalSectionLocker enter(m_pThreadCS);

    if (m_pThread)         // does the thread still exist?
    {
        // without a critical section, once reached this point it may happen
        // that the OS scheduler gives control to the MyThread::Entry() function,
        // which in turn may return (because it completes its work) making
        // invalid the m_pThread pointer

        if (m_pThread->Pause() != wxTHREAD_NO_ERROR )
            wxLogError("Can't pause the thread!");
    }
}

void MyFrame::OnClose(wxCloseEvent&)
{
    {
        wxCriticalSectionLocker enter(m_pThreadCS);

        if (m_pThread)         // does the thread still exist?
        {
            wxMessageOutputDebug().Printf("MYFRAME: deleting thread");

            if (m_pThread->Delete() != wxTHREAD_NO_ERROR )
                wxLogError("Can't delete the thread!");
        }
    }       // exit from the critical section to give the thread
            // the possibility to enter its destructor
            // (which is guarded with m_pThreadCS critical section!)

    while (1)
    {
```

```
        { // was the ~MyThread() function executed?
            wxCriticalSectionLocker enter(m_pThreadCS);
            if (!m_pThread) break;
        }

        // wait for thread completion
        wxThread::This()->Sleep(1);
    }

    Destroy();
}
```

For a more detailed and comprehensive example, see Thread Sample. For a simpler way to share data and synchronization objects between the main and the secondary thread see wxThreadHelper.

Conversely, **joinable** threads do not delete themselves when they are done processing and as such are safe to create on the stack. Joinable threads also provide the ability for one to get value it returned from Entry() through Wait(). You shouldn't hurry to create all the threads joinable, however, because this has a disadvantage as well: you **must** Wait() for a joinable thread or the system resources used by it will never be freed, and you also must delete the corresponding wxThread object yourself if you did not create it on the stack. In contrast, detached threads are of the "fire-and-forget" kind: you only have to start a detached thread and it will terminate and destroy itself.

21.784.3 wxThread Deletion

Regardless of whether it has terminated or not, you should call Wait() on a **joinable** thread to release its memory, as outlined in Types of wxThreads. If you created a joinable thread on the heap, remember to delete it manually with the `delete` operator or similar means as only detached threads handle this type of memory management.

Since **detached** threads delete themselves when they are finished processing, you should take care when calling a routine on one. If you are certain the thread is still running and would like to end it, you may call Delete() to gracefully end it (which implies that the thread will be deleted after that call to Delete()). It should be implied that you should **never** attempt to delete a detached thread with the `delete` operator or similar means.

As mentioned, Wait() or Delete() functions attempt to gracefully terminate a joinable and a detached thread, respectively. They do this by waiting until the thread in question calls TestDestroy() or ends processing (i.e. returns from wxThread::Entry).

Obviously, if the thread does call TestDestroy() and does not end, the thread which called Wait() or Delete() will come to halt. This is why it's important to call TestDestroy() in the Entry() routine of your threads as often as possible and immediately exit when it returns true.

As a last resort you can end the thread immediately through Kill(). It is strongly recommended that you do not do this, however, as it does not free the resources associated with the object (although the wxThread object of detached threads will still be deleted) and could leave the C runtime library in an undefined state.

21.784.4 wxWidgets Calls in Secondary Threads

All threads other than the "main application thread" (the one running wxApp::OnInit() or the one your main function runs in, for example) are considered "secondary threads".

GUI calls, such as those to a wxWindow or wxBitmap are explicitly not safe at all in secondary threads and could end your application prematurely. This is due to several reasons, including the underlying native API and the fact that wxThread does not run a GUI event loop similar to other APIs as MFC.

A workaround for some wxWidgets ports is calling wxMutexGUIEnter() before any GUI calls and then calling wx↩MutexGUILeave() afterwords. However, the recommended way is to simply process the GUI calls in the main thread through an event that is posted by wxQueueEvent(). This does not imply that calls to these classes are thread-safe, however, as most wxWidgets classes are not thread-safe, including wxString.

21.784.5 Don't Poll a wxThread

A common problem users experience with wxThread is that in their main thread they will check the thread every now and then to see if it has ended through IsRunning(), only to find that their application has run into problems

because the thread is using the default behaviour (i.e. it's **detached**) and has already deleted itself. Naturally, they instead attempt to use joinable threads in place of the previous behaviour. However, polling a wxThread for when it has ended is in general a bad idea - in fact calling a routine on any running wxThread should be avoided if possible. Instead, find a way to notify yourself when the thread has ended.

Usually you only need to notify the main thread, in which case you can post an event to it via wxQueueEvent(). In the case of secondary threads you can call a routine of another class when the thread is about to complete processing and/or set the value of a variable, possibly using mutexes (see wxMutex) and/or other synchronization means if necessary.

Library: wxBase

Category: Threading

See also

wxThreadHelper, wxMutex, wxCondition, wxCriticalSection, Multithreading Overview

Public Types

- typedef void * ExitCode

  The return type for the thread functions.

Public Member Functions

- wxThread (wxThreadKind kind=wxTHREAD_DETACHED)

  This constructor creates a new detached (default) or joinable C++ thread object.
- virtual ~wxThread ()

  The destructor frees the resources associated with the thread.
- wxThreadError Create (unsigned int stackSize=0)

  Creates a new thread.
- wxThreadError Delete (ExitCode *rc=NULL, wxThreadWait waitMode=wxTHREAD_WAIT_BLOCK)

  *Calling Delete() gracefully terminates a **detached** thread, either when the thread calls TestDestroy() or when it finishes processing.*
- wxThreadIdType GetId () const

  Gets the thread identifier: this is a platform dependent number that uniquely identifies the thread throughout the system during its existence (i.e. the thread identifiers may be reused).
- WXHANDLE MSWGetHandle () const

  Gets the native thread handle.
- wxThreadKind GetKind () const

  Returns the thread kind as it was given in the ctor.
- unsigned int GetPriority () const

  Gets the priority of the thread, between 0 (lowest) and 100 (highest).
- bool IsAlive () const

  Returns true if the thread is alive (i.e. started and not terminating).
- bool IsDetached () const

  Returns true if the thread is of the detached kind, false if it is a joinable one.
- bool IsPaused () const

  Returns true if the thread is paused.
- bool IsRunning () const

  Returns true if the thread is running.
- wxThreadError Kill ()

Immediately terminates the target thread.

- wxThreadError Pause ()

  Suspends the thread.

- wxThreadError Resume ()

  Resumes a thread suspended by the call to Pause().

- wxThreadError Run ()

  Starts the thread execution.

- void SetPriority (unsigned int priority)

  Sets the priority of the thread, between 0 (lowest) and 100 (highest).

- virtual bool TestDestroy ()

  This function should be called periodically by the thread to ensure that calls to Pause() and Delete() will work.

- ExitCode Wait (wxThreadWait flags=wxTHREAD_WAIT_BLOCK)

  *Waits for a **joinable** thread to terminate and returns the value the thread returned from Entry() or "`(ExitCode)-1`" on error.*

Static Public Member Functions

- static int GetCPUCount ()

  Returns the number of system CPUs or -1 if the value is unknown.

- static wxThreadIdType GetCurrentId ()

  Returns the platform specific thread ID of the current thread as a long.

- static wxThreadIdType GetMainId ()

  Returns the thread ID of the main thread.

- static bool IsMain ()

  Returns true if the calling thread is the main application thread.

- static bool SetConcurrency (size_t level)

  Sets the thread concurrency level for this process.

- static void Sleep (unsigned long milliseconds)

  Pauses the thread execution for the given amount of time.

- static wxThread * This ()

  Return the thread object for the calling thread.

- static void Yield ()

  Give the rest of the thread's time-slice to the system allowing the other threads to run.

Protected Member Functions

- virtual ExitCode Entry ()=0

  This is the entry point of the thread.

- void Exit (ExitCode exitcode=0)

  This is a protected function of the wxThread class and thus can only be called from a derived class.

Private Member Functions

- virtual void OnExit ()

  Called when the thread exits.

21.784.6 Member Typedef Documentation

typedef void* wxThread::ExitCode

The return type for the thread functions.

21.784.7 Constructor & Destructor Documentation

wxThread::wxThread (wxThreadKind *kind* = wxTHREAD_DETACHED)

This constructor creates a new detached (default) or joinable C++ thread object.

It does not create or start execution of the real thread - for this you should use the Run() method.

The possible values for *kind* parameters are:

- **wxTHREAD_DETACHED** - Creates a detached thread.
- **wxTHREAD_JOINABLE** - Creates a joinable thread.

virtual wxThread::~wxThread ( ) [virtual]

The destructor frees the resources associated with the thread.

Notice that you should never delete a detached thread – you may only call Delete() on it or wait until it terminates (and auto destructs) itself.

Because the detached threads delete themselves, they can only be allocated on the heap. Joinable threads should be deleted explicitly. The Delete() and Kill() functions will not delete the C++ thread object. It is also safe to allocate them on stack.

21.784.8 Member Function Documentation

wxThreadError wxThread::Create (unsigned int *stackSize* = 0)

Creates a new thread.

The thread object is created in the suspended state, and you should call Run() to start running it. You may optionally specify the stack size to be allocated to it (Ignored on platforms that don't support setting it explicitly, eg. Unix system without `pthread_attr_setstacksize`).

If you do not specify the stack size, the system's default value is used.

Note

It is not necessary to call this method since 2.9.5, Run() will create the thread internally. You only need to call Create() if you need to do something with the thread (e.g. pass its ID to an external library) before it starts.

Warning

It is a good idea to explicitly specify a value as systems' default values vary from just a couple of KB on some systems (BSD and OS/2 systems) to one or several MB (Windows, Solaris, Linux). So, if you have a thread that requires more than just a few KB of memory, you will have mysterious problems on some platforms but not on the common ones. On the other hand, just indicating a large stack size by default will give you performance issues on those systems with small default stack since those typically use fully committed memory for the stack. On the contrary, if you use a lot of threads (say several hundred), virtual address space can get tight unless you explicitly specify a smaller amount of thread stack space for each thread.

Returns

One of:

- **wxTHREAD_NO_ERROR** - No error.
- **wxTHREAD_NO_RESOURCE** - There were insufficient resources to create the thread.
- **wxTHREAD_NO_RUNNING** - The thread is already running

wxThreadError wxThread::Delete (ExitCode * *rc* = NULL, wxThreadWait *waitMode* = wxTHREAD_WAIT_BLOCK)

Calling Delete() gracefully terminates a **detached** thread, either when the thread calls TestDestroy() or when it finishes processing.

Parameters

rc	The thread exit code, if rc is not NULL.
waitMode	As described in wxThreadWait documentation, wxTHREAD_WAIT_BLOCK should be used as the wait mode even although currently wxTHREAD_WAIT_YIELD is for compatibility reasons. This parameter is new in wxWidgets 2.9.2.

Note

This function works on a joinable thread but in that case makes the TestDestroy() function of the thread return true and then waits for its completion (i.e. it differs from Wait() because it asks the thread to terminate before waiting).

See wxThread Deletion for a broader explanation of this routine.

virtual ExitCode wxThread::Entry ( ) [protected], [pure virtual]

This is the entry point of the thread.

This function is pure virtual and must be implemented by any derived class. The thread execution will start here.

The returned value is the thread exit code which is only useful for joinable threads and is the value returned by Wait(). This function is called by wxWidgets itself and should never be called directly.

void wxThread::Exit (ExitCode *exitcode* = 0) [protected]

This is a protected function of the wxThread class and thus can only be called from a derived class.

It also can only be called in the context of this thread, i.e. a thread can only exit from itself, not from another thread.

This function will terminate the OS thread (i.e. stop the associated path of execution) and also delete the associated C++ object for detached threads. OnExit() will be called just before exiting.

static int wxThread::GetCPUCount ( ) [static]

Returns the number of system CPUs or -1 if the value is unknown.

For multi-core systems the returned value is typically the total number of *cores*, since the OS usually abstract a single N-core CPU as N different cores.

See also

SetConcurrency()

static wxThreadIdType wxThread::GetCurrentId ( ) [static]

Returns the platform specific thread ID of the current thread as a long.

This can be used to uniquely identify threads, even if they are not wxThreads.

See also

GetMainId()

wxThreadIdType wxThread::GetId ( ) const

Gets the thread identifier: this is a platform dependent number that uniquely identifies the thread throughout the system during its existence (i.e. the thread identifiers may be reused).

wxThreadKind wxThread::GetKind ( ) const

Returns the thread kind as it was given in the ctor.

Since

2.9.0

static wxThreadIdType wxThread::GetMainId ( ) [static]

Returns the thread ID of the main thread.

See also

IsMain()

Since

2.9.1

unsigned int wxThread::GetPriority ( ) const

Gets the priority of the thread, between 0 (lowest) and 100 (highest).

See also

SetPriority()

bool wxThread::IsAlive ( ) const

Returns true if the thread is alive (i.e. started and not terminating).

Note that this function can only safely be used with joinable threads, not detached ones as the latter delete themselves and so when the real thread is no longer alive, it is not possible to call this function because the wxThread object no longer exists.

bool wxThread::IsDetached ( ) const

Returns true if the thread is of the detached kind, false if it is a joinable one.

static bool wxThread::IsMain ( ) [static]

Returns true if the calling thread is the main application thread.

Main thread in the context of wxWidgets is the one which initialized the library.

See also

GetMainId(), GetCurrentId()

bool wxThread::IsPaused ( ) const

Returns true if the thread is paused.

bool wxThread::IsRunning ( ) const

Returns true if the thread is running.

This method may only be safely used for joinable threads, see the remark in IsAlive().

wxThreadError wxThread::Kill ( )

Immediately terminates the target thread.

"This function is dangerous and should be used with extreme care" (and not used at all whenever possible)! The resources allocated to the thread will not be freed and the state of the C runtime library may become inconsistent. Use Delete() for detached threads or Wait() for joinable threads instead.

For detached threads Kill() will also delete the associated C++ object. However this will not happen for joinable threads and this means that you will still have to delete the wxThread object yourself to avoid memory leaks.

In neither case OnExit() of the dying thread will be called, so no thread-specific cleanup will be performed. This function can only be called from another thread context, i.e. a thread cannot kill itself.

It is also an error to call this function for a thread which is not running or paused (in the latter case, the thread will be resumed first) – if you do it, a **wxTHREAD_NOT_RUNNING** error will be returned.

WXHANDLE wxThread::MSWGetHandle ( ) const

Gets the native thread handle.

This method only exists in wxMSW, use GetId() in portable code.

Since

3.1.0

virtual void wxThread::OnExit ( ) [private], [virtual]

Called when the thread exits.

This function is called in the context of the thread associated with the wxThread object, not in the context of the main thread. This function will not be called if the thread was Kill() killed.

This function should never be called directly.

wxThreadError wxThread::Pause ( )

Suspends the thread.

Under some implementations (Win32), the thread is suspended immediately, under others it will only be suspended when it calls TestDestroy() for the next time (hence, if the thread doesn't call it at all, it won't be suspended).

This function can only be called from another thread context.

wxThreadError wxThread::Resume ( )

Resumes a thread suspended by the call to Pause().

This function can only be called from another thread context.

wxThreadError wxThread::Run ( )

Starts the thread execution.

Note that once you Run() a **detached** thread, *any* function call you do on the thread pointer (you must allocate it on the heap) is *"unsafe"*; i.e. the thread may have terminated at any moment after Run() and your pointer may be dangling. See Types of wxThreads for an example of safe manipulation of detached threads.

This function can only be called from another thread context.

Finally, note that once a thread has completed and its Entry() function returns, you cannot call Run() on it again (an assert will fail in debug builds or `wxTHREAD_RUNNING` will be returned in release builds).

static bool wxThread::SetConcurrency (size_t *level*) [static]

Sets the thread concurrency level for this process.

This is, roughly, the number of threads that the system tries to schedule to run in parallel. The value of 0 for *level* may be used to set the default one.

Returns

true on success or false otherwise (for example, if this function is not implemented for this platform – currently everything except Solaris).

void wxThread::SetPriority (unsigned int *priority*)

Sets the priority of the thread, between 0 (lowest) and 100 (highest).

The following symbolic constants can be used in addition to raw values in 0..100 range:

- `wxPRIORITY_MIN`: 0
- `wxPRIORITY_DEFAULT`: 50
- `wxPRIORITY_MAX`: 100

Notice that in the MSW implementation the thread priority can currently be only set after creating the thread with CreateThread(). But under all platforms this method can be called either before launching the thread using Run() or after doing it.

static void wxThread::Sleep (unsigned long *milliseconds*) [static]

Pauses the thread execution for the given amount of time.

This is the same as wxMilliSleep().

virtual bool wxThread::TestDestroy ( ) [virtual]

This function should be called periodically by the thread to ensure that calls to Pause() and Delete() will work.

If it returns true, the thread should exit as soon as possible. Notice that under some platforms (POSIX), implementation of Pause() also relies on this function being called, so not calling it would prevent both stopping and suspending thread from working.

static wxThread* wxThread::This ( ) [static]

Return the thread object for the calling thread.

NULL is returned if the calling thread is the main (GUI) thread, but IsMain() should be used to test whether the thread is really the main one because NULL may also be returned for the thread not created with wxThread class. Generally speaking, the return value for such a thread is undefined.

ExitCode wxThread::Wait (wxThreadWait *flags* = wxTHREAD_WAIT_BLOCK)

Waits for a **joinable** thread to terminate and returns the value the thread returned from Entry() or "`(ExitCode)-1`" on error.

Notice that, unlike Delete(), this function doesn't cancel the thread in any way so the caller waits for as long as it takes to the thread to exit.

You can only Wait() for **joinable** (not detached) threads.

This function can only be called from another thread context.

Parameters

flags	As described in wxThreadWait documentation, wxTHREAD_WAIT_BLOCK should be used as the wait mode even although currently wxTHREAD_WAIT_YIELD is for compatibility reasons. This parameter is new in wxWidgets 2.9.2.

See wxThread Deletion for a broader explanation of this routine.

static void wxThread::Yield ( ) `[static]`

Give the rest of the thread's time-slice to the system allowing the other threads to run.

Note that using this function is **strongly** discouraged, since in many cases it indicates a design weakness of your threading model (as does using Sleep() functions).

Threads should use the CPU in an efficient manner, i.e. they should do their current work efficiently, then as soon as the work is done block on a wakeup event (wxCondition, wxMutex, select(), poll(), ...) which will get signalled e.g. by other threads or a user device once further thread work is available. Using Yield() or Sleep() indicates polling-type behaviour, since we're fuzzily giving up our timeslice and wait until sometime later we'll get reactivated, at which time we realize that there isn't really much to do and Yield() again...

The most critical characteristic of Yield() is that it's operating system specific: there may be scheduler changes which cause your thread to not wake up relatively soon again, but instead many seconds later, causing huge performance issues for your application.

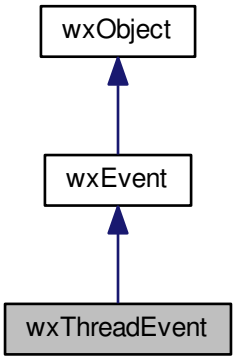
With a well-behaving, CPU-efficient thread the operating system is likely to properly care for its reactivation the moment it needs it, whereas with non-deterministic, Yield-using threads all bets are off and the system scheduler is free to penalize them drastically, and this effect gets worse with increasing system load due to less free CPU resources available. You may refer to various Linux kernel `sched_yield` discussions for more information.

See also Sleep().

21.785 wxThreadEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxThreadEvent:



21.785.1 Detailed Description

This class adds some simple functionality to wxEvent to facilitate inter-thread communication.

This event is not natively emitted by any control/class: it is just a helper class for the user. Its most important feature is the GetEventCategory() implementation which allows thread events **NOT** to be processed by wxEventLoop↩ Base::YieldFor calls (unless the `wxEVT_CATEGORY_THREAD` is specified - which is never in wx code).

Library: wxCore

Category: Events, Threading

See also

Multithreading Overview, wxEventLoopBase::YieldFor

Since

2.9.0

Public Member Functions

- wxThreadEvent (wxEventType eventType=wxEVT_THREAD, int id=wxID_ANY)

  Constructor.
- virtual wxEvent * Clone () const

  Clones this event making sure that all internal members which use COW (only `m_commandString` for now; see Reference Counting) are unshared (see wxObject::UnShare).
- virtual wxEventCategory GetEventCategory () const

  Returns `wxEVT_CATEGORY_THREAD`.
- template<typename T >
  void SetPayload (const T &payload)

  Sets custom data payload.

- template<typename T >
  T GetPayload () const

  Get custom data payload.
- long GetExtraLong () const

  Returns extra information integer value.
- int GetInt () const

  Returns stored integer value.
- wxString GetString () const

  Returns stored string value.
- void SetExtraLong (long extraLong)

  Sets the extra information value.
- void SetInt (int intCommand)

  Sets the integer value.
- void SetString (const wxString &string)

  Sets the string value.

Additional Inherited Members

21.785.2 Constructor & Destructor Documentation

wxThreadEvent::wxThreadEvent (wxEventType *eventType* = wxEVT_THREAD, int *id* = wxID_ANY)

Constructor.

21.785.3 Member Function Documentation

virtual wxEvent* wxThreadEvent::Clone ( ) const `[virtual]`

Clones this event making sure that all internal members which use COW (only `m_commandString` for now; see Reference Counting) are unshared (see wxObject::UnShare).

Implements wxEvent.

virtual wxEventCategory wxThreadEvent::GetEventCategory ( ) const `[virtual]`

Returns `wxEVT_CATEGORY_THREAD`.

This is important to avoid unwanted processing of thread events when calling wxEventLoopBase::YieldFor().

Reimplemented from wxEvent.

long wxThreadEvent::GetExtraLong ( ) const

Returns extra information integer value.

int wxThreadEvent::GetInt ( ) const

Returns stored integer value.

template<typename T > T wxThreadEvent::GetPayload ( ) const

Get custom data payload.

Correct type is checked in debug builds.

Note

This method is not available with Visual C++ 6.

Since

2.9.1

See also

SetPayload(), wxAny

wxString wxThreadEvent::GetString ( ) const

Returns stored string value.

void wxThreadEvent::SetExtraLong (long *extraLong*)

Sets the extra information value.

void wxThreadEvent::SetInt (int *intCommand*)

Sets the integer value.

template<typename T > void wxThreadEvent::SetPayload (const T & *payload*)

Sets custom data payload.

The *payload* argument may be of any type that wxAny can handle (i.e. pretty much anything). Note that T's copy constructor must be thread-safe, i.e. create a copy that doesn't share anything with the original (see Clone()).

Note

This method is not available with Visual C++ 6.

Since

2.9.1

See also

GetPayload(), wxAny

void wxThreadEvent::SetString (const wxString & *string*)

Sets the string value.

21.786 wxThreadHelper Class Reference

`#include <wx/thread.h>`

21.786.1 Detailed Description

The wxThreadHelper class is a mix-in class that manages a single background thread, either detached or joinable (see wxThread for the differences).

By deriving from wxThreadHelper, a class can implement the thread code in its own wxThreadHelper::Entry() method and easily share data and synchronization objects between the main thread and the worker thread.

Doing this prevents the awkward passing of pointers that is needed when the original object in the main thread needs to synchronize with its worker thread in its own wxThread derived object.

For example, wxFrame may need to make some calculations in a background thread and then display the results of those calculations in the main window.

Ordinarily, a wxThread derived object would be created with the calculation code implemented in wxThread::Entry. To access the inputs to the calculation, the frame object would often need to pass a pointer to itself to the thread object. Similarly, the frame object would hold a pointer to the thread object.

Shared data and synchronization objects could be stored in either object though the object without the data would have to access the data through a pointer. However with wxThreadHelper the frame object and the thread object are treated as the same object. Shared data and synchronization variables are stored in the single object, eliminating a layer of indirection and the associated pointers.

Example:

```
wxDECLARE_EVENT(myEVT_THREAD_UPDATE, wxThreadEvent);

class MyFrame : public wxFrame, public wxThreadHelper
{
public:
    MyFrame(...) { ... }
    ~MyFrame()
    {
        // it's better to do any thread cleanup in the OnClose()
        // event handler, rather than in the destructor.
        // This is because the event loop for a top-level window is not
        // active anymore when its destructor is called and if the thread
        // sends events when ending, they won't be processed unless
        // you ended the thread from OnClose.
        // See @ref overview_windowdeletion for more info.
    }

    ...
    void DoStartALongTask();
    void OnThreadUpdate(wxThreadEvent& evt);
    void OnClose(wxCloseEvent& evt);
    ...

protected:
    virtual wxThread::ExitCode Entry();

    // the output data of the Entry() routine:
    char m_data[1024];
    wxCriticalSection m_dataCS; // protects field above

    wxDECLARE_EVENT_TABLE();
};

wxDEFINE_EVENT(myEVT_THREAD_UPDATE, wxThreadEvent);
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_THREAD(wxID_ANY, myEVT_THREAD_UPDATE, MyFrame::OnThreadUpdate)
    EVT_CLOSE(MyFrame::OnClose)
wxEND_EVENT_TABLE()

void MyFrame::DoStartALongTask()
{
    // we want to start a long task, but we don't want our GUI to block
    // while it's executed, so we use a thread to do it.
    if (CreateThread(wxTHREAD_JOINABLE) !=
      wxTHREAD_NO_ERROR)
    {
        wxLogError("Could not create the worker thread!");
```

```
        return;
    }

    // go!
    if (GetThread()->Run() != wxTHREAD_NO_ERROR)
    {
        wxLogError("Could not run the worker thread!");
        return;
    }
}

wxThread::ExitCode MyFrame::Entry()
{
    // IMPORTANT:
    // this function gets executed in the secondary thread context!

    int offset = 0;

    // here we do our long task, periodically calling TestDestroy():
    while (!GetThread()->TestDestroy())
    {
        // since this Entry() is implemented in MyFrame context we don't
        // need any pointer to access the m_data, m_processedData, m_dataCS
        // variables... very nice!

        // this is an example of the generic structure of a download thread:
        char buffer[1024];
        download_chunk(buffer, 1024);     // this takes time...

        {
            // ensure no one reads m_data while we write it
            wxCriticalSectionLocker lock(m_dataCS);
            memcpy(m_data+offset, buffer, 1024);
            offset += 1024;
        }


        // VERY IMPORTANT: do not call any GUI function inside this
        //                 function; rather use wxQueueEvent():
        wxQueueEvent(this, new wxThreadEvent(wxEVT_COMMAND_MYTHREAD_UPDATE));
            // we used pointer 'this' assuming it's safe; see OnClose()
    }

    // TestDestroy() returned true (which means the main thread asked us
    // to terminate as soon as possible) or we ended the long task...
    return (wxThread::ExitCode)0;
}

void MyFrame::OnClose(wxCloseEvent&)
{
    // important: before terminating, we _must_ wait for our joinable
    // thread to end, if it's running; in fact it uses variables of this
    // instance and posts events to *this event handler

    if (GetThread() &&      // DoStartALongTask() may have not been called
        GetThread()->IsRunning())
        GetThread()->Wait();

    Destroy();
}

void MyFrame::OnThreadUpdate(wxThreadEvent& evt)
{
    // ...do something... e.g. m_pGauge->Pulse();

    // read some parts of m_data just for fun:
    wxCriticalSectionLocker lock(m_dataCS);
    wxPrintf("%c", m_data[100]);
}
```

Library: wxBase

Category: Threading

See also

wxThread, wxThreadEvent

Public Member Functions

- wxThreadHelper (wxThreadKind kind=wxTHREAD_JOINABLE)

  This constructor simply initializes internal member variables and tells wxThreadHelper which type the thread internally managed should be.
- virtual ~wxThreadHelper ()

  The destructor frees the resources associated with the thread, forcing it to terminate (it uses wxThread::Kill function).
- virtual ExitCode Entry ()=0

  This is the entry point of the thread.
- virtual void OnDelete ()

  Callback called by Delete() before actually deleting the thread.
- virtual void OnKill ()

  Callback called by Kill() before actually killing the thread.
- wxThreadError Create (unsigned int stackSize=0)
- wxThreadError CreateThread (wxThreadKind kind=wxTHREAD_JOINABLE, unsigned int stackSize=0)

  Creates a new thread of the given kind.
- wxThread * GetThread () const

  This is a public function that returns the wxThread object associated with the thread.
- wxThreadKind GetThreadKind () const

  Returns the last type of thread given to the CreateThread() function or to the constructor.

21.786.2 Constructor & Destructor Documentation

wxThreadHelper::wxThreadHelper (wxThreadKind *kind* = wxTHREAD_JOINABLE)

This constructor simply initializes internal member variables and tells wxThreadHelper which type the thread internally managed should be.

virtual wxThreadHelper::~wxThreadHelper ( ) [virtual]

The destructor frees the resources associated with the thread, forcing it to terminate (it uses wxThread::Kill function).

Because of the wxThread::Kill unsafety, you should always wait (with wxThread::Wait) for joinable threads to end or call wxThread::Delete on detached threads, instead of relying on this destructor for stopping the thread.

21.786.3 Member Function Documentation

wxThreadError wxThreadHelper::Create (unsigned int *stackSize* = 0)

Deprecated Use CreateThread() instead.

wxThreadError wxThreadHelper::CreateThread (wxThreadKind *kind* = wxTHREAD_JOINABLE, unsigned int *stackSize* = 0)

Creates a new thread of the given *kind*.

The thread object is created in the suspended state, and you should call GetThread()->Run() to start running it.

You may optionally specify the stack size to be allocated to it (ignored on platforms that don't support setting it explicitly, e.g. Unix).

Returns

One of the wxThreadError enum values.

virtual ExitCode wxThreadHelper::Entry ( ) [pure virtual]

This is the entry point of the thread.

This function is pure virtual and must be implemented by any derived class. The thread execution will start here.

You'll typically want your Entry() to look like:

```
wxThread::ExitCode Entry()
{
    while (!GetThread()->TestDestroy())
    {
        // ... do some work ...

        if (IsWorkCompleted)
            break;

        if (HappenedStoppingError)
            return (wxThread::ExitCode)1;   // failure
    }

    return (wxThread::ExitCode)0;           // success
}
```

The returned value is the thread exit code which is only useful for joinable threads and is the value returned by `"GetThread()->Wait()"`.

This function is called by wxWidgets itself and should never be called directly.

wxThread* wxThreadHelper::GetThread ( ) const

This is a public function that returns the wxThread object associated with the thread.

wxThreadKind wxThreadHelper::GetThreadKind ( ) const

Returns the last type of thread given to the CreateThread() function or to the constructor.

virtual void wxThreadHelper::OnDelete ( ) [virtual]

Callback called by Delete() before actually deleting the thread.

This function can be overridden by the derived class to perform some specific task when the thread is gracefully destroyed. Notice that it will be executed in the context of the thread that called Delete() and **not** in this thread's context.

TestDestroy() will be true for the thread before OnDelete() gets executed.

Since

2.9.2

See also

OnKill()

virtual void wxThreadHelper::OnKill ( ) [virtual]

Callback called by Kill() before actually killing the thread.

This function can be overridden by the derived class to perform some specific task when the thread is terminated. Notice that it will be executed in the context of the thread that called Kill() and **not** in this thread's context.

Since

2.9.2

See also

OnDelete()

21.787 wxThumbBarButton Class Reference

`#include <wx/taskbarbutton.h>`

21.787.1 Detailed Description

A thumbnail toolbar button is a control that displayed in the thumbnail image of a window in a taskbar button flyout.

Library: wxCore

Category: Miscellaneous

Availability: only available for the wxMSW port.

See also

wxTaskBarButton

Since

3.1.0

Public Member Functions

- wxThumbBarButton ()

  Default constructor to allow 2-phase creation.
- wxThumbBarButton (int id, const wxIcon &icon, const wxString &tooltip=wxString(), bool enable=true, bool dismissOnClick=false, bool hasBackground=true, bool shown=true, bool interactive=true)

  Constructs the thumbnail toolbar button.
- bool Create (int id, const wxIcon &icon, const wxString &tooltip=wxString(), bool enable=true, bool dismiss↩ OnClick=false, bool hasBackground=true, bool shown=true, bool interactive=true)
- virtual ~wxThumbBarButton ()
- int GetID () const

  Returns the identifier associated with this control.
- const wxIcon & GetIcon () const

  Returns the icon associated with this control.
- const wxString & GetTooltip () const

  Returns the tooltip.
- bool IsEnable () const

  Returns true if the button is enabled, false if it has been disabled.
- void Enable (bool enable=true)

  Enables or disables the thumbnail toolbar button.
- void Disable ()

Equivalent to calling wxThumbBarButton::Enable(false).

- bool IsDismissOnClick () const

  Returns true if the button will dismiss on click.

- void EnableDismissOnClick (bool enable=true)

  Whether the window thumbnail is dismissed after a button click.

- void DisableDimissOnClick ()

  Equivalent to calling wxThumbBarButton::DisableDimissOnClick(false).

- bool HasBackground () const

  Returns true if the button has button border.

- void SetHasBackground (bool has=true)

  Set the property that whether the button has background.

- bool IsShown () const

  Returns true if the button is shown, false if it has been hidden.

- void Show (bool shown=true)

  Show or hide the thumbnail toolbar button.

- void Hide ()

  Hide the thumbnail toolbar button.

- bool IsInteractive () const

  Returns true if the button is interactive.

- void SetInteractive (bool interactive=true)

  Set the property which holds whether the button is interactive.

21.787.2 Constructor & Destructor Documentation

wxThumbBarButton::wxThumbBarButton ( )

Default constructor to allow 2-phase creation.

wxThumbBarButton::wxThumbBarButton (int *id*, const wxIcon & *icon*, const wxString & *tooltip* = wxString(), bool *enable* = `true`, bool *dismissOnClick* = `false`, bool *hasBackground* = `true`, bool *shown* = `true`, bool *interactive* = `true`)

Constructs the thumbnail toolbar button.

Parameters

id	The identifier for the control.
icon	The icon used as the button image.
tooltip	The text of the button's tooltip, displayed when the mouse pointer hovers over the button.
enable	If true (default), the button is active and available to the user. If false, the button is disabled. It is present, but has a visual state that indicates that it will not respond to user action.
dismissOnClick	If true, when the button is clicked, the taskbar button's flyout closes immediately. false by default.
hasBackground	If false, the button border is not drawn. true by default.
shown	If false, the button is not shown to the user. true by default.
interactive	If false, the button is enabled but not interactive; no pressed button state is drawn. This flag is intended for instances where the button is used in a notification. true by default.

virtual wxThumbBarButton::~wxThumbBarButton ( ) `[virtual]`

21.787.3 Member Function Documentation

bool wxThumbBarButton::Create (int *id,* const wxIcon & *icon,* const wxString & *tooltip* = wxString (), bool *enable* = true, bool *dismissOnClick* = false, bool *hasBackground* = true, bool *shown* = true, bool *interactive* = true)

void wxThumbBarButton::Disable ( )

Equivalent to calling wxThumbBarButton::Enable(false).

void wxThumbBarButton::DisableDimissOnClick ( )

Equivalent to calling wxThumbBarButton::DisableDimissOnClick(false).

void wxThumbBarButton::Enable (bool *enable* = true)

Enables or disables the thumbnail toolbar button.

void wxThumbBarButton::EnableDismissOnClick (bool *enable* = true)

Whether the window thumbnail is dismissed after a button click.

const wxIcon& wxThumbBarButton::GetIcon ( ) const

Returns the icon associated with this control.

int wxThumbBarButton::GetID ( ) const

Returns the identifier associated with this control.

const wxString& wxThumbBarButton::GetTooltip ( ) const

Returns the tooltip.

bool wxThumbBarButton::HasBackground ( ) const

Returns true if the button has button border.

void wxThumbBarButton::Hide ( )

Hide the thumbnail toolbar button.

Equivalent to calling wxThumbBarButton::Show(false).

bool wxThumbBarButton::IsDismissOnClick ( ) const

Returns true if the button will dismiss on click.

bool wxThumbBarButton::IsEnable ( ) const

Returns true if the button is enabled, false if it has been disabled.

bool wxThumbBarButton::IsInteractive ( ) const

Returns true if the button is interactive.

bool wxThumbBarButton::IsShown ( ) const

Returns true if the button is shown, false if it has been hidden.

void wxThumbBarButton::SetHasBackground (bool *has* = `true`)

Set the property that whether the button has background.

void wxThumbBarButton::SetInteractive (bool *interactive* = `true`)

Set the property which holds whether the button is interactive.

A non-interactive thumbnail toolbar button does not react to user interaction, but is still visually enabled.

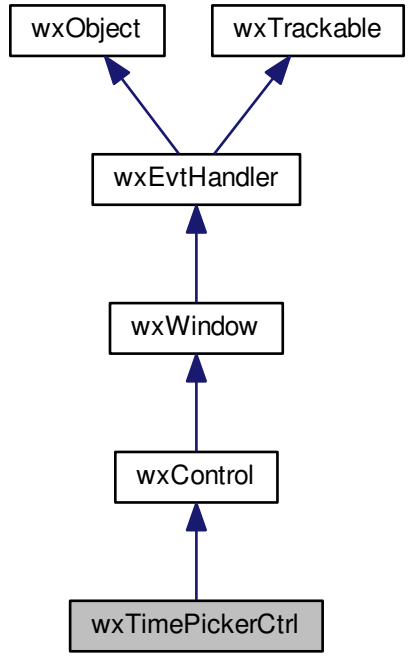
void wxThumbBarButton::Show (bool *shown* = `true`)

Show or hide the thumbnail toolbar button.

21.788 wxTimePickerCtrl Class Reference

```
#include <wx/timectrl.h>
```

Inheritance diagram for wxTimePickerCtrl:



21.788.1 Detailed Description

This control allows the user to enter time.

It is similar to wxDatePickerCtrl but is used for time, and not date, selection. While GetValue() and SetValue() still work with values of type wxDateTime (because wxWidgets doesn't provide a time-only class), their date part is ignored by this control.

It is only available if `wxUSE_TIMEPICKCTRL` is set to 1.

This control currently doesn't have any specific flags.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxDateEvent& event)

Event macros for events emitted by this class:

- EVT_TIME_CHANGED(id, func): This event fires when the user changes the current selection in the control.

Library: wxAdvanced

Category: Picker Controls

See also

wxDatePickerCtrl, wxDateEvent

Since

2.9.3

Public Member Functions

- wxTimePickerCtrl ()

  Default constructor.
- wxTimePickerCtrl (wxWindow *parent, wxWindowID id, const wxDateTime &dt=wxDefaultDateTime, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTP_DEFAULT, const wxValidator &validator=wxDefaultValidator, const wxString &name="timectrl")

  Initializes the object and calls Create() with all the parameters.
- bool Create (wxWindow *parent, wxWindowID id, const wxDateTime &dt=wxDefaultDateTime, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDP_DEFAULT|wxDP_SHOWCENTURY, const wxValidator &validator=wxDefaultValidator, const wxString &name="timectrl")

  Create the control window.
- bool GetTime (int *hour, int *min, int *sec) const

  Returns the currently entered time as hours, minutes and seconds.
- virtual wxDateTime GetValue () const

  Returns the currently entered time.
- bool SetTime (int hour, int min, int sec)

  Changes the current time of the control.
- virtual void SetValue (const wxDateTime &dt)

  Changes the current value of the control.

Additional Inherited Members

21.788.2 Constructor & Destructor Documentation

wxTimePickerCtrl::wxTimePickerCtrl ( )

Default constructor.

wxTimePickerCtrl::wxTimePickerCtrl (wxWindow * *parent*, wxWindowID *id*, const wxDateTime & *dt* = wxDefaultDateTime, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTP_DEFAULT, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "timectrl")

Initializes the object and calls Create() with all the parameters.

21.788.3 Member Function Documentation

bool wxTimePickerCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxDateTime & *dt* = wxDefaultDateTime, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDP_DEFAULT|wxDP_SHOWCENTURY, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = "timectrl")

Create the control window.

This method should only be used for objects created using default constructor.

Parameters

parent	Parent window, must not be non-NULL.
id	The identifier for the control.
dt	The initial value of the control, if an invalid date (such as the default value) is used, the control is set to current time.
pos	Initial position.
size	Initial size. If left at default value, the control chooses its own best size by using the height approximately equal to a text control and width large enough to show the time fully.
style	The window style, should be left at 0 as there are no special styles for this control in this version.
validator	Validator which can be used for additional checks.
name	Control name.

Returns

true if the control was successfully created or false if creation failed.

bool wxTimePickerCtrl::GetTime (int * *hour*, int * *min*, int * *sec*) const

Returns the currently entered time as hours, minutes and seconds.

All the arguments must be non-NULL, false is returned otherwise and none of them is modified.

See also

SetTime()

Since

2.9.4

virtual wxDateTime wxTimePickerCtrl::GetValue ( ) const [virtual]

Returns the currently entered time.

The date part of the returned wxDateTime object is always set to today and should be ignored, only the time part is relevant.

bool wxTimePickerCtrl::SetTime (int *hour*, int *min*, int *sec*)

Changes the current time of the control.

Calling this method does not result in a time change event.

Parameters

hour	The new hour value in 0..23 interval.
min	The new minute value in 0..59 interval.
sec	The new second value in 0..59 interval.

Returns

true if the time was changed or false on failure, e.g. if the time components were invalid.

See also

GetTime()

Since

2.9.4

virtual void wxTimePickerCtrl::SetValue (const wxDateTime & *dt*) [virtual]

Changes the current value of the control.

The date part of *dt* is ignored, only the time part is displayed in the control. The *dt* object must however be valid.

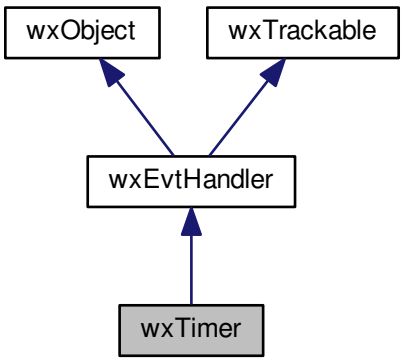
In particular notice that it is a bad idea to use default wxDateTime constructor from hour, minute and second values as it uses the today date for the date part which means that some times can be invalid if today happens to be the day of DST change. For example, when switching to summer time the time 2:00 typically doesn't exist as the clocks jump directly to 3:00. To avoid this problem, use a fixed date on which DST is known not to change (e.g. Jan 1, 2012) for the date part of the argument or use SetTime().

Calling this method does not result in a time change event.

21.789 wxTimer Class Reference

`#include <wx/timer.h>`

Inheritance diagram for wxTimer:



21.789.1 Detailed Description

The wxTimer class allows you to execute code at specified intervals.

Its precision is platform-dependent, but in general will not be better than `1ms` nor worse than `1s`.

There are three different ways to use this class:

- You may derive a new class from wxTimer and override the wxTimer::Notify member to perform the required action.

- You may redirect the notifications to any wxEvtHandler derived object by using the non-default constructor or wxTimer::SetOwner. Then use the `EVT_TIMER` macro to connect it to the event handler which will receive wxTimerEvent notifications.
- You may use a derived class and the `EVT_TIMER` macro to connect it to an event handler defined in the derived class. If the default constructor is used, the timer object will be its own owner object, since it is derived from wxEvtHandler.

In any case, you must start the timer with wxTimer::Start() after constructing it before it actually starts sending notifications. It can be stopped later with wxTimer::Stop().

Note

A timer can only be used from the main thread.

Library: wxBase

Category: Miscellaneous

See also

wxStopWatch

Public Member Functions

- wxTimer ()

  Default constructor.
- wxTimer (wxEvtHandler *owner, int id=-1)

  Creates a timer and associates it with owner.
- virtual ~wxTimer ()

  Destructor.
- int GetId () const

  Returns the ID of the events generated by this timer.
- int GetInterval () const

  Returns the current interval for the timer (in milliseconds).
- wxEvtHandler * GetOwner () const

  Returns the current owner of the timer.
- bool IsOneShot () const

  Returns true if the timer is one shot, i.e. if it will stop after firing the first notification automatically.
- bool IsRunning () const

  Returns true if the timer is running, false if it is stopped.
- virtual void Notify ()

  This member should be overridden by the user if the default constructor was used and SetOwner() wasn't called.
- void SetOwner (wxEvtHandler *owner, int id=-1)

  Associates the timer with the given owner object.
- virtual bool Start (int milliseconds=-1, bool oneShot=wxTIMER_CONTINUOUS)

  (Re)starts the timer.
- bool StartOnce (int milliseconds=-1)

  Starts the timer for a once-only notification.
- virtual void Stop ()

  Stops the timer.

Additional Inherited Members

21.789.2 Constructor & Destructor Documentation

wxTimer::wxTimer ( )

Default constructor.

If you use it to construct the object and don't call SetOwner() later, you must override Notify() method to process the notifications.

wxTimer::wxTimer (wxEvtHandler * *owner*, int *id* = -1)

Creates a timer and associates it with *owner*.

Please see SetOwner() for the description of parameters.

virtual wxTimer::~wxTimer ( ) [virtual]

Destructor.

Stops the timer if it is running.

21.789.3 Member Function Documentation

int wxTimer::GetId ( ) const

Returns the ID of the events generated by this timer.

int wxTimer::GetInterval ( ) const

Returns the current interval for the timer (in milliseconds).

wxEvtHandler* wxTimer::GetOwner ( ) const

Returns the current *owner* of the timer.

If non-NULL this is the event handler which will receive the timer events (see wxTimerEvent) when the timer is running.

bool wxTimer::IsOneShot ( ) const

Returns true if the timer is one shot, i.e. if it will stop after firing the first notification automatically.

bool wxTimer::IsRunning ( ) const

Returns true if the timer is running, false if it is stopped.

virtual void wxTimer::Notify ( ) [virtual]

This member should be overridden by the user if the default constructor was used and SetOwner() wasn't called.

Perform whatever action which is to be taken periodically here.

Notice that throwing exceptions from this method is currently not supported, use event-based timer handling approach if an exception can be thrown while handling timer notifications.

void wxTimer::SetOwner (wxEvtHandler * *owner*, int *id* = -1)

Associates the timer with the given *owner* object.

When the timer is running, the owner will receive timer events (see wxTimerEvent) with *id* equal to *id* specified here.

virtual bool wxTimer::Start (int *milliseconds* = -1, bool *oneShot* = wxTIMER_CONTINUOUS) [virtual]

(Re)starts the timer.

If *milliseconds* parameter is -1 (value by default), the previous value is used. Returns false if the timer could not be started, true otherwise (in MS Windows timers are a limited resource).

If *oneShot* is false (the default), the Notify() function will be called repeatedly until the timer is stopped. If true, it will be called only once and the timer will stop automatically.

To make your code more readable you may also use the following symbolic constants:

- wxTIMER_CONTINUOUS: Start a normal, continuously running, timer
- wxTIMER_ONE_SHOT: Start a one shot timer Alternatively, use StartOnce().

If the timer was already running, it will be stopped by this method before restarting it.

bool wxTimer::StartOnce (int *milliseconds* = -1)

Starts the timer for a once-only notification.

This is a simple wrapper for Start() with `wxTIMER_ONE_SHOT` parameter.

Since

2.9.5

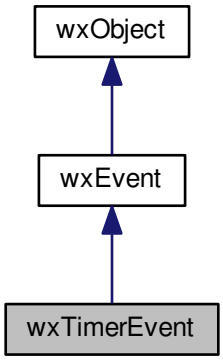
virtual void wxTimer::Stop ( ) [virtual]

Stops the timer.

21.790 wxTimerEvent Class Reference

`#include <wx/timer.h>`

Inheritance diagram for wxTimerEvent:



21.790.1 Detailed Description

wxTimerEvent object is passed to the event handler of timer events (see wxTimer::SetOwner).

For example:

```
class MyFrame : public wxFrame
{
public:
    ...
    void OnTimer(wxTimerEvent& event);

private:
    wxTimer m_timer;
    wxDECLARE_EVENT_TABLE();
};

wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_TIMER(TIMER_ID, MyFrame::OnTimer)
wxEND_EVENT_TABLE()

MyFrame::MyFrame()
       : m_timer(this, TIMER_ID)
{
    m_timer.Start(1000);    // 1 second interval
}

void MyFrame::OnTimer(wxTimerEvent& event)
{
    // do whatever you want to do every second here
}
```

Library: wxBase

Category: Events

See also

wxTimer

Public Member Functions

- wxTimerEvent ()

- wxTimerEvent (wxTimer &timer)
- int GetInterval () const

  Returns the interval of the timer which generated this event.
- wxTimer & GetTimer () const

  Returns the timer object which generated this event.

Additional Inherited Members

21.790.2 Constructor & Destructor Documentation

wxTimerEvent::wxTimerEvent ( )

wxTimerEvent::wxTimerEvent (wxTimer & *timer*)

21.790.3 Member Function Documentation

int wxTimerEvent::GetInterval ( ) const

Returns the interval of the timer which generated this event.

wxTimer& wxTimerEvent::GetTimer ( ) const

Returns the timer object which generated this event.

21.791 wxTimerRunner Class Reference

`#include <wx/timer.h>`

21.791.1 Detailed Description

Starts the timer in its ctor, stops in the dtor.

Public Member Functions

- wxTimerRunner (wxTimer &timer)
- wxTimerRunner (wxTimer &timer, int milli, bool oneShot=false)
- void Start (int milli, bool oneShot=false)
- ~wxTimerRunner ()

21.791.2 Constructor & Destructor Documentation

wxTimerRunner::wxTimerRunner (wxTimer & *timer*)

wxTimerRunner::wxTimerRunner (wxTimer & *timer,* int *milli,* bool *oneShot* = `false`)

wxTimerRunner::~wxTimerRunner ( )

21.791.3 Member Function Documentation

void wxTimerRunner::Start (int *milli,* bool *oneShot* = `false`)

21.792 wxTimeSpan Class Reference

```
#include <wx/datetime.h>
```

21.792.1 Detailed Description

wxTimeSpan class represents a time interval.

Library: wxBase

Category: Data Structures

See also

Date and Time, wxDateTime

Public Member Functions

- wxTimeSpan ()

  Default constructor, constructs a zero timespan.
- wxTimeSpan (long hours, long min=0, wxLongLong sec=0, wxLongLong msec=0)

  Constructs timespan from separate values for each component, with the date set to 0.
- wxTimeSpan Abs () const

  Returns the absolute value of the timespan: does not modify the object.
- wxTimeSpan Add (const wxTimeSpan &diff) const

  Returns the sum of two time spans.
- wxTimeSpan & Add (const wxTimeSpan &diff)

  Adds the given wxTimeSpan to this wxTimeSpan and returns a reference to itself.
- wxString Format (const wxString &format=wxDefaultTimeSpanFormat) const

  Returns the string containing the formatted representation of the time span.
- int GetDays () const

  Returns the difference in number of days.
- int GetHours () const

  Returns the difference in number of hours.
- wxLongLong GetMilliseconds () const

  Returns the difference in number of milliseconds.
- int GetMinutes () const

  Returns the difference in number of minutes.
- wxLongLong GetSeconds () const

  Returns the difference in number of seconds.
- wxLongLong GetValue () const

  Returns the internal representation of timespan.
- int GetWeeks () const

  Returns the difference in number of weeks.
- bool IsEqualTo (const wxTimeSpan &ts) const

  Returns true if two timespans are equal.
- bool IsLongerThan (const wxTimeSpan &ts) const

  Compares two timespans: works with the absolute values, i.e. -2 hours is longer than 1 hour.
- bool IsNegative () const

Returns true if the timespan is negative.

- bool IsNull () const

  Returns true if the timespan is empty.

- bool IsPositive () const

  Returns true if the timespan is positive.

- bool IsShorterThan (const wxTimeSpan &ts) const

  Compares two timespans: works with the absolute values, i.e. 1 hour is shorter than -2 hours.

- wxTimeSpan Multiply (int n) const

  Returns the product of this time span by n.

- wxTimeSpan & Multiply (int n)

  Multiplies this time span by n.

- wxTimeSpan & Neg ()

  Negate the value of the timespan.

- wxTimeSpan Negate () const

  Returns timespan with inverted sign.

- wxTimeSpan Subtract (const wxTimeSpan &diff) const

  Returns the difference of two time spans.

- wxTimeSpan & Subtract (const wxTimeSpan &diff)

  Subtracts the given wxTimeSpan to this wxTimeSpan and returns a reference to itself.

- wxTimeSpan & operator+= (const wxTimeSpan &diff)

  Adds the given wxTimeSpan to this wxTimeSpan and returns the result.

- wxTimeSpan & operator*= (int n)

  Multiplies this time span by n.

- wxTimeSpan & operator- ()

  Negate the value of the timespan.

- wxTimeSpan & operator-= (const wxTimeSpan &diff)

  Subtracts the given wxTimeSpan to this wxTimeSpan and returns the result.

Static Public Member Functions

- static wxTimeSpan Day ()

  Returns the timespan for one day.

- static wxTimeSpan Days (long days)

  Returns the timespan for the given number of days.

- static wxTimeSpan Hour ()

  Returns the timespan for one hour.

- static wxTimeSpan Hours (long hours)

  Returns the timespan for the given number of hours.

- static wxTimeSpan Millisecond ()

  Returns the timespan for one millisecond.

- static wxTimeSpan Milliseconds (wxLongLong ms)

  Returns the timespan for the given number of milliseconds.

- static wxTimeSpan Minute ()

  Returns the timespan for one minute.

- static wxTimeSpan Minutes (long min)

  Returns the timespan for the given number of minutes.

- static wxTimeSpan Second ()

  Returns the timespan for one second.

- static wxTimeSpan Seconds (wxLongLong sec)

  Returns the timespan for the given number of seconds.

- static wxTimeSpan Week ()

  Returns the timespan for one week.
- static wxTimeSpan Weeks (long weeks)

  Returns the timespan for the given number of weeks.

21.792.2 Constructor & Destructor Documentation

wxTimeSpan::wxTimeSpan ( )

Default constructor, constructs a zero timespan.

wxTimeSpan::wxTimeSpan (long *hours*, long *min* = `0`, wxLongLong *sec* = `0`, wxLongLong *msec* = `0`)

Constructs timespan from separate values for each component, with the date set to 0.

Hours are not restricted to 0-24 range, neither are minutes, seconds or milliseconds.

21.792.3 Member Function Documentation

wxTimeSpan wxTimeSpan::Abs ( ) const

Returns the absolute value of the timespan: does not modify the object.

wxTimeSpan wxTimeSpan::Add (const wxTimeSpan & *diff*) const

Returns the sum of two time spans.

Returns

A new wxDateSpan object with the result.

wxTimeSpan& wxTimeSpan::Add (const wxTimeSpan & *diff*)

Adds the given wxTimeSpan to this wxTimeSpan and returns a reference to itself.

static wxTimeSpan wxTimeSpan::Day ( ) `[static]`

Returns the timespan for one day.

static wxTimeSpan wxTimeSpan::Days (long *days*) `[static]`

Returns the timespan for the given number of days.

wxString wxTimeSpan::Format (const wxString & *format* = `wxDefaultTimeSpanFormat`) const

Returns the string containing the formatted representation of the time span.

The following format specifiers are allowed after %:

- `H` - Number of Hours
- `M` - Number of Minutes

- `S` - Number of Seconds
- `l` - Number of Milliseconds
- `D` - Number of Days
- `E` - Number of Weeks
- `%` - The percent character

Note that, for example, the number of hours in the description above is not well defined: it can be either the total number of hours (for example, for a time span of 50 hours this would be 50) or just the hour part of the time span, which would be 2 in this case as 50 hours is equal to 2 days and 2 hours.

wxTimeSpan resolves this ambiguity in the following way: if there had been, indeed, the `D` format specified preceding the `H`, then it is interpreted as 2. Otherwise, it is 50.

The same applies to all other format specifiers: if they follow a specifier of larger unit, only the rest part is taken, otherwise the full value is used.

int wxTimeSpan::GetDays ( ) const

Returns the difference in number of days.

int wxTimeSpan::GetHours ( ) const

Returns the difference in number of hours.

wxLongLong wxTimeSpan::GetMilliseconds ( ) const

Returns the difference in number of milliseconds.

int wxTimeSpan::GetMinutes ( ) const

Returns the difference in number of minutes.

wxLongLong wxTimeSpan::GetSeconds ( ) const

Returns the difference in number of seconds.

wxLongLong wxTimeSpan::GetValue ( ) const

Returns the internal representation of timespan.

int wxTimeSpan::GetWeeks ( ) const

Returns the difference in number of weeks.

static wxTimeSpan wxTimeSpan::Hour ( ) [static]

Returns the timespan for one hour.

static wxTimeSpan wxTimeSpan::Hours (long *hours*) `[static]`

Returns the timespan for the given number of hours.

bool wxTimeSpan::IsEqualTo (const wxTimeSpan & *ts*) const

Returns true if two timespans are equal.

bool wxTimeSpan::IsLongerThan (const wxTimeSpan & *ts*) const

Compares two timespans: works with the absolute values, i.e. -2 hours is longer than 1 hour.
Also, it will return false if the timespans are equal in absolute value.

bool wxTimeSpan::IsNegative ( ) const

Returns true if the timespan is negative.

bool wxTimeSpan::IsNull ( ) const

Returns true if the timespan is empty.

bool wxTimeSpan::IsPositive ( ) const

Returns true if the timespan is positive.

bool wxTimeSpan::IsShorterThan (const wxTimeSpan & *ts*) const

Compares two timespans: works with the absolute values, i.e. 1 hour is shorter than -2 hours.
Also, it will return false if the timespans are equal in absolute value.

static wxTimeSpan wxTimeSpan::Millisecond ( ) `[static]`

Returns the timespan for one millisecond.

static wxTimeSpan wxTimeSpan::Milliseconds (wxLongLong *ms*) `[static]`

Returns the timespan for the given number of milliseconds.

static wxTimeSpan wxTimeSpan::Minute ( ) `[static]`

Returns the timespan for one minute.

static wxTimeSpan wxTimeSpan::Minutes (long *min*) `[static]`

Returns the timespan for the given number of minutes.

wxTimeSpan wxTimeSpan::Multiply (int *n*) const

Returns the product of this time span by *n*.

Returns

A new wxTimeSpan object with the result.

wxTimeSpan& wxTimeSpan::Multiply (int *n*)

Multiplies this time span by *n*.

Returns

A reference to this wxTimeSpan object modified in place.

wxTimeSpan& wxTimeSpan::Neg ( )

Negate the value of the timespan.

See also

Negate()

wxTimeSpan wxTimeSpan::Negate ( ) const

Returns timespan with inverted sign.

See also

Neg()

wxTimeSpan& wxTimeSpan::operator*= (int *n*)

Multiplies this time span by *n*.

Returns

A reference to this wxTimeSpan object modified in place.

wxTimeSpan& wxTimeSpan::operator+= (const wxTimeSpan & *diff*)

Adds the given wxTimeSpan to this wxTimeSpan and returns the result.

wxTimeSpan& wxTimeSpan::operator- ( )

Negate the value of the timespan.

See also

Negate()

wxTimeSpan& wxTimeSpan::operator-= (const wxTimeSpan & *diff*)

Subtracts the given wxTimeSpan to this wxTimeSpan and returns the result.

static wxTimeSpan wxTimeSpan::Second ( ) [static]

Returns the timespan for one second.

static wxTimeSpan wxTimeSpan::Seconds (wxLongLong *sec*) [static]

Returns the timespan for the given number of seconds.

wxTimeSpan wxTimeSpan::Subtract (const wxTimeSpan & *diff*) const

Returns the difference of two time spans.

Returns

A new wxDateSpan object with the result.

wxTimeSpan& wxTimeSpan::Subtract (const wxTimeSpan & *diff*)

Subtracts the given wxTimeSpan to this wxTimeSpan and returns a reference to itself.

static wxTimeSpan wxTimeSpan::Week ( ) [static]

Returns the timespan for one week.

static wxTimeSpan wxTimeSpan::Weeks (long *weeks*) [static]

Returns the timespan for the given number of weeks.

21.793 wxTipProvider Class Reference

```
#include <wx/tipdlg.h>
```

21.793.1 Detailed Description

This is the class used together with wxShowTip() function.

It must implement wxTipProvider::GetTip function and return the current tip from it (different tip each time it is called).

You will never use this class yourself, but you need it to show startup tips with wxShowTip. Also, if you want to get the tips text from elsewhere than a simple text file, you will want to derive a new class from wxTipProvider and use it instead of the one returned by wxCreateFileTipProvider().

Library: wxAdvanced

Category: Miscellaneous

See also

wxTipProvider Overview, wxShowTip

Public Member Functions

- wxTipProvider (size_t currentTip)

  Constructor.

- virtual ~wxTipProvider ()
- size_t GetCurrentTip () const

  Returns the index of the current tip (i.e. the one which would be returned by GetTip()).

- virtual wxString GetTip ()=0

  Return the text of the current tip and pass to the next one.

21.793.2 Constructor & Destructor Documentation

wxTipProvider::wxTipProvider (size_t *currentTip*)

Constructor.

Parameters

currentTip	The starting tip index.

virtual wxTipProvider::~wxTipProvider ( ) [virtual]

21.793.3 Member Function Documentation

size_t wxTipProvider::GetCurrentTip ( ) const

Returns the index of the current tip (i.e. the one which would be returned by GetTip()).

The program usually remembers the value returned by this function after calling wxShowTip(). Note that it is not the same as the value which was passed to wxShowTip + 1 because the user might have pressed the "Next" button in the tip dialog.

virtual wxString wxTipProvider::GetTip ( ) [pure virtual]

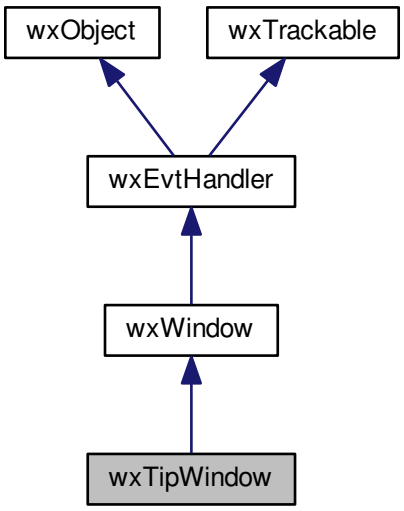
Return the text of the current tip and pass to the next one.

This function is pure virtual, it should be implemented in the derived classes.

21.794 wxTipWindow Class Reference

```
#include <wx/tipwin.h>
```

Inheritance diagram for wxTipWindow:



21.794.1 Detailed Description

Shows simple text in a popup tip window on creation.

This is used by wxSimpleHelpProvider to show popup help. The window automatically destroys itself when the user clicks on it or it loses the focus.

You may also use this class to emulate the tooltips when you need finer control over them than what the standard tooltips provide.

Library: wxCore

Category: Managed Windows

Public Member Functions

- wxTipWindow (wxWindow *parent, const wxString &text, wxCoord maxLength=100, wxTipWindow **windowPtr=NULL, wxRect *rectBounds=NULL)

  Constructor.

- void SetBoundingRect (const wxRect &rectBound)

  By default, the tip window disappears when the user clicks the mouse or presses a keyboard key or if it loses focus in any other way - for example because the user switched to another application window.

- void SetTipWindowPtr (wxTipWindow **windowPtr)

  When the tip window closes itself (which may happen at any moment and unexpectedly to the caller) it may NULL out the pointer pointed to by windowPtr.

Additional Inherited Members

21.794.2 Constructor & Destructor Documentation

wxTipWindow::wxTipWindow (wxWindow * *parent,* const wxString & *text,* wxCoord *maxLength* = 100, wxTipWindow ** *windowPtr* = NULL, wxRect * *rectBounds* = NULL)

Constructor.

The tip is shown immediately after the window is constructed.

Parameters

parent	The parent window, must be non-NULL
text	The text to show, may contain the new line characters
maxLength	The length of each line, in pixels. Set to a very large value to avoid wrapping lines
windowPtr	Simply passed to SetTipWindowPtr() below, please see its documentation for the description of this parameter
rectBounds	If non-NULL, passed to SetBoundingRect() below, please see its documentation for the description of this parameter

21.794.3 Member Function Documentation

void wxTipWindow::SetBoundingRect (const wxRect & *rectBound*)

By default, the tip window disappears when the user clicks the mouse or presses a keyboard key or if it loses focus in any other way - for example because the user switched to another application window.

Additionally, if a non-empty *rectBound* is provided, the tip window will also automatically close if the mouse leaves this area. This is useful to dismiss the tip mouse when the mouse leaves the object it is associated with.

Parameters

rectBound	The bounding rectangle for the mouse in the screen coordinates

void wxTipWindow::SetTipWindowPtr (wxTipWindow ** *windowPtr*)

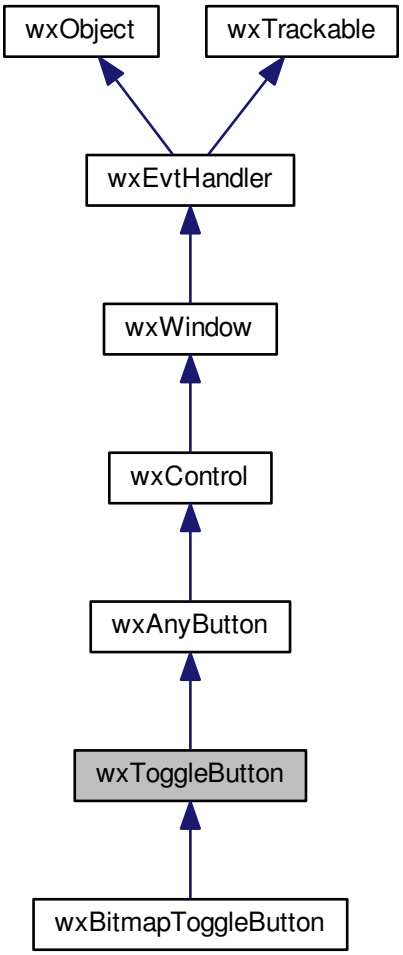
When the tip window closes itself (which may happen at any moment and unexpectedly to the caller) it may NULL out the pointer pointed to by *windowPtr*.

This is helpful to avoid dereferencing the tip window which had been already closed and deleted.

21.795 wxToggleButton Class Reference

`#include <wx/tglbtn.h>`

Inheritance diagram for wxToggleButton:



21.795.1 Detailed Description

wxToggleButton is a button that stays pressed when clicked by the user.

In other words, it is similar to wxCheckBox in functionality but looks like a wxButton.

Since wxWidgets version 2.9.0 this control emits an update UI event.

You can see wxToggleButton in action in Controls Sample.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_TOGGLEBUTTON(id, func): Handles a wxEVT_TOGGLEBUTTON event.

Library: wxCore

Category: Controls

See also

wxCheckBox, wxButton, wxBitmapToggleButton

Public Member Functions

- wxToggleButton ()

  Default constructor.
- wxToggleButton (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &val=wxDefaultValidator, const wxString &name=wxCheckBoxNameStr)

  Constructor, creating and showing a toggle button.
- virtual ~wxToggleButton ()

  Destructor, destroying the toggle button.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &label, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &val=wxDefaultValidator, const wxString &name=wxCheckBoxNameStr)

  Creates the toggle button for two-step construction.
- virtual bool GetValue () const

  Gets the state of the toggle button.
- virtual void SetValue (bool state)

  Sets the toggle button to the given state.

Additional Inherited Members

21.795.2 Constructor & Destructor Documentation

wxToggleButton::wxToggleButton ( )

Default constructor.

wxToggleButton::wxToggleButton (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxValidator & *val* = wxDefaultValidator, const wxString & *name* = `wxCheckBoxNameStr`)

Constructor, creating and showing a toggle button.

Parameters

parent	Parent window. Must not be NULL.
id	Toggle button identifier. The value wxID_ANY indicates a default value.
label	Text to be displayed next to the toggle button.
pos	Toggle button position. If wxDefaultPosition is specified then a default position is chosen.
size	Toggle button size. If wxDefaultSize is specified then a default size is chosen.

style	Window style. See wxToggleButton.
val	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxToggleButton::~wxToggleButton ( ) [virtual]

Destructor, destroying the toggle button.

21.795.3 Member Function Documentation

bool wxToggleButton::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *label*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *val* = wxDefaultValidator, const wxString & *name* = wxCheckBoxNameStr)

Creates the toggle button for two-step construction.

See wxToggleButton() for details.

virtual bool wxToggleButton::GetValue ( ) const [virtual]

Gets the state of the toggle button.

Returns

Returns true if it is pressed, false otherwise.

Reimplemented in wxBitmapToggleButton.

virtual void wxToggleButton::SetValue (bool *state*) [virtual]

Sets the toggle button to the given state.

This does not cause a EVT_TOGGLEBUTTON event to be emitted.

Parameters

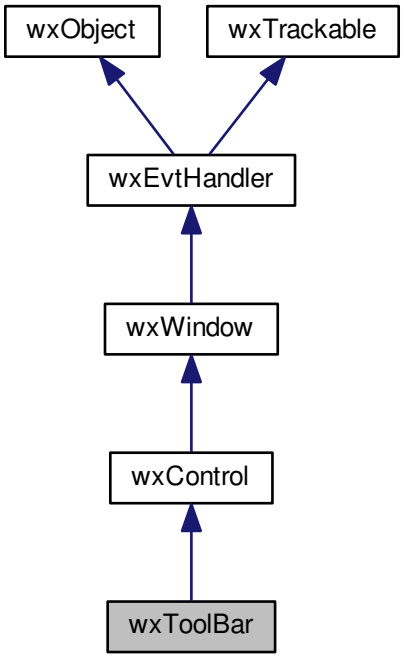
state	If true, the button is pressed.

Reimplemented in wxBitmapToggleButton.

21.796 wxToolBar Class Reference

```
#include <wx/toolbar.h>
```

Inheritance diagram for wxToolBar:



21.796.1 Detailed Description

A toolbar is a bar of buttons and/or other controls usually placed below the menu bar in a wxFrame.

You may create a toolbar that is managed by a frame calling wxFrame::CreateToolBar(). Under Pocket PC, you should always use this function for creating the toolbar to be managed by the frame, so that wxWidgets can use a combined menubar and toolbar. Where you manage your own toolbars, create wxToolBar as usual.

There are several different types of tools you can add to a toolbar. These types are controlled by the wxItemKind enumeration.

Note that many methods in wxToolBar such as wxToolBar::AddTool return a `wxToolBarToolBase*` object. This should be regarded as an opaque handle representing the newly added toolbar item, providing access to its id and position within the toolbar. Changes to the item's state should be made through calls to wxToolBar methods, for example wxToolBar::EnableTool. Calls to `wxToolBarToolBase` methods (undocumented by purpose) will not change the visible state of the item within the tool bar.

wxMSW note: Note that under wxMSW toolbar paints tools to reflect system-wide colours. If you use more than 16 colours in your tool bitmaps, you may wish to suppress this behaviour, otherwise system colours in your bitmaps will inadvertently be mapped to system colours. To do this, set the msw.remap system option before creating the toolbar:

```
wxSystemOptions::SetOption("msw.remap", 0);
```

If you wish to use 32-bit images (which include an alpha channel for transparency) use:

```
wxSystemOptions::SetOption("msw.remap", 2);
```

Then colour remapping is switched off, and a transparent background used. But only use this option under Windows XP with true colour:

```
if (wxTheApp->GetComCtl32Version() >= 600 && ::wxDisplayDepth() >= 32)
```

Styles

This class supports the following styles:

- wxTB_FLAT: Gives the toolbar a flat look (Windows and GTK only).
- wxTB_DOCKABLE: Makes the toolbar floatable and dockable (GTK only).
- wxTB_HORIZONTAL: Specifies horizontal layout (default).
- wxTB_VERTICAL: Specifies vertical layout.
- wxTB_TEXT: Shows the text in the toolbar buttons; by default only icons are shown.
- wxTB_NOICONS: Specifies no icons in the toolbar buttons; by default they are shown.
- wxTB_NODIVIDER: Specifies no divider (border) above the toolbar (Windows only)
- wxTB_NOALIGN: Specifies no alignment with the parent window (Windows only, not very useful).
- wxTB_HORZ_LAYOUT: Shows the text and the icons alongside, not vertically stacked (Windows and GTK 2 only). This style must be used with `wxTB_TEXT`.
- wxTB_HORZ_TEXT: Combination of `wxTB_HORZ_LAYOUT` and `wxTB_TEXT`.
- wxTB_NO_TOOLTIPS: Don't show the short help tooltips for the tools when the mouse hovers over them.
- wxTB_BOTTOM: Align the toolbar at the bottom of parent window.
- wxTB_RIGHT: Align the toolbar at the right side of parent window.
- wxTB_DEFAULT_STYLE: Combination of `wxTB_HORIZONTAL` and `wxTB_FLAT`. This style is new since wxWidgets 2.9.5.

See also Window Styles. Note that the wxMSW native toolbar ignores `wxTB_NOICONS` style. Also, toggling the `wxTB_TEXT` works only if the style was initially on.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxCommandEvent& event)

Event macros for events emitted by this class:

- EVT_TOOL(id, func): Process a `wxEVT_TOOL` event (a synonym for `wxEVT_MENU`). Pass the id of the tool.
- EVT_MENU(id, func): The same as EVT_TOOL().
- EVT_TOOL_RANGE(id1, id2, func): Process a `wxEVT_TOOL` event for a range of identifiers. Pass the ids of the tools.
- EVT_MENU_RANGE(id1, id2, func): The same as EVT_TOOL_RANGE().
- EVT_TOOL_RCLICKED(id, func): Process a `wxEVT_TOOL_RCLICKED` event. Pass the id of the tool. (Not available on wxOSX.)
- EVT_TOOL_RCLICKED_RANGE(id1, id2, func): Process a `wxEVT_TOOL_RCLICKED` event for a range of ids. Pass the ids of the tools. (Not available on wxOSX.)
- EVT_TOOL_ENTER(id, func): Process a `wxEVT_TOOL_ENTER` event. Pass the id of the toolbar itself. The value of wxCommandEvent::GetSelection() is the tool id, or -1 if the mouse cursor has moved off a tool. (Not available on wxOSX.)

- EVT_TOOL_DROPDOWN(id, func): Process a `wxEVT_TOOL_DROPDOWN` event. If unhandled, displays the default dropdown menu set using wxToolBar::SetDropdownMenu().

The toolbar class emits menu commands in the same way that a frame menubar does, so you can use one EVT_MENU() macro for both a menu item and a toolbar button. The event handler functions take a wxCommandEvent argument. For most event macros, the identifier of the tool is passed, but for EVT_TOOL_ENTER() the toolbar window identifier is passed and the tool identifier is retrieved from the wxCommandEvent. This is because the identifier may be `wxID_ANY` when the mouse moves off a tool, and `wxID_ANY` is not allowed as an identifier in the event system.

Library: wxCore

Category: Miscellaneous Windows

See also

Toolbar Overview

Public Member Functions

- wxToolBar ()

  Default constructor.
- wxToolBar (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTB_HORIZONTAL, const wxString &name=wxToolBarNameStr)

  Constructs a toolbar.
- virtual ~wxToolBar ()

  Toolbar destructor.
- wxToolBarToolBase * AddCheckTool (int toolId, const wxString &label, const wxBitmap &bitmap1, const wxBitmap &bmpDisabled=wxNullBitmap, const wxString &shortHelp=wxEmptyString, const wxString &longHelp=wxEmptyString, wxObject *clientData=NULL)

  Adds a new check (or toggle) tool to the toolbar.
- virtual wxToolBarToolBase * AddControl (wxControl *control, const wxString &label=wxEmptyString)

  Adds any control to the toolbar, typically e.g. a wxComboBox.
- wxToolBarToolBase * AddRadioTool (int toolId, const wxString &label, const wxBitmap &bitmap1, const wxBitmap &bmpDisabled=wxNullBitmap, const wxString &shortHelp=wxEmptyString, const wxString &longHelp=wxEmptyString, wxObject *clientData=NULL)

  Adds a new radio tool to the toolbar.
- virtual wxToolBarToolBase * AddSeparator ()

  Adds a separator for spacing groups of tools.
- wxToolBarToolBase * AddStretchableSpace ()

  Adds a stretchable space to the toolbar.
- virtual void ClearTools ()

  Deletes all the tools in the toolbar.
- virtual bool DeleteTool (int toolId)

  Removes the specified tool from the toolbar and deletes it.
- virtual bool DeleteToolByPos (size_t pos)

  This function behaves like DeleteTool() but it deletes the tool at the specified position and not the one with the given id.
- virtual void EnableTool (int toolId, bool enable)

  Enables or disables the tool.
- wxToolBarToolBase * FindById (int id) const

  Returns a pointer to the tool identified by id or NULL if no corresponding tool is found.

- virtual wxControl * FindControl (int id)

  Returns a pointer to the control identified by id or NULL if no corresponding control is found.
- virtual wxToolBarToolBase * FindToolForPosition (wxCoord x, wxCoord y) const

  Finds a tool for the given mouse position.
- wxSize GetMargins () const

  Returns the left/right and top/bottom margins, which are also used for inter-toolspacing.
- virtual wxSize GetToolBitmapSize () const

  Returns the size of bitmap that the toolbar expects to have.
- const wxToolBarToolBase * GetToolByPos (int pos) const

  Returns a pointer to the tool at ordinal position pos.
- virtual wxObject * GetToolClientData (int toolId) const

  Get any client data associated with the tool.
- virtual bool GetToolEnabled (int toolId) const

  Called to determine whether a tool is enabled (responds to user input).
- virtual wxString GetToolLongHelp (int toolId) const

  Returns the long help for the given tool.
- virtual int GetToolPacking () const

  Returns the value used for packing tools.
- virtual int GetToolPos (int toolId) const

  Returns the tool position in the toolbar, or `wxNOT_FOUND` if the tool is not found.
- virtual int GetToolSeparation () const

  Returns the default separator size.
- virtual wxString GetToolShortHelp (int toolId) const

  Returns the short help for the given tool.
- virtual wxSize GetToolSize () const

  Returns the size of a whole button, which is usually larger than a tool bitmap because of added 3D effects.
- virtual bool GetToolState (int toolId) const

  Gets the on/off state of a toggle tool.
- size_t GetToolsCount () const

  Returns the number of tools in the toolbar.
- virtual wxToolBarToolBase * InsertControl (size_t pos, wxControl *control, const wxString &label=wxEmptyString)

  Inserts the control into the toolbar at the given position.
- virtual wxToolBarToolBase * InsertSeparator (size_t pos)

  Inserts the separator into the toolbar at the given position.
- wxToolBarToolBase * InsertStretchableSpace (size_t pos)

  Inserts a stretchable space at the given position.
- virtual bool OnLeftClick (int toolId, bool toggleDown)

  Called when the user clicks on a tool with the left mouse button.
- virtual void OnMouseEnter (int toolId)

  This is called when the mouse cursor moves into a tool or out of the toolbar.
- virtual void OnRightClick (int toolId, long x, long y)
- virtual bool Realize ()

  This function should be called after you have added tools.
- virtual wxToolBarToolBase * RemoveTool (int id)

  Removes the given tool from the toolbar but doesn't delete it.
- void SetBitmapResource (int resourceId)

  Sets the bitmap resource identifier for specifying tool bitmaps as indices into a custom bitmap.
- bool SetDropdownMenu (int id, wxMenu *menu)

  Sets the dropdown menu for the tool given by its id.
- virtual void SetToolBitmapSize (const wxSize &size)

Sets the default size of each tool bitmap.

- virtual void SetToolClientData (int id, wxObject *clientData)

  Sets the client data associated with the tool.
- virtual void SetToolDisabledBitmap (int id, const wxBitmap &bitmap)

  Sets the bitmap to be used by the tool with the given ID when the tool is in a disabled state.
- virtual void SetToolLongHelp (int toolId, const wxString &helpString)

  Sets the long help for the given tool.
- virtual void SetToolNormalBitmap (int id, const wxBitmap &bitmap)

  Sets the bitmap to be used by the tool with the given ID.
- virtual void SetToolPacking (int packing)

  Sets the value used for spacing tools.
- virtual void SetToolSeparation (int separation)

  Sets the default separator size.
- virtual void SetToolShortHelp (int toolId, const wxString &helpString)

  Sets the short help for the given tool.
- virtual void ToggleTool (int toolId, bool toggle)

  Toggles a tool on or off.
- virtual wxToolBarToolBase * CreateTool (int toolId, const wxString &label, const wxBitmap &bmpNormal, const wxBitmap &bmpDisabled=wxNullBitmap, wxItemKind kind=wxITEM_NORMAL, wxObject *clientData=NULL, const wxString &shortHelp=wxEmptyString, const wxString &longHelp=wxEmptyString)

  Factory function to create a new toolbar tool.
- virtual wxToolBarToolBase * CreateTool (wxControl *control, const wxString &label)

  Factory function to create a new control toolbar tool.

- virtual wxToolBarToolBase * AddTool (wxToolBarToolBase *tool)

  Adds a tool to the toolbar.
- wxToolBarToolBase * AddTool (int toolId, const wxString &label, const wxBitmap &bitmap, const wxString &shortHelp=wxEmptyString, wxItemKind kind=wxITEM_NORMAL)

  Adds a tool to the toolbar.
- wxToolBarToolBase * AddTool (int toolId, const wxString &label, const wxBitmap &bitmap, const wxBitmap &bmpDisabled, wxItemKind kind=wxITEM_NORMAL, const wxString &shortHelpString=wxEmptyString, const wxString &longHelpString=wxEmptyString, wxObject *clientData=NULL)

  Adds a tool to the toolbar.

- wxToolBarToolBase * InsertTool (size_t pos, int toolId, const wxString &label, const wxBitmap &bitmap, const wxBitmap &bmpDisabled=wxNullBitmap, wxItemKind kind=wxITEM_NORMAL, const wxString &shortHelp=wxEmptyString, const wxString &longHelp=wxEmptyString, wxObject *clientData=NULL)

  Inserts the tool with the specified attributes into the toolbar at the given position.
- wxToolBarToolBase * InsertTool (size_t pos, wxToolBarToolBase *tool)

  Inserts the tool with the specified attributes into the toolbar at the given position.

- virtual void SetMargins (int x, int y)

  Set the values to be used as margins for the toolbar.
- void SetMargins (const wxSize &size)

  Set the margins for the toolbar.

Additional Inherited Members

21.796.2 Constructor & Destructor Documentation

wxToolBar::wxToolBar ( )

Default constructor.

wxToolBar::wxToolBar (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTB_HORIZONTAL, const wxString & *name* = `wxToolBarNameStr`)

Constructs a toolbar.

Parameters

parent	Pointer to a parent window.
id	Window identifier. If -1, will automatically create an identifier.
pos	Window position. wxDefaultPosition indicates that wxWidgets should generate a default position for the window. If using the wxWindow class directly, supply an actual position.
size	Window size. wxDefaultSize indicates that wxWidgets should generate a default size for the window.
style	Window style. See wxToolBar initial description for details.
name	Window name.

Remarks

After a toolbar is created, you use AddTool() and perhaps AddSeparator(), and then you must call Realize() to construct and display the toolbar tools.

virtual wxToolBar::~wxToolBar ( ) [virtual]

Toolbar destructor.

21.796.3 Member Function Documentation

wxToolBarToolBase* wxToolBar::AddCheckTool (int *toolId*, const wxString & *label*, const wxBitmap & *bitmap1*, const wxBitmap & *bmpDisabled* = wxNullBitmap, const wxString & *shortHelp* = wxEmptyString, const wxString & *longHelp* = wxEmptyString, wxObject * *clientData* = NULL)

Adds a new check (or toggle) tool to the toolbar.

The parameters are the same as in AddTool().

See also

AddTool()

virtual wxToolBarToolBase* wxToolBar::AddControl (wxControl * *control*, const wxString & *label* = wxEmptyString) [virtual]

Adds any control to the toolbar, typically e.g. a wxComboBox.

Parameters

control	The control to be added.
label	Text to be displayed near the control.

Remarks

wxMSW: the label is only displayed if there is enough space available below the embedded control.
wxMac: labels are only displayed if wxWidgets is built with wxMAC_USE_NATIVE_TOOLBAR set to 1

wxToolBarToolBase* wxToolBar::AddRadioTool (int *toolId*, const wxString & *label*, const wxBitmap & *bitmap1*, const wxBitmap & *bmpDisabled* = wxNullBitmap, const wxString & *shortHelp* = wxEmptyString, const wxString & *longHelp* = wxEmptyString, wxObject * *clientData* = NULL)

Adds a new radio tool to the toolbar.

Consecutive radio tools form a radio group such that exactly one button in the group is pressed at any moment, in other words whenever a button in the group is pressed the previously pressed button is automatically released. You should avoid having the radio groups of only one element as it would be impossible for the user to use such button.

By default, the first button in the radio group is initially pressed, the others are not.

See also

AddTool()

virtual wxToolBarToolBase* wxToolBar::AddSeparator ( ) [virtual]

Adds a separator for spacing groups of tools.

Notice that the separator uses the look appropriate for the current platform so it can be a vertical line (MSW, some versions of GTK) or just an empty space or something else.

See also

AddTool(), SetToolSeparation(), AddStretchableSpace()

wxToolBarToolBase* wxToolBar::AddStretchableSpace ( )

Adds a stretchable space to the toolbar.

Any space not taken up by the fixed items (all items except for stretchable spaces) is distributed in equal measure between the stretchable spaces in the toolbar. The most common use for this method is to add a single stretchable space before the items which should be right-aligned in the toolbar, but more exotic possibilities are possible, e.g. a stretchable space may be added in the beginning and the end of the toolbar to centre all toolbar items.

See also

AddTool(), AddSeparator(), InsertStretchableSpace()

Since

2.9.1

virtual wxToolBarToolBase* wxToolBar::AddTool (wxToolBarToolBase * *tool*) [virtual]

Adds a tool to the toolbar.

Parameters

tool	The tool to be added.

Remarks

After you have added tools to a toolbar, you must call Realize() in order to have the tools appear.

See also

AddSeparator(), AddCheckTool(), AddRadioTool(), InsertTool(), DeleteTool(), Realize(), SetDropdownMenu()

wxToolBarToolBase* **wxToolBar::AddTool** (**int** *toolId*, **const wxString &** *label*, **const wxBitmap &** *bitmap*, **const wxString &** *shortHelp* = **wxEmptyString**, **wxItemKind** *kind* = **wxITEM_NORMAL**)

Adds a tool to the toolbar.

This most commonly used version has fewer parameters than the full version below which specifies the more rarely used button features.

Parameters

toolId	An integer by which the tool may be identified in subsequent operations.
label	The string to be displayed with the tool.
bitmap	The primary tool bitmap.
shortHelp	This string is used for the tools tooltip.
kind	May be wxITEM_NORMAL for a normal button (default), wxITEM_CHECK for a checkable tool (such tool stays pressed after it had been toggled) or wxITEM_RADIO for a checkable tool which makes part of a radio group of tools each of which is automatically unchecked whenever another button in the group is checked. wxITEM_DROPDOWN specifies that a drop-down menu button will appear next to the tool button (only GTK+ and MSW). Call SetDropdownMenu() afterwards.

Remarks

After you have added tools to a toolbar, you must call Realize() in order to have the tools appear.

See also

AddSeparator(), AddCheckTool(), AddRadioTool(), InsertTool(), DeleteTool(), Realize(), SetDropdownMenu()

wxToolBarToolBase* wxToolBar::AddTool (int *toolId*, const wxString & *label*, const wxBitmap & *bitmap*, const wxBitmap & *bmpDisabled*, wxItemKind *kind* = wxITEM_NORMAL, const wxString & *shortHelpString* = wxEmptyString, const wxString & *longHelpString* = wxEmptyString, wxObject * *clientData* = NULL)

Adds a tool to the toolbar.

Parameters

toolId	An integer by which the tool may be identified in subsequent operations.
label	The string to be displayed with the tool.
bitmap	The primary tool bitmap.
bmpDisabled	The bitmap used when the tool is disabled. If it is equal to wxNullBitmap (default), the disabled bitmap is automatically generated by greying the normal one.
kind	May be wxITEM_NORMAL for a normal button (default), wxITEM_CHECK for a checkable tool (such tool stays pressed after it had been toggled) or wxITEM_RADIO for a checkable tool which makes part of a radio group of tools each of which is automatically unchecked whenever another button in the group is checked. wxITEM_DROPDOWN specifies that a drop-down menu button will appear next to the tool button (only GTK+ and MSW). Call SetDropdownMenu() afterwards.
shortHelpString	This string is used for the tools tooltip.
longHelpString	This string is shown in the statusbar (if any) of the parent frame when the mouse pointer is inside the tool.
clientData	An optional pointer to client data which can be retrieved later using GetToolClientData().

Remarks

After you have added tools to a toolbar, you must call Realize() in order to have the tools appear.

See also

AddSeparator(), AddCheckTool(), AddRadioTool(), InsertTool(), DeleteTool(), Realize(), SetDropdownMenu()

virtual void wxToolBar::ClearTools ( ) [virtual]

Deletes all the tools in the toolbar.

virtual **wxToolBarToolBase**∗ wxToolBar::CreateTool (int *toolId,* const **wxString** & *label,* const **wxBitmap** & *bmpNormal,* const **wxBitmap** & *bmpDisabled* = **wxNullBitmap**, **wxItemKind** *kind* = **wxITEM_NORMAL**, **wxObject** ∗ *clientData* = NULL, const **wxString** & *shortHelp* = **wxEmptyString**, const **wxString** & *longHelp* = **wxEmptyString**) [virtual]

Factory function to create a new toolbar tool.

virtual **wxToolBarToolBase**∗ wxToolBar::CreateTool (**wxControl** ∗ *control,* const **wxString** & *label*) [virtual]

Factory function to create a new control toolbar tool.

virtual bool wxToolBar::DeleteTool (int *toolId*) [virtual]

Removes the specified tool from the toolbar and deletes it.

If you don't want to delete the tool, but just to remove it from the toolbar (to possibly add it back later), you may use RemoveTool() instead.

Note

It is unnecessary to call Realize() for the change to take place, it will happen immediately.

Returns

true if the tool was deleted, false otherwise.

See also

DeleteToolByPos()

virtual bool wxToolBar::DeleteToolByPos (size_t *pos*) [virtual]

This function behaves like DeleteTool() but it deletes the tool at the specified position and not the one with the given id.

virtual void wxToolBar::EnableTool (int *toolId,* bool *enable*) [virtual]

Enables or disables the tool.

Parameters

toolId	ID of the tool to enable or disable, as passed to AddTool().
enable	If true, enables the tool, otherwise disables it.

Remarks

Some implementations will change the visible state of the tool to indicate that it is disabled.

See also

GetToolEnabled(), ToggleTool()

wxToolBarToolBase∗ wxToolBar::FindById (int *id*) const

Returns a pointer to the tool identified by *id* or NULL if no corresponding tool is found.

virtual wxControl* wxToolBar::FindControl (int *id*) [virtual]

Returns a pointer to the control identified by *id* or NULL if no corresponding control is found.

virtual wxToolBarToolBase* wxToolBar::FindToolForPosition (wxCoord *x*, wxCoord *y*) const [virtual]

Finds a tool for the given mouse position.

Parameters

x	X position.
y	Y position.

Returns

A pointer to a tool if a tool is found, or NULL otherwise.

Remarks

Currently not implemented in wxGTK (always returns NULL there).

wxSize wxToolBar::GetMargins ( ) const

Returns the left/right and top/bottom margins, which are also used for inter-toolspacing.

See also

SetMargins()

virtual wxSize wxToolBar::GetToolBitmapSize ( ) const [virtual]

Returns the size of bitmap that the toolbar expects to have.

The default bitmap size is platform-dependent: for example, it is 16*15 for MSW and 24*24 for GTK. This size does *not* necessarily indicate the best size to use for the toolbars on the given platform, for this you should use `wx↩ ArtProvider::GetNativeSizeHint(wxART_TOOLBAR)` but in any case, as the bitmap size is deduced automatically from the size of the bitmaps associated with the tools added to the toolbar, it is usually unnecessary to call SetToolBitmapSize() explicitly.

Remarks

Note that this is the size of the bitmap you pass to AddTool(), and not the eventual size of the tool button.

See also

SetToolBitmapSize(), GetToolSize()

const wxToolBarToolBase* wxToolBar::GetToolByPos (int *pos*) const

Returns a pointer to the tool at ordinal position *pos*.

Don't confuse this with FindToolForPosition().

Since

2.9.1

See also

GetToolsCount()

virtual wxObject* wxToolBar::GetToolClientData (int *toolId*) const `[virtual]`

Get any client data associated with the tool.

Parameters

toolId	ID of the tool in question, as passed to AddTool().

Returns

Client data, or NULL if there is none.

virtual bool wxToolBar::GetToolEnabled (int *toolId*) const `[virtual]`

Called to determine whether a tool is enabled (responds to user input).

Parameters

toolId	ID of the tool in question, as passed to AddTool().

Returns

true if the tool is enabled, false otherwise.

See also

EnableTool()

virtual wxString wxToolBar::GetToolLongHelp (int *toolId*) const `[virtual]`

Returns the long help for the given tool.

Parameters

toolId	ID of the tool in question, as passed to AddTool().

See also

SetToolLongHelp(), SetToolShortHelp()

virtual int wxToolBar::GetToolPacking ( ) const `[virtual]`

Returns the value used for packing tools.

See also

SetToolPacking()

virtual int wxToolBar::GetToolPos (int *toolId*) const `[virtual]`

Returns the tool position in the toolbar, or `wxNOT_FOUND` if the tool is not found.

Parameters

toolId	ID of the tool in question, as passed to AddTool().

size_t wxToolBar::GetToolsCount ( ) const

Returns the number of tools in the toolbar.

virtual int wxToolBar::GetToolSeparation ( ) const `[virtual]`

Returns the default separator size.

See also

SetToolSeparation()

virtual wxString wxToolBar::GetToolShortHelp (int *toolId*) const `[virtual]`

Returns the short help for the given tool.

Parameters

toolId	ID of the tool in question, as passed to AddTool().

See also

GetToolLongHelp(), SetToolShortHelp()

virtual wxSize wxToolBar::GetToolSize ( ) const `[virtual]`

Returns the size of a whole button, which is usually larger than a tool bitmap because of added 3D effects.

See also

SetToolBitmapSize(), GetToolBitmapSize()

virtual bool wxToolBar::GetToolState (int *toolId*) const `[virtual]`

Gets the on/off state of a toggle tool.

Parameters

toolId	ID of the tool in question, as passed to AddTool().

Returns

true if the tool is toggled on, false otherwise.

See also

ToggleTool()

virtual **wxToolBarToolBase**∗ **wxToolBar::InsertControl** (**size_t** *pos*, **wxControl** ∗ *control*, **const wxString &** *label* = **wxEmptyString**) [virtual]

Inserts the control into the toolbar at the given position.

You must call Realize() for the change to take place.

See also

AddControl(), InsertTool()

virtual **wxToolBarToolBase**∗ **wxToolBar::InsertSeparator** (**size_t** *pos*) [virtual]

Inserts the separator into the toolbar at the given position.

You must call Realize() for the change to take place.

See also

AddSeparator(), InsertTool()

wxToolBarToolBase∗ **wxToolBar::InsertStretchableSpace** (**size_t** *pos*)

Inserts a stretchable space at the given position.

See AddStretchableSpace() for details about stretchable spaces.

See also

InsertTool(), InsertSeparator()

Since

2.9.1

wxToolBarToolBase∗ **wxToolBar::InsertTool** (**size_t** *pos*, **int** *toolId*, **const wxString &** *label*, **const wxBitmap &** *bitmap*, **const wxBitmap &** *bmpDisabled* = **wxNullBitmap**, **wxItemKind** *kind* = **wxITEM_NORMAL**, **const wxString &** *shortHelp* = **wxEmptyString**, **const wxString &** *longHelp* = **wxEmptyString**, **wxObject** ∗ *clientData* = NULL)

Inserts the tool with the specified attributes into the toolbar at the given position.

You must call Realize() for the change to take place.

See also

AddTool(), InsertControl(), InsertSeparator()

Returns

The newly inserted tool or NULL on failure. Notice that with the overload taking *tool* parameter the caller is responsible for deleting the tool in the latter case.

wxToolBarToolBase* **wxToolBar::InsertTool (size_t *pos,* wxToolBarToolBase * *tool*)**

Inserts the tool with the specified attributes into the toolbar at the given position.

You must call Realize() for the change to take place.

See also

AddTool(), InsertControl(), InsertSeparator()

Returns

The newly inserted tool or NULL on failure. Notice that with the overload taking *tool* parameter the caller is responsible for deleting the tool in the latter case.

virtual bool wxToolBar::OnLeftClick (int *toolId,* bool *toggleDown*) `[virtual]`

Called when the user clicks on a tool with the left mouse button.

This is the old way of detecting tool clicks; although it will still work, you should use the EVT_MENU() or EVT_TOOL() macro instead.

Parameters

toolId	The identifier passed to AddTool().
toggleDown	true if the tool is a toggle and the toggle is down, otherwise is false.

Returns

If the tool is a toggle and this function returns false, the toggle state (internal and visual) will not be changed. This provides a way of specifying that toggle operations are not permitted in some circumstances.

See also

OnMouseEnter(), OnRightClick()

virtual void wxToolBar::OnMouseEnter (int *toolId*) `[virtual]`

This is called when the mouse cursor moves into a tool or out of the toolbar.

This is the old way of detecting mouse enter events; although it will still work, you should use the EVT_TOOL_ENTER() macro instead.

Parameters

toolId	Greater than -1 if the mouse cursor has moved into the tool, or -1 if the mouse cursor has moved. The programmer can override this to provide extra information about the tool, such as a short description on the status line.

Remarks

With some derived toolbar classes, if the mouse moves quickly out of the toolbar, wxWidgets may not be able to detect it. Therefore this function may not always be called when expected.

virtual void wxToolBar::OnRightClick (int *toolId,* long *x,* long *y*) `[virtual]`

Deprecated This is the old way of detecting tool right clicks; although it will still work, you should use the EVT_T↩ OOL_RCLICKED() macro instead.

Called when the user clicks on a tool with the right mouse button. The programmer should override this function to detect right tool clicks.

Parameters

toolId	The identifier passed to AddTool().
x	The x position of the mouse cursor.
y	The y position of the mouse cursor.

Remarks

A typical use of this member might be to pop up a menu.

See also

OnMouseEnter(), OnLeftClick()

virtual bool wxToolBar::Realize ( ) [virtual]

This function should be called after you have added tools.

virtual wxToolBarToolBase* wxToolBar::RemoveTool (int *id*) [virtual]

Removes the given tool from the toolbar but doesn't delete it.

This allows to insert/add this tool back to this (or another) toolbar later.

Note

It is unnecessary to call Realize() for the change to take place, it will happen immediately.

See also

DeleteTool()

void wxToolBar::SetBitmapResource (int *resourceId*)

Sets the bitmap resource identifier for specifying tool bitmaps as indices into a custom bitmap.

This is a Windows CE-specific method not available in the other ports.

Availability: only available for the wxWinCE port.

bool wxToolBar::SetDropdownMenu (int *id,* wxMenu * *menu*)

Sets the dropdown menu for the tool given by its *id*.

The tool itself will delete the menu when it's no longer needed. Only supported under GTK+ und MSW.

If you define a EVT_TOOL_DROPDOWN() handler in your program, you must call wxEvent::Skip() from it or the menu won't be displayed.

virtual void wxToolBar::SetMargins (int *x,* int *y*) [virtual]

Set the values to be used as margins for the toolbar.

Parameters

x	Left margin, right margin and inter-tool separation value.
y	Top margin, bottom margin and inter-tool separation value.

Remarks

This must be called before the tools are added if absolute positioning is to be used, and the default (zero-size) margins are to be overridden.

See also

GetMargins()

void wxToolBar::SetMargins (const wxSize & *size*)

Set the margins for the toolbar.

Parameters

size	Margin size.

Remarks

This must be called before the tools are added if absolute positioning is to be used, and the default (zero-size) margins are to be overridden.

See also

GetMargins(), wxSize

virtual void wxToolBar::SetToolBitmapSize (const wxSize & *size*) [virtual]

Sets the default size of each tool bitmap.

The default bitmap size is 16 by 15 pixels.

Parameters

size	The size of the bitmaps in the toolbar.

Remarks

This should be called to tell the toolbar what the tool bitmap size is. Call it before you add tools.

See also

GetToolBitmapSize(), GetToolSize()

virtual void wxToolBar::SetToolClientData (int *id,* wxObject * *clientData*) [virtual]

Sets the client data associated with the tool.

Parameters

id	ID of the tool in question, as passed to AddTool().
clientData	The client data to use.

virtual void wxToolBar::SetToolDisabledBitmap (int *id,* const wxBitmap & *bitmap*) [virtual]

Sets the bitmap to be used by the tool with the given ID when the tool is in a disabled state.

This can only be used on Button tools, not controls.

Parameters

id	ID of the tool in question, as passed to AddTool().
bitmap	Bitmap to use for disabled tools.

Note

The native toolbar classes on the main platforms all synthesize the disabled bitmap from the normal bitmap, so this function will have no effect on those platforms.

virtual void wxToolBar::SetToolLongHelp (int *toolId,* const wxString & *helpString*) [virtual]

Sets the long help for the given tool.

Parameters

toolId	ID of the tool in question, as passed to AddTool().
helpString	A string for the long help.

Remarks

You might use the long help for displaying the tool purpose on the status line.

See also

GetToolLongHelp(), SetToolShortHelp(),

virtual void wxToolBar::SetToolNormalBitmap (int *id,* const wxBitmap & *bitmap*) [virtual]

Sets the bitmap to be used by the tool with the given ID.

This can only be used on Button tools, not controls.

Parameters

id	ID of the tool in question, as passed to AddTool().
bitmap	Bitmap to use for normals tools.

virtual void wxToolBar::SetToolPacking (int *packing*) [virtual]

Sets the value used for spacing tools.

The default value is 1.

Parameters

packing	The value for packing.

Remarks

The packing is used for spacing in the vertical direction if the toolbar is horizontal, and for spacing in the horizontal direction if the toolbar is vertical.

See also

GetToolPacking()

virtual void wxToolBar::SetToolSeparation (int *separation*) [virtual]

Sets the default separator size.

The default value is 5.

Parameters

separation	The separator size.

See also

AddSeparator()

virtual void wxToolBar::SetToolShortHelp (int *toolId,* const wxString & *helpString*) [virtual]

Sets the short help for the given tool.

Parameters

toolId	ID of the tool in question, as passed to AddTool().
helpString	The string for the short help.

Remarks

An application might use short help for identifying the tool purpose in a tooltip.

See also

GetToolShortHelp(), SetToolLongHelp()

virtual void wxToolBar::ToggleTool (int *toolId,* bool *toggle*) [virtual]

Toggles a tool on or off.

This does not cause any event to get emitted.

Parameters

toolId	ID of the tool in question, as passed to AddTool().

toggle	If true, toggles the tool on, otherwise toggles it off.

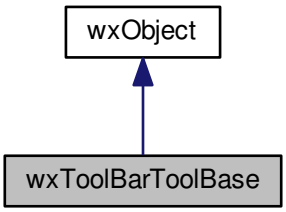
Remarks

Only applies to a tool that has been specified as a toggle tool.

21.797 wxToolBarToolBase Class Reference

`#include <wx/toolbar.h>`

Inheritance diagram for wxToolBarToolBase:



21.797.1 Detailed Description

A toolbar tool represents one item on the toolbar.

It has a unique id (except for the separators), the style (telling whether it is a normal button, separator or a control), the state (toggled or not, enabled or not) and short and long help strings. The default implementations use the short help string for the tooltip text which is popped up when the mouse pointer enters the tool and the long help string for the applications status bar.

Public Member Functions

- wxToolBarToolBase (wxToolBarBase *tbar=NULL, int toolid=wxID_SEPARATOR, const wxString &label=wxEmptyString, const wxBitmap &bmpNormal=wxNullBitmap, const wxBitmap &bmpDisabled=wxNullBitmap, wxItemKind kind=wxITEM_NORMAL, wxObject *clientData=NULL, const wxString &shortHelpString=wxEmptyString, const wxString &longHelpString=wxEmptyString)
- wxToolBarToolBase (wxToolBarBase *tbar, wxControl *control, const wxString &label)
- virtual ~wxToolBarToolBase ()
- int GetId () const
- wxControl * GetControl () const
- wxToolBarBase * GetToolBar () const
- bool IsStretchable () const
- bool IsButton () const
- bool IsControl () const
- bool IsSeparator () const
- bool IsStretchableSpace () const
- int GetStyle () const
- wxItemKind GetKind () const
- void MakeStretchable ()

- bool IsEnabled () const
- bool IsToggled () const
- bool CanBeToggled () const
- const wxBitmap & GetNormalBitmap () const
- const wxBitmap & GetDisabledBitmap () const
- const wxBitmap & GetBitmap () const
- const wxString & GetLabel () const
- const wxString & GetShortHelp () const
- const wxString & GetLongHelp () const
- wxObject * GetClientData () const
- virtual bool Enable (bool enable)
- virtual bool Toggle (bool toggle)
- virtual bool SetToggle (bool toggle)
- virtual bool SetShortHelp (const wxString &help)
- virtual bool SetLongHelp (const wxString &help)
- void Toggle ()
- virtual void SetNormalBitmap (const wxBitmap &bmp)
- virtual void SetDisabledBitmap (const wxBitmap &bmp)
- virtual void SetLabel (const wxString &label)
- void SetClientData (wxObject *clientData)
- virtual void Detach ()
- virtual void Attach (wxToolBarBase *tbar)
- virtual void SetDropdownMenu (wxMenu *menu)
- wxMenu * GetDropdownMenu () const

Additional Inherited Members

21.797.2 Constructor & Destructor Documentation

wxToolBarToolBase::wxToolBarToolBase (wxToolBarBase * *tbar* = NULL, int *toolid* = wxID_SEPARATOR, const wxString & *label* = wxEmptyString, const wxBitmap & *bmpNormal* = wxNullBitmap, const wxBitmap & *bmpDisabled* = wxNullBitmap, wxItemKind *kind* = wxITEM_NORMAL, wxObject * *clientData* = NULL, const wxString & *shortHelpString* = wxEmptyString, const wxString & *longHelpString* = wxEmptyString)

wxToolBarToolBase::wxToolBarToolBase (wxToolBarBase * *tbar*, wxControl * *control*, const wxString & *label*)

virtual wxToolBarToolBase::~wxToolBarToolBase ( ) [virtual]

21.797.3 Member Function Documentation

virtual void wxToolBarToolBase::Attach (wxToolBarBase * *tbar*) [virtual]

bool wxToolBarToolBase::CanBeToggled ( ) const

virtual void wxToolBarToolBase::Detach ( ) [virtual]

virtual bool wxToolBarToolBase::Enable (bool *enable*) [virtual]

const wxBitmap& wxToolBarToolBase::GetBitmap ( ) const

wxObject* wxToolBarToolBase::GetClientData ( ) const

wxControl* wxToolBarToolBase::GetControl ( ) const

const wxBitmap& wxToolBarToolBase::GetDisabledBitmap ( ) const

wxMenu* wxToolBarToolBase::GetDropdownMenu ( ) const

int wxToolBarToolBase::GetId ( ) const

wxItemKind wxToolBarToolBase::GetKind ( ) const

const wxString& wxToolBarToolBase::GetLabel ( ) const

const wxString& wxToolBarToolBase::GetLongHelp ( ) const

const wxBitmap& wxToolBarToolBase::GetNormalBitmap ( ) const

const wxString& wxToolBarToolBase::GetShortHelp ( ) const

int wxToolBarToolBase::GetStyle ( ) const

wxToolBarBase* wxToolBarToolBase::GetToolBar ( ) const

bool wxToolBarToolBase::IsButton ( ) const

bool wxToolBarToolBase::IsControl ( ) const

bool wxToolBarToolBase::IsEnabled ( ) const

bool wxToolBarToolBase::IsSeparator ( ) const

bool wxToolBarToolBase::IsStretchable ( ) const

bool wxToolBarToolBase::IsStretchableSpace ( ) const

bool wxToolBarToolBase::IsToggled ( ) const

void wxToolBarToolBase::MakeStretchable ( )

void wxToolBarToolBase::SetClientData (wxObject * *clientData*)

virtual void wxToolBarToolBase::SetDisabledBitmap (const wxBitmap & *bmp*) [virtual]

virtual void wxToolBarToolBase::SetDropdownMenu (wxMenu * *menu*) [virtual]

virtual void wxToolBarToolBase::SetLabel (const wxString & *label*) [virtual]

virtual bool wxToolBarToolBase::SetLongHelp (const wxString & *help*) [virtual]

virtual void wxToolBarToolBase::SetNormalBitmap (const wxBitmap & *bmp*) [virtual]

virtual bool wxToolBarToolBase::SetShortHelp (const wxString & *help*) [virtual]

virtual bool wxToolBarToolBase::SetToggle (bool *toggle*) [virtual]

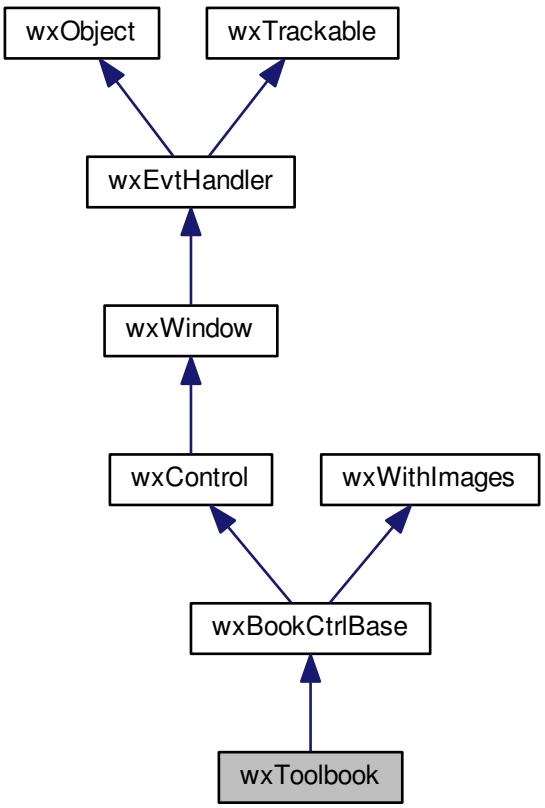
virtual bool wxToolBarToolBase::Toggle (bool *toggle*) [virtual]

void wxToolBarToolBase::Toggle ( )

21.798 wxToolbook Class Reference

`#include <wx/toolbook.h>`

Inheritance diagram for wxToolbook:



21.798.1 Detailed Description

wxToolbook is a class similar to wxNotebook but which uses a wxToolBar to show the labels instead of the tabs.

There is no documentation for this class yet but its usage is identical to wxNotebook (except for the features clearly related to tabs only), so please refer to that class documentation for now. You can also use the Notebook Sample to see wxToolbook in action.

Styles

This class supports the following styles:

- wxTBK_BUTTONBAR: Use wxButtonToolBar-based implementation under Mac OS (ignored under other platforms).
- wxTBK_HORZ_LAYOUT: Shows the text and the icons alongside, not vertically stacked (only implement under Windows and GTK 2 platforms as it relies on `wxTB_HORZ_LAYOUT` flag support).

The common wxBookCtrl styles described in the wxBookCtrl Overview are also supported.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxBookCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_TOOLBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_TOOLBOOK_PAGE_CHANGED` event.
- EVT_TOOLBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_TOOLBOOK_PAGE_CHANGING` event. This event can be vetoed (using wxNotifyEvent::Veto()).

Library: wxCore

Category: Book Controls

See also

wxBookCtrl Overview, wxBookCtrlBase, wxNotebook, Notebook Sample

Public Member Functions

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Create the tool book control that has already been constructed with the default constructor.
- wxToolBarBase * GetToolBar () const

  Returns the wxToolBarBase associated with the control.
- wxToolbook ()

  Constructs a choicebook control.
- wxToolbook (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxEmptyString)

  Constructs a choicebook control.

Additional Inherited Members

21.798.2 Constructor & Destructor Documentation

wxToolbook::wxToolbook ( )

Constructs a choicebook control.

wxToolbook::wxToolbook (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Constructs a choicebook control.

21.798.3 Member Function Documentation

bool wxToolbook::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxEmptyString)

Create the tool book control that has already been constructed with the default constructor.

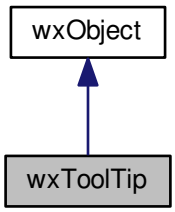
wxToolBarBase* wxToolbook::GetToolBar ( ) const

Returns the wxToolBarBase associated with the control.

21.799 wxToolTip Class Reference

`#include <wx/tooltip.h>`

Inheritance diagram for wxToolTip:



21.799.1 Detailed Description

This class holds information about a tooltip associated with a window (see wxWindow::SetToolTip()).

The four static methods, wxToolTip::Enable(), wxToolTip::SetDelay() wxToolTip::SetAutoPop() and wxToolTip::SetReshow() can be used to globally alter tooltips behaviour.

Library: wxCore

Category: Help

Public Member Functions

- wxToolTip (const wxString &tip)
  Constructor.
- wxString GetTip () const
  Get the tooltip text.
- wxWindow * GetWindow () const
  Get the associated window.
- void SetTip (const wxString &tip)

Set the tooltip text.

Static Public Member Functions

- static void Enable (bool flag)

  Enable or disable tooltips globally.
- static void SetAutoPop (long msecs)

  Set the delay after which the tooltip disappears or how long a tooltip remains visible.
- static void SetDelay (long msecs)

  Set the delay after which the tooltip appears.
- static void SetMaxWidth (int width)

  Set tooltip maximal width in pixels.
- static void SetReshow (long msecs)

  Set the delay between subsequent tooltips to appear.

Additional Inherited Members

21.799.2 Constructor & Destructor Documentation

wxToolTip::wxToolTip (const wxString & *tip*)

Constructor.

21.799.3 Member Function Documentation

static void wxToolTip::Enable (bool *flag*) `[static]`

Enable or disable tooltips globally.

Note

May not be supported on all platforms (eg. wxCocoa).

wxString wxToolTip::GetTip ( ) const

Get the tooltip text.

wxWindow* wxToolTip::GetWindow ( ) const

Get the associated window.

static void wxToolTip::SetAutoPop (long *msecs*) `[static]`

Set the delay after which the tooltip disappears or how long a tooltip remains visible.

Note

May not be supported on all platforms (eg. wxCocoa, GTK).

static void wxToolTip::SetDelay (long *msecs*) `[static]`

Set the delay after which the tooltip appears.

Note

May not be supported on all platforms (eg. wxCocoa).

static void wxToolTip::SetMaxWidth (int *width*) `[static]`

Set tooltip maximal width in pixels.

By default, tooltips are wrapped at a suitably chosen width. You can pass -1 as *width* to disable wrapping them completely, 0 to restore the default behaviour or an arbitrary positive value to wrap them at the given width.

Notice that this function does not change the width of the tooltips created before calling it.

Note

Currently this function is wxMSW-only.

static void wxToolTip::SetReshow (long *msecs*) `[static]`

Set the delay between subsequent tooltips to appear.

Note

May not be supported on all platforms (eg. wxCocoa, GTK).

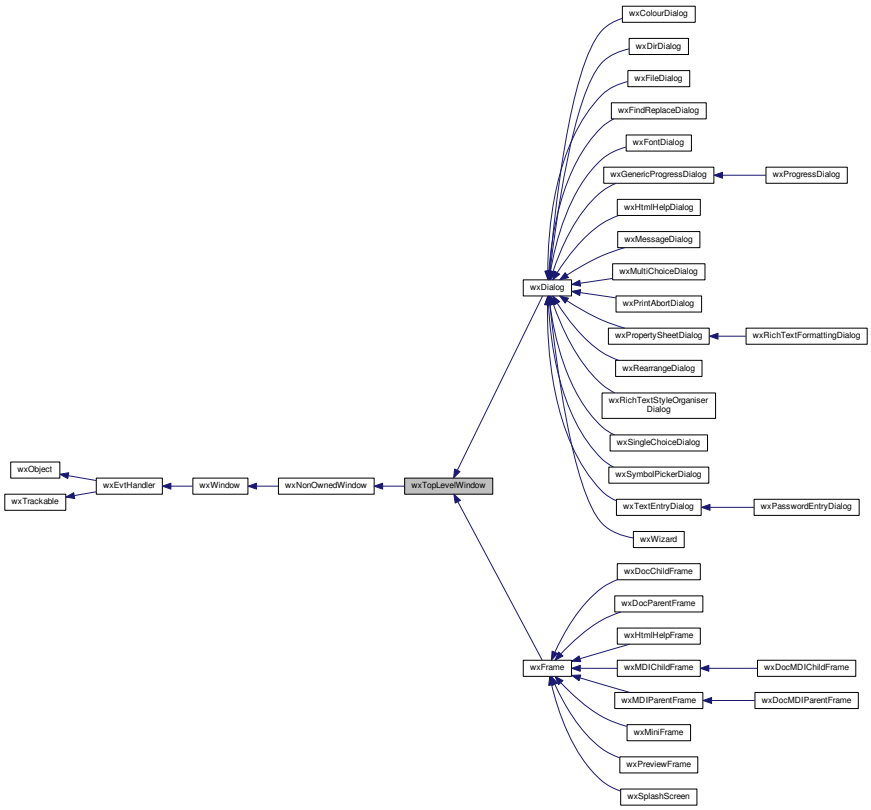
void wxToolTip::SetTip (const wxString & *tip*)

Set the tooltip text.

21.800 wxTopLevelWindow Class Reference

`#include <wx/toplevel.h>`

Inheritance diagram for wxTopLevelWindow:



21.800.1 Detailed Description

wxTopLevelWindow is a common base class for wxDialog and wxFrame.

It is an abstract base class meaning that you never work with objects of this class directly, but all of its methods are also applicable for the two classes above.

Note that the instances of wxTopLevelWindow are managed by wxWidgets in the internal top level window list.

Events emitted by this class

Event macros for events emitted by this class:

- EVT_MAXIMIZE(id, func): Process a `wxEVT_MAXIMIZE` event. See wxMaximizeEvent.
- EVT_MOVE(func): Process a `wxEVT_MOVE` event, which is generated when a window is moved. See wxMoveEvent.
- EVT_MOVE_START(func): Process a `wxEVT_MOVE_START` event, which is generated when the user starts to move or size a window. wxMSW only. See wxMoveEvent.
- EVT_MOVE_END(func): Process a `wxEVT_MOVE_END` event, which is generated when the user stops moving or sizing a window. wxMSW only. See wxMoveEvent.
- EVT_SHOW(func): Process a `wxEVT_SHOW` event. See wxShowEvent.

Library: wxCore

Category: Managed Windows

See also

wxDialog, wxFrame

Public Member Functions

- wxTopLevelWindow ()

  Default ctor.
- wxTopLevelWindow (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Constructor creating the top level window.
- virtual ~wxTopLevelWindow ()

  Destructor.
- bool Create (wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const wxString &name=wxFrameNameStr)

  Creates the top level window.
- virtual bool CanSetTransparent ()

  Returns true if the platform supports making the window translucent.
- void CenterOnScreen (int direction=wxBOTH)

  A synonym for CentreOnScreen().
- void CentreOnScreen (int direction=wxBOTH)

  Centres the window on screen.
- virtual bool EnableCloseButton (bool enable=true)

  Enables or disables the Close button (most often in the right upper corner of a dialog) and the Close entry of the system menu (most often in the left upper corner of the dialog).
- wxWindow * GetDefaultItem () const

  Returns a pointer to the button which is the default for this window, or NULL.
- wxIcon GetIcon () const

  Returns the standard icon of the window.
- const wxIconBundle & GetIcons () const

  Returns all icons associated with the window, there will be none of them if neither SetIcon() nor SetIcons() had been called before.
- virtual wxString GetTitle () const

  Gets a string containing the window title.
- virtual bool HandleSettingChange (WXWPARAM wParam, WXLPARAM lParam)

  Unique to the wxWinCE port.
- virtual void Iconize (bool iconize=true)

  Iconizes or restores the window.
- virtual bool IsActive ()

  Returns true if this window is currently active, i.e. if the user is currently working with it.
- virtual bool IsAlwaysMaximized () const

  Returns true if this window is expected to be always maximized, either due to platform policy or due to local policy regarding particular class.
- virtual bool IsFullScreen () const

  Returns true if the window is in fullscreen mode.

- virtual bool IsIconized () const

  Returns true if the window is iconized.
- virtual bool IsMaximized () const

  Returns true if the window is maximized.
- bool IsUsingNativeDecorations () const

  This method is specific to wxUniversal port.
- virtual bool Layout ()

  See wxWindow::SetAutoLayout(): when auto layout is on, this function gets called automatically when the window is resized.
- virtual void Maximize (bool maximize=true)

  Maximizes or restores the window.
- wxMenu * MSWGetSystemMenu () const

  MSW-specific function for accessing the system menu.
- virtual void RequestUserAttention (int flags=wxUSER_ATTENTION_INFO)

  Use a system-dependent way to attract users attention to the window when it is in background.
- void Restore ()

  Restore a previously iconized or maximized window to its normal state.
- wxWindow * SetDefaultItem (wxWindow *win)

  Changes the default item for the panel, usually win is a button.
- wxWindow * SetTmpDefaultItem (wxWindow *win)
- wxWindow * GetTmpDefaultItem () const
- void SetIcon (const wxIcon &icon)

  Sets the icon for this window.
- virtual void SetIcons (const wxIconBundle &icons)

  Sets several icons of different sizes for this window: this allows to use different icons for different situations (e.g.
- void SetLeftMenu (int id=wxID_ANY, const wxString &label=wxEmptyString, wxMenu *subMenu=NULL)

  Sets action or menu activated by pressing left hardware button on the smart phones.
- virtual void SetMaxSize (const wxSize &size)

  A simpler interface for setting the size hints than SetSizeHints().
- virtual void SetMinSize (const wxSize &size)

  A simpler interface for setting the size hints than SetSizeHints().
- void SetRightMenu (int id=wxID_ANY, const wxString &label=wxEmptyString, wxMenu *subMenu=NULL)

  Sets action or menu activated by pressing right hardware button on the smart phones.
- virtual void SetSizeHints (int minW, int minH, int maxW=-1, int maxH=-1, int incW=-1, int incH=-1)

  Allows specification of minimum and maximum window sizes, and window size increments.
- void SetSizeHints (const wxSize &minSize, const wxSize &maxSize=wxDefaultSize, const wxSize &incSize=wxDefaultSize)

  Allows specification of minimum and maximum window sizes, and window size increments.
- virtual void SetTitle (const wxString &title)

  Sets the window title.
- virtual bool SetTransparent (wxByte alpha)

  If the platform supports it will set the window to be translucent.
- virtual bool ShouldPreventAppExit () const

  This virtual function is not meant to be called directly but can be overridden to return false (it returns true by default) to allow the application to close even if this, presumably not very important, window is still opened.
- virtual void OSXSetModified (bool modified)

  This function sets the wxTopLevelWindow's modified state on OS X, which currently draws a black dot in the wxTopLevelWindow's close button.
- virtual bool OSXIsModified () const

  Returns the current modified state of the wxTopLevelWindow on OS X.
- virtual void SetRepresentedFilename (const wxString &filename)

Sets the file name represented by this wxTopLevelWindow.

- virtual void ShowWithoutActivating ()

  Show the wxTopLevelWindow, but do not give it keyboard focus.

- virtual bool EnableFullScreenView (bool enable=true)

  Adds or removes a full screen button to the right upper corner of a window's title bar under OS X 10.7 and later.

- virtual bool ShowFullScreen (bool show, long style=wxFULLSCREEN_ALL)

  Depending on the value of show parameter the window is either shown full screen or restored to its normal state.

- void UseNativeDecorations (bool native=true)

  This method is specific to wxUniversal port.

- void UseNativeDecorationsByDefault (bool native=true)

  This method is specific to wxUniversal port.

Static Public Member Functions

- static wxSize GetDefaultSize ()

  Get the default size for a new top level window.

Additional Inherited Members

21.800.2 Constructor & Destructor Documentation

wxTopLevelWindow::wxTopLevelWindow ( )

Default ctor.

wxTopLevelWindow::wxTopLevelWindow (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Constructor creating the top level window.

virtual wxTopLevelWindow::~wxTopLevelWindow ( ) `[virtual]`

Destructor.

Remember that wxTopLevelWindows do not get immediately destroyed when the user (or the app) closes them; they have a **delayed** destruction.

See Window Deletion for more info.

21.800.3 Member Function Documentation

virtual bool wxTopLevelWindow::CanSetTransparent ( ) `[virtual]`

Returns true if the platform supports making the window translucent.

See also

SetTransparent()

Reimplemented from wxWindow.

void wxTopLevelWindow::CenterOnScreen (int *direction* = wxBOTH)

A synonym for CentreOnScreen().

void wxTopLevelWindow::CentreOnScreen (int *direction* = wxBOTH)

Centres the window on screen.

Parameters

direction	Specifies the direction for the centering. May be `wxHORIZONTAL`, `wxVERTICAL` or `wxBOTH`.

See also

wxWindow::CentreOnParent()

bool wxTopLevelWindow::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *title*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxDEFAULT_FRAME_STYLE, const wxString & *name* = `wxFrameNameStr`)

Creates the top level window.

virtual bool wxTopLevelWindow::EnableCloseButton (bool *enable* = `true`) [virtual]

Enables or disables the Close button (most often in the right upper corner of a dialog) and the Close entry of the system menu (most often in the left upper corner of the dialog).

Currently only implemented for wxMSW and wxGTK.

Returns true if operation was successful. This may be wrong on X11 (including GTK+) where the window manager may not support this operation and there is no way to find out.

virtual bool wxTopLevelWindow::EnableFullScreenView (bool *enable* = `true`) [virtual]

Adds or removes a full screen button to the right upper corner of a window's title bar under OS X 10.7 and later.

Currently only available for wxOSX/Cocoa.

Parameters

enable	If true (default) adds the full screen button in the title bar; if false the button is removed.

Returns

true if the button was added or removed, false if running under a pre-OS X 10.7 system or another OS.

Note

Having the button is also required to let ShowFullScreen() make use of the full screen API available since OS X 10.7: a full screen window gets its own space and entering and exiting the mode is animated. If the button is not present the old way of switching to full screen is used.

Availability: only available for the wxOSX port.

See also

ShowFullScreen()

Since

3.1.0

wxWindow* wxTopLevelWindow::GetDefaultItem ( ) const

Returns a pointer to the button which is the default for this window, or NULL.

The default button is the one activated by pressing the Enter key.

static wxSize wxTopLevelWindow::GetDefaultSize ( ) [static]

Get the default size for a new top level window.

This is used internally by wxWidgets on some platforms to determine the default size for a window created using wxDefaultSize so it is not necessary to use it when creating a wxTopLevelWindow, however it may be useful if a rough estimation of the window size is needed for some other reason.

Since

2.9.2

wxIcon wxTopLevelWindow::GetIcon ( ) const

Returns the standard icon of the window.

The icon will be invalid if it hadn't been previously set by SetIcon().

See also

GetIcons()

const wxIconBundle& wxTopLevelWindow::GetIcons ( ) const

Returns all icons associated with the window, there will be none of them if neither SetIcon() nor SetIcons() had been called before.

Use GetIcon() to get the main icon of the window.

See also

wxIconBundle

virtual wxString wxTopLevelWindow::GetTitle ( ) const [virtual]

Gets a string containing the window title.

See also

SetTitle()

wxWindow* wxTopLevelWindow::GetTmpDefaultItem ( ) const

virtual bool wxTopLevelWindow::HandleSettingChange (WXWPARAM *wParam*, WXLPARAM *lParam*) [virtual]

Unique to the wxWinCE port.

Responds to showing/hiding SIP (soft input panel) area and resize window accordingly. Override this if you want to avoid resizing or do additional operations.

virtual void wxTopLevelWindow::Iconize (bool *iconize* = true) [virtual]

Iconizes or restores the window.

Parameters

iconize	If true, iconizes the window; if false, shows and restores it.

See also

IsIconized(), Restore()(), wxIconizeEvent.

Reimplemented in wxDialog.

virtual bool wxTopLevelWindow::IsActive ( ) [virtual]

Returns true if this window is currently active, i.e. if the user is currently working with it.

virtual bool wxTopLevelWindow::IsAlwaysMaximized ( ) const [virtual]

Returns true if this window is expected to be always maximized, either due to platform policy or due to local policy regarding particular class.

Reimplemented in wxMDIChildFrame.

virtual bool wxTopLevelWindow::IsFullScreen ( ) const [virtual]

Returns true if the window is in fullscreen mode.

See also

ShowFullScreen()

virtual bool wxTopLevelWindow::IsIconized ( ) const [virtual]

Returns true if the window is iconized.

Reimplemented in wxDialog.

virtual bool wxTopLevelWindow::IsMaximized ( ) const [virtual]

Returns true if the window is maximized.

bool wxTopLevelWindow::IsUsingNativeDecorations ( ) const

This method is specific to wxUniversal port.

Returns true if this window is using native decorations, false if we draw them ourselves.

See also

UseNativeDecorations(), UseNativeDecorationsByDefault()

virtual bool wxTopLevelWindow::Layout ( ) [virtual]

See wxWindow::SetAutoLayout(): when auto layout is on, this function gets called automatically when the window is resized.

Reimplemented from wxWindow.

virtual void wxTopLevelWindow::Maximize (bool *maximize* = true) [virtual]

Maximizes or restores the window.

Parameters

maximize	If true, maximizes the window, otherwise it restores it.

See also

Restore(), Iconize()

Reimplemented in wxMDIChildFrame.

wxMenu* wxTopLevelWindow::MSWGetSystemMenu ( ) const

MSW-specific function for accessing the system menu.

Returns a wxMenu pointer representing the system menu of the window under MSW. The returned wxMenu may be used, if non-NULL, to add extra items to the system menu. The usual wxEVT_MENU events (that can be processed using EVT_MENU event table macro) will then be generated for them. All the other wxMenu methods may be used as well but notice that they won't allow you to access any standard system menu items (e.g. they can't be deleted or modified in any way currently).

Notice that because of the native system limitations the identifiers of the items added to the system menu must be multiples of 16, otherwise no events will be generated for them.

The returned pointer must *not* be deleted, it is owned by the window and will be only deleted when the window itself is destroyed.

This function is not available in the other ports by design, any occurrences of it in the portable code must be guarded by

```
#ifdef __WXMSW__
```

preprocessor guards.

Since

2.9.3

virtual bool wxTopLevelWindow::OSXIsModified ( ) const `[virtual]`

Returns the current modified state of the wxTopLevelWindow on OS X.

On other platforms, this method does nothing.

See also

OSXSetModified()

virtual void wxTopLevelWindow::OSXSetModified (bool *modified*) `[virtual]`

This function sets the wxTopLevelWindow's modified state on OS X, which currently draws a black dot in the wxTopLevelWindow's close button.

On other platforms, this method does nothing.

See also

OSXIsModified()

virtual void wxTopLevelWindow::RequestUserAttention (int *flags* = wxUSER_ATTENTION_INFO) `[virtual]`

Use a system-dependent way to attract users attention to the window when it is in background.

flags may have the value of either `wxUSER_ATTENTION_INFO` (default) or `wxUSER_ATTENTION_ERROR` which results in a more drastic action. When in doubt, use the default value.

Note

This function should normally be only used when the application is not already in foreground.

This function is currently implemented for Win32 where it flashes the window icon in the taskbar, and for wxGTK with task bars supporting it.

void wxTopLevelWindow::Restore ( )

Restore a previously iconized or maximized window to its normal state.

In wxGTK this method currently doesn't return the maximized window to its normal state and you must use Maximize() with false argument explicitly for this. In the other ports, it both unmaximizes the maximized windows and uniconizes the iconized ones.

See also

Iconize(), Maximize()

wxWindow* wxTopLevelWindow::SetDefaultItem (wxWindow * *win*)

Changes the default item for the panel, usually *win* is a button.

See also

GetDefaultItem()

void wxTopLevelWindow::SetIcon (const wxIcon & *icon*)

Sets the icon for this window.

Parameters

icon	The wxIcon to associate with this window.

Remarks

The window takes a 'copy' of *icon*, but since it uses reference counting, the copy is very quick. It is safe to delete *icon* after calling this function.

Note

In wxMSW, *icon* must be either 16x16 or 32x32 icon.

See also

wxIcon, SetIcons()

virtual void wxTopLevelWindow::SetIcons (const wxIconBundle & *icons*) [virtual]

Sets several icons of different sizes for this window: this allows to use different icons for different situations (e.g.

task switching bar, taskbar, window title bar) instead of scaling, with possibly bad looking results, the only icon set by SetIcon().

Parameters

icons	The icons to associate with this window.

Note

In wxMSW, *icons* must contain a 16x16 or 32x32 icon, preferably both.

See also

wxIconBundle

Reimplemented in wxDialog.

void wxTopLevelWindow::SetLeftMenu (int *id* = wxID_ANY, const wxString & *label* = wxEmptyString, wxMenu * *subMenu* = NULL)

Sets action or menu activated by pressing left hardware button on the smart phones.

Unavailable on full keyboard machines.

Parameters

id	Identifier for this button.
label	Text to be displayed on the screen area dedicated to this hardware button.
subMenu	The menu to be opened after pressing this hardware button.

See also

SetRightMenu().

virtual void wxTopLevelWindow::SetMaxSize (const wxSize & *size*) [virtual]

A simpler interface for setting the size hints than SetSizeHints().

Reimplemented from wxWindow.

virtual void wxTopLevelWindow::SetMinSize (const wxSize & *size*) [virtual]

A simpler interface for setting the size hints than SetSizeHints().

Reimplemented from wxWindow.

virtual void wxTopLevelWindow::SetRepresentedFilename (const wxString & *filename*) [virtual]

Sets the file name represented by this wxTopLevelWindow.

Under OS X, this file name is used to set the "proxy icon", which appears in the window title bar near its title, corresponding to this file name. Under other platforms it currently doesn't do anything but it is harmless to call it now and it might be implemented to do something useful in the future so you're encouraged to use it for any window representing a file-based document.

Since

2.9.4

void wxTopLevelWindow::SetRightMenu (int *id* = wxID_ANY, const wxString & *label* = wxEmptyString, wxMenu * *subMenu* = NULL)

Sets action or menu activated by pressing right hardware button on the smart phones.

Unavailable on full keyboard machines.

Parameters

id	Identifier for this button.
label	Text to be displayed on the screen area dedicated to this hardware button.
subMenu	The menu to be opened after pressing this hardware button.

See also

SetLeftMenu().

virtual void wxTopLevelWindow::SetSizeHints (int *minW*, int *minH*, int *maxW* = -1, int *maxH* = -1, int *incW* = -1, int *incH* = -1) [virtual]

Allows specification of minimum and maximum window sizes, and window size increments.

If a pair of values is not set (or set to -1), no constraints will be used.

Parameters

minW	The minimum width.
minH	The minimum height.
maxW	The maximum width.
maxH	The maximum height.
incW	Specifies the increment for sizing the width (GTK/Motif/Xt only).
incH	Specifies the increment for sizing the height (GTK/Motif/Xt only).

Remarks

Notice that this function not only prevents the user from resizing the window outside the given bounds but it also prevents the program itself from doing it using wxWindow::SetSize().

Reimplemented from wxWindow.

void wxTopLevelWindow::SetSizeHints (const wxSize & *minSize*, const wxSize & *maxSize* = wxDefaultSize, const wxSize & *incSize* = wxDefaultSize) [virtual]

Allows specification of minimum and maximum window sizes, and window size increments.

If a pair of values is not set (or set to -1), no constraints will be used.

Parameters

minSize	The minimum size of the window.
maxSize	The maximum size of the window.
incSize	Increment size (only taken into account under X11-based ports such as wxGTK/wxMotif/wx↩ X11).

Remarks

Notice that this function not only prevents the user from resizing the window outside the given bounds but it also prevents the program itself from doing it using wxWindow::SetSize().

Reimplemented from wxWindow.

virtual void wxTopLevelWindow::SetTitle (const wxString & *title*) [virtual]

Sets the window title.

Parameters

title	The window title.

See also

GetTitle()

wxWindow* wxTopLevelWindow::SetTmpDefaultItem (wxWindow * *win*)

virtual bool wxTopLevelWindow::SetTransparent (wxByte *alpha*) [virtual]

If the platform supports it will set the window to be translucent.

Parameters

alpha	Determines how opaque or transparent the window will be, if the platform supports the operation. A value of 0 sets the window to be fully transparent, and a value of 255 sets the window to be fully opaque.

Reimplemented from wxWindow.

virtual bool wxTopLevelWindow::ShouldPreventAppExit ( ) const [virtual]

This virtual function is not meant to be called directly but can be overridden to return false (it returns true by default) to allow the application to close even if this, presumably not very important, window is still opened.

By default, the application stays alive as long as there are any open top level windows.

virtual bool wxTopLevelWindow::ShowFullScreen (bool *show*, long *style* = wxFULLSCREEN_ALL) [virtual]

Depending on the value of *show* parameter the window is either shown full screen or restored to its normal state.

style is a bit list containing some or all of the following values, which indicate what elements of the window to hide in full-screen mode:

- wxFULLSCREEN_NOMENUBAR
- wxFULLSCREEN_NOTOOLBAR
- wxFULLSCREEN_NOSTATUSBAR
- wxFULLSCREEN_NOBORDER
- wxFULLSCREEN_NOCAPTION
- wxFULLSCREEN_ALL (all of the above)

This function has not been tested with MDI frames.

Note

Showing a window full screen also actually Show()s the window if it isn't shown.

See also

EnableFullScreenView(), IsFullScreen()

virtual void wxTopLevelWindow::ShowWithoutActivating ( ) [virtual]

Show the wxTopLevelWindow, but do not give it keyboard focus.

This can be used for pop up or notification windows that should not steal the current focus.

void wxTopLevelWindow::UseNativeDecorations (bool *native* = true)

This method is specific to wxUniversal port.

Use native or custom-drawn decorations for this window only. Notice that to have any effect this method must be called before really creating the window, i.e. two step creation must be used:

```
MyFrame *frame = new MyFrame;       // use default ctor
frame->UseNativeDecorations(false); // change from default "true"
frame->Create(parent, title, ...);  // really create the frame
```

See also

UseNativeDecorationsByDefault(), IsUsingNativeDecorations()

void wxTopLevelWindow::UseNativeDecorationsByDefault (bool *native* = true)

This method is specific to wxUniversal port.

Top level windows in wxUniversal port can use either system-provided window decorations (i.e. title bar and various icons, buttons and menus in it) or draw the decorations themselves. By default the system decorations are used if they are available, but this method can be called with *native* set to false to change this for all windows created after this point.

Also note that if WXDECOR environment variable is set, then custom decorations are used by default and so it may make sense to call this method with default argument if the application can't use custom decorations at all for some reason.

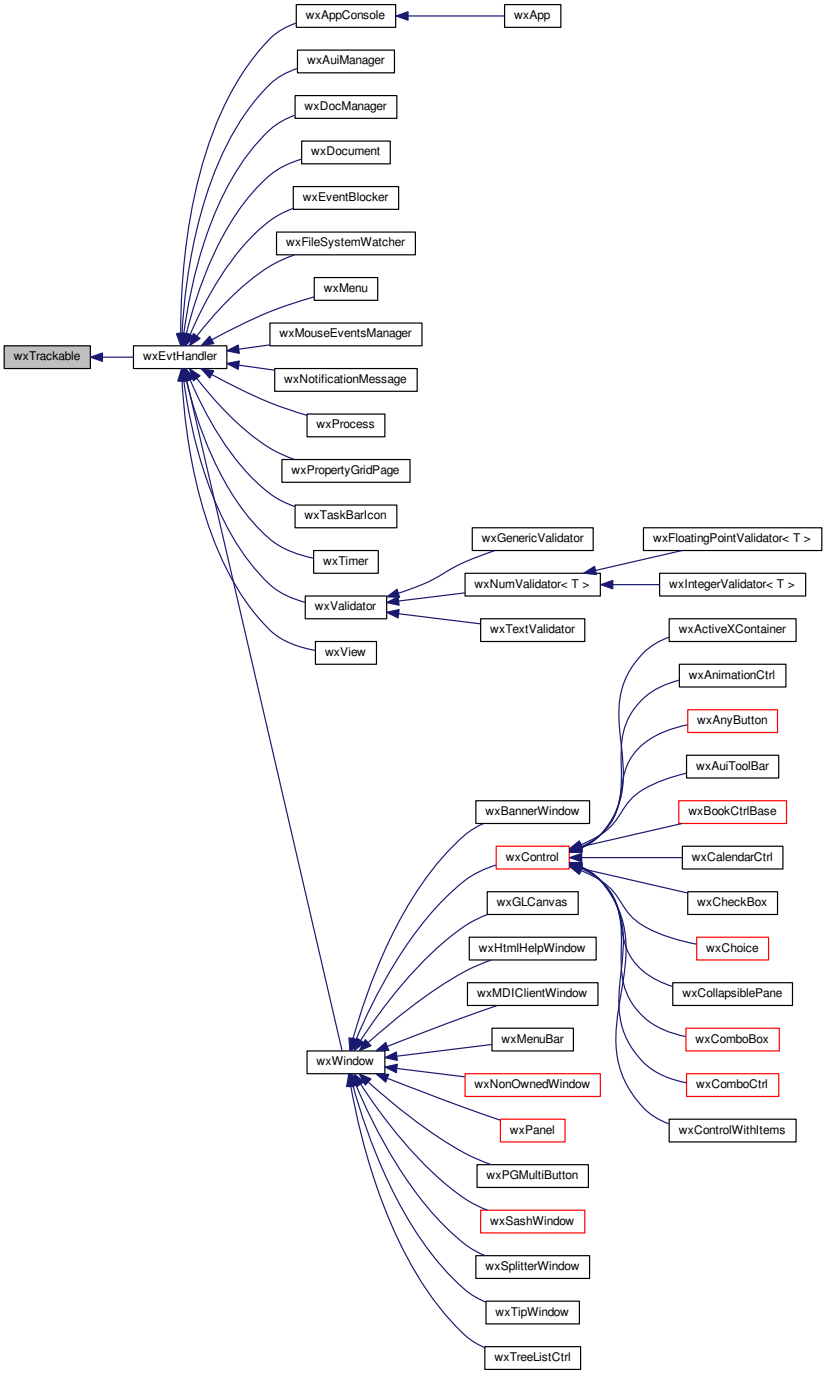
See also

UseNativeDecorations()

21.801 wxTrackable Class Reference

`#include <wx/tracker.h>`

Inheritance diagram for wxTrackable:



21.801.1 Detailed Description

Add-on base class for a trackable object.

This class maintains an internal linked list of classes of type wxTrackerNode and calls OnObjectDestroy() on them if this object is destroyed. The most common usage is by using the wxWeakRef<T> class template which automates this. This class has no public API. Its only use is by deriving another class from it to make it trackable.

```
class MyClass: public Foo, public wxTrackable
{
    // whatever
}

typedef wxWeakRef<MyClass> MyClassRef;
```

Library: wxBase

Category: Smart Pointers

21.802 wxTransform2D Class Reference

`#include <wx/geometry.h>`

Public Member Functions

- virtual ~wxTransform2D ()
- virtual void Transform (wxPoint2DInt *pt) const =0
- virtual void Transform (wxRect2DInt *r) const
- virtual wxPoint2DInt Transform (const wxPoint2DInt &pt) const
- virtual wxRect2DInt Transform (const wxRect2DInt &r) const
- virtual void InverseTransform (wxPoint2DInt *pt) const =0
- virtual void InverseTransform (wxRect2DInt *r) const
- virtual wxPoint2DInt InverseTransform (const wxPoint2DInt &pt) const
- virtual wxRect2DInt InverseTransform (const wxRect2DInt &r) const

21.802.1 Constructor & Destructor Documentation

virtual wxTransform2D::~wxTransform2D ( ) [virtual]

21.802.2 Member Function Documentation

virtual void wxTransform2D::InverseTransform (wxPoint2DInt * *pt*) const [pure virtual]

virtual void wxTransform2D::InverseTransform (wxRect2DInt * *r*) const [virtual]

virtual wxPoint2DInt wxTransform2D::InverseTransform (const wxPoint2DInt & *pt*) const [virtual]

virtual wxRect2DInt wxTransform2D::InverseTransform (const wxRect2DInt & *r*) const [virtual]

virtual void wxTransform2D::Transform (wxPoint2DInt * *pt*) const [pure virtual]

virtual void wxTransform2D::Transform (wxRect2DInt * *r*) const [virtual]

virtual wxPoint2DInt wxTransform2D::Transform (const wxPoint2DInt & *pt*) const [virtual]

virtual wxRect2DInt wxTransform2D::Transform (const wxRect2DInt & *r*) const [virtual]

21.803 wxTranslations Class Reference

```
#include <wx/translation.h>
```

21.803.1 Detailed Description

This class allows to get translations for strings.

In wxWidgets this class manages message catalogs which contain the translations of the strings used to the current language. Unlike wxLocale, it isn't bound to locale. It can be used either independently of, or in conjunction with wxLocale. In the latter case, you should initialize wxLocale (which creates wxTranslations instance) first; in the former, you need to create a wxTranslations object and Set() it manually.

Only one wxTranslations instance is active at a time; it is set with the Set() method and obtained using Get().

Unlike wxLocale, wxTranslations' primary mean of identifying language is by its "canonical name", i.e. ISO 639 code, possibly combined with ISO 3166 country code and additional modifiers (examples include "fr", "en_GB" or "ca@valencia"; see wxLocale::GetCanonicalName() for more information). This allows apps using wxTranslations API to use even languages not recognized by the operating system or not listed in wxLanguage enum.

Since

2.9.1

See also

wxLocale, wxTranslationsLoader, wxFileTranslationsLoader

Public Member Functions

- wxTranslations ()

  Constructor.
- void SetLoader (wxTranslationsLoader *loader)

  Changes loader use to read catalogs to a non-default one.
- void SetLanguage (wxLanguage lang)

  Sets translations language to use.
- void SetLanguage (const wxString &lang)

  Sets translations language to use.
- wxArrayString GetAvailableTranslations (const wxString &domain) const

  Returns list of all translations of domain that were found.
- wxString GetBestTranslation (const wxString &domain, wxLanguage msgIdLanguage)

  Returns the best UI language for the domain.
- wxString GetBestTranslation (const wxString &domain, const wxString &msgIdLanguage="en")

  Returns the best UI language for the domain.
- bool AddStdCatalog ()

  Add standard wxWidgets catalogs ("wxstd" and possible port-specific catalogs).
- bool AddCatalog (const wxString &domain)

  Add a catalog for use with the current locale.
- bool AddCatalog (const wxString &domain, wxLanguage msgIdLanguage)

  Same as AddCatalog(const wxString&), but takes an additional argument, msgIdLanguage.
- bool AddCatalog (const wxString &domain, wxLanguage msgIdLanguage, const wxString &msgIdCharset)

  Same as AddCatalog(const wxString&, wxLanguage), but takes two additional arguments, msgIdLanguage and msgIdCharset.
- bool IsLoaded (const wxString &domain) const

Check if the given catalog is loaded, and returns true if it is.

- const wxString * GetTranslatedString (const wxString &origString, const wxString &domain=wxEmptyString) const

  Retrieves the translation for a string in all loaded domains unless the domain parameter is specified (and then only this catalog/domain is searched).

- const wxString * GetTranslatedString (const wxString &origString, unsigned n, const wxString &domain=wxEmptyString) const

  Retrieves the translation for a string in all loaded domains unless the domain parameter is specified (and then only this catalog/domain is searched).

- wxString GetHeaderValue (const wxString &header, const wxString &domain=wxEmptyString) const

  Returns the header value for header header.

Static Public Member Functions

- static wxTranslations * Get ()

  Returns current translations object, may return NULL.

- static void Set (wxTranslations *t)

  Sets current translations object.

21.803.2 Constructor & Destructor Documentation

wxTranslations::wxTranslations ( )

Constructor.

21.803.3 Member Function Documentation

bool wxTranslations::AddCatalog (const wxString & *domain*)

Add a catalog for use with the current locale.

By default, it is searched for in standard places (see wxFileTranslationsLoader), but you may also prepend additional directories to the search path with wxFileTranslationsLoader::AddCatalogLookupPathPrefix().

All loaded catalogs will be used for message lookup by GetString() for the current locale.

In this overload, `msgid` strings are assumed to be in English and written only using 7-bit ASCII characters. If you have to deal with non-English strings or 8-bit characters in the source code, see the instructions in Writing Non-English Applications.

Returns

true if catalog was successfully loaded, false otherwise (which might mean that the catalog is not found or that it isn't in the correct format).

bool wxTranslations::AddCatalog (const wxString & *domain*, wxLanguage *msgIdLanguage*)

Same as AddCatalog(const wxString&), but takes an additional argument, *msgIdLanguage*.

Parameters

domain	The catalog domain to add.
msgIdLanguage	Specifies the language of "msgid" strings in source code (i.e. arguments to GetString(), wxGetTranslation() and the _() macro). It is used if AddCatalog() cannot find any catalog for current language: if the language is same as source code language, then strings from source code are used instead.

Returns

true if catalog was successfully loaded, false otherwise (which might mean that the catalog is not found or that it isn't in the correct format).

bool wxTranslations::AddCatalog (const wxString & *domain*, wxLanguage *msgIdLanguage*, const wxString & *msgIdCharset*)

Same as AddCatalog(const wxString&, wxLanguage), but takes two additional arguments, *msgIdLanguage* and *msgIdCharset*.

This overload is only available in non-Unicode build.

Parameters

domain	The catalog domain to add.
msgIdLanguage	Specifies the language of "msgid" strings in source code (i.e. arguments to GetString(), wxGetTranslation() and the _() macro). It is used if AddCatalog() cannot find any catalog for current language: if the language is same as source code language, then strings from source code are used instead.
msgIdCharset	Lets you specify the charset used for msgids in sources in case they use 8-bit characters (e.g. German or French strings).

Returns

true if catalog was successfully loaded, false otherwise (which might mean that the catalog is not found or that it isn't in the correct format).

bool wxTranslations::AddStdCatalog ( )

Add standard wxWidgets catalogs ("wxstd" and possible port-specific catalogs).

Returns

true if a suitable catalog was found, false otherwise

See also

AddCatalog()

static wxTranslations* wxTranslations::Get ( ) [static]

Returns current translations object, may return NULL.

You must either call this early in app initialization code, or let wxLocale do it for you.

wxArrayString wxTranslations::GetAvailableTranslations (const wxString & *domain*) const

Returns list of all translations of *domain* that were found.

This method can be used e.g. to populate list of application's translations offered to the user. To do this, pass the app's main catalog as *domain*.

See also

GetBestTranslation()

wxString wxTranslations::GetBestTranslation (const wxString & *domain*, wxLanguage *msgIdLanguage*)

Returns the best UI language for the *domain*.

The language is determined from the preferred UI language or languages list the user configured in the OS. Notice that this may or may not correspond to the default *locale* as obtained from wxLocale::GetSystemLanguage(); modern operation systems (Windows Vista+, OS X) have separate language and regional (= locale) settings.

Parameters

domain	The catalog domain to look for.
msgIdLanguage	Specifies the language of "msgid" strings in source code (i.e. arguments to GetString(), wxGetTranslation() and the _() macro).

Returns

Language code if a suitable match was found, empty string otherwise.

Since

2.9.5

wxString wxTranslations::GetBestTranslation (const wxString & *domain*, const wxString & *msgIdLanguage* = "en")

Returns the best UI language for the *domain*.

The language is determined from the preferred UI language or languages list the user configured in the OS. Notice that this may or may not correspond to the default *locale* as obtained from wxLocale::GetSystemLanguage(); modern operation systems (Windows Vista+, OS X) have separate language and regional (= locale) settings.

Parameters

domain	The catalog domain to look for.
msgIdLanguage	Specifies the language of "msgid" strings in source code (i.e. arguments to GetString(), wxGetTranslation() and the _() macro).

Returns

Language code if a suitable match was found, empty string otherwise.

Since

2.9.5

wxString wxTranslations::GetHeaderValue (const wxString & *header,* const wxString & *domain* = wxEmptyString) const

Returns the header value for header *header*.

The search for *header* is case sensitive. If an *domain* is passed, this domain is searched. Else all domains will be searched until a header has been found.

The return value is the value of the header if found. Else this will be empty.

const wxString* wxTranslations::GetTranslatedString (const wxString & *origString,* const wxString & *domain* = wxEmptyString) const

Retrieves the translation for a string in all loaded domains unless the *domain* parameter is specified (and then only this catalog/domain is searched).

Returns NULL if translation is not available.

This function is thread-safe.

Remarks

Domains are searched in the last to first order, i.e. catalogs added later override those added before.

Since

3.0

const wxString* wxTranslations::GetTranslatedString (const wxString & *origString,* unsigned *n,* const wxString & *domain* = wxEmptyString) const

Retrieves the translation for a string in all loaded domains unless the *domain* parameter is specified (and then only this catalog/domain is searched).

Returns NULL if translation is not available.

This form is used when retrieving translation of string that has different singular and plural form in English or different plural forms in some other language.

Parameters

origString	The singular form of the string to be converted.
n	The number on which the plural form choice depends on. (In some languages, there are different plural forms for e.g. n=2 and n=3 etc., in addition to the singular form (n=1) being different.)
domain	The only domain (i.e. message catalog) to search if specified. By default this parameter is empty, indicating that all loaded catalogs should be searched.

See GNU gettext manual for additional information on plural forms handling. This method is called by the wxGet↩Translation() function and _() macro.

This function is thread-safe.

Remarks

Domains are searched in the last to first order, i.e. catalogs added later override those added before.

Since

3.0

bool wxTranslations::IsLoaded (const wxString & *domain*) const

Check if the given catalog is loaded, and returns true if it is.

According to GNU gettext tradition, each catalog normally corresponds to 'domain' which is more or less the application name.

See also

AddCatalog()

static void wxTranslations::Set (wxTranslations * *t*) [static]

Sets current translations object.

Deletes previous translation object and takes ownership of *t*.

void wxTranslations::SetLanguage (wxLanguage *lang*)

Sets translations language to use.

wxLANGUAGE_DEFAULT has special meaning: best suitable translation, given user's preference and available translations, will be used.

void wxTranslations::SetLanguage (const wxString & *lang*)

Sets translations language to use.

Empty *lang* string has the same meaning as wxLANGUAGE_DEFAULT in SetLanguage(wxLanguage): best suitable translation, given user's preference and available translations, will be used.

void wxTranslations::SetLoader (wxTranslationsLoader * *loader*)

Changes loader use to read catalogs to a non-default one.

Deletes previous loader and takes ownership of *loader*.

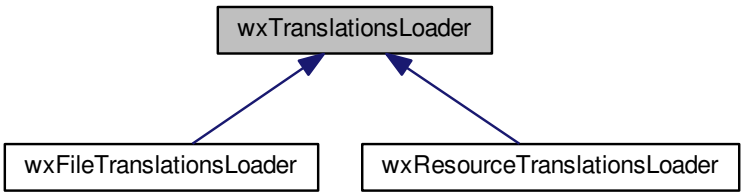
See also

wxTranslationsLoader, wxFileTranslationsLoader, wxResourceTranslationsLoader

21.804 wxTranslationsLoader Class Reference

`#include <wx/translation.h>`

Inheritance diagram for wxTranslationsLoader:



21.804.1 Detailed Description

Abstraction of translations discovery and loading.

This interface makes it possible to override wxWidgets' default catalogs loading mechanism and load MO files from locations other than the filesystem (e.g. embed them in executable).

Implementations must implement the LoadCatalog() method.

See also

wxFileTranslationsLoader, wxResourceTranslationsLoader

Since

2.9.1

Public Member Functions

- wxTranslationsLoader ()

  Trivial default constructor.
- virtual wxMsgCatalog * LoadCatalog (const wxString &domain, const wxString &lang)=0

  Called to load requested catalog.
- virtual wxArrayString GetAvailableTranslations (const wxString &domain) const =0

  Implements wxTranslations::GetAvailableTranslations().

21.804.2 Constructor & Destructor Documentation

wxTranslationsLoader::wxTranslationsLoader ( )

Trivial default constructor.

21.804.3 Member Function Documentation

virtual wxArrayString wxTranslationsLoader::GetAvailableTranslations (const wxString & *domain*) const `[pure virtual]`

Implements wxTranslations::GetAvailableTranslations().

virtual wxMsgCatalog* wxTranslationsLoader::LoadCatalog (const wxString & *domain,* const wxString & *lang*) `[pure virtual]`

Called to load requested catalog.

If the catalog is found, LoadCatalog() should create wxMsgCatalog instance with its data and return it. The caller will take ownership of the catalog.

Parameters

domain	Domain to load.
lang	Language to look for. This is "canonical name" (see wxLocale::GetCanonicalName()), i.e. ISO 639 code, possibly combined with country code or additional modifiers (e.g. "fr", "en_GB" or "ca@valencia").

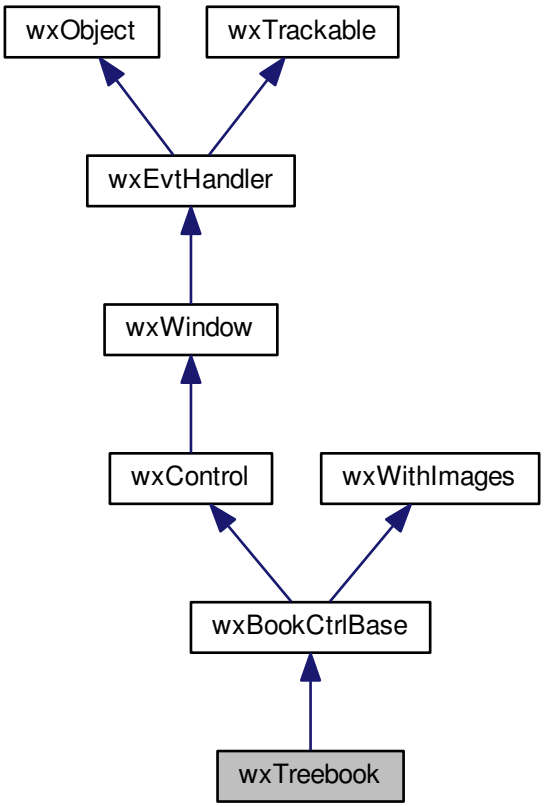
Returns

Loaded catalog or NULL on failure.

21.805 wxTreebook Class Reference

`#include <wx/treebook.h>`

Inheritance diagram for wxTreebook:



21.805.1 Detailed Description

This class is an extension of the wxNotebook class that allows a tree structured set of pages to be shown in a control.

A classic example is a netscape preferences dialog that shows a tree of preference sections on the left and select section page on the right.

To use the class simply create it and populate with pages using InsertPage(), InsertSubPage(), AddPage(), AddSubPage().

If your tree is no more than 1 level in depth then you could simply use AddPage() and AddSubPage() to sequentially populate your tree by adding at every step a page or a subpage to the end of the tree.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxBookCtrlEvent& event)

Event macros for events emitted by this class:

- EVT_TREEBOOK_PAGE_CHANGED(id, func): The page selection was changed. Processes a `wxEVT_TREEBOOK_PAGE_CHANGED` event.

- EVT_TREEBOOK_PAGE_CHANGING(id, func): The page selection is about to be changed. Processes a `wxEVT_TREEBOOK_PAGE_CHANGING` event. This event can be vetoed.
- EVT_TREEBOOK_NODE_COLLAPSED(id, func): The page node is going to be collapsed. Processes a `wxEVT_TREEBOOK_NODE_COLLAPSED` event.
- EVT_TREEBOOK_NODE_EXPANDED(id, func): The page node is going to be expanded. Processes a `wxEVT_TREEBOOK_NODE_EXPANDED` event.

Library: wxCore

Category: Book Controls

See also

wxBookCtrl, wxBookCtrlEvent, wxNotebook, wxTreeCtrl, wxImageList, wxBookCtrl Overview, Notebook Sample

Public Member Functions

- wxTreebook ()

  Default constructor.
- wxTreebook (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxBK_DEFAULT, const wxString &name=wxEmptyString)

  Creates an empty wxTreebook.
- virtual ~wxTreebook ()

  Destroys the wxTreebook object.
- virtual bool AddPage (wxWindow *page, const wxString &text, bool bSelect=false, int imageId=wxNOT_FOUND)

  Adds a new page.
- virtual bool AddSubPage (wxWindow *page, const wxString &text, bool bSelect=false, int imageId=wxNOT_FOUND)

  Adds a new child-page to the last top-level page.
- bool CollapseNode (size_t pageId)

  Shortcut for ExpandNode(pageId, false).
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxBK_DEFAULT, const wxString &name=wxEmptyString)

  Creates a treebook control.
- virtual bool DeletePage (size_t pagePos)

  Deletes the page at the specified position and all its children.
- virtual bool ExpandNode (size_t pageId, bool expand=true)

  Expands (collapses) the pageId node.
- int GetPageParent (size_t page) const

  Returns the parent page of the given one or `wxNOT_FOUND` if this is a top-level page.
- virtual int GetSelection () const

  Returns the currently selected page, or `wxNOT_FOUND` if none was selected.
- virtual bool InsertPage (size_t pagePos, wxWindow *page, const wxString &text, bool bSelect=false, int imageId=wxNOT_FOUND)

  Inserts a new page just before the page indicated by pagePos.
- virtual bool InsertSubPage (size_t pagePos, wxWindow *page, const wxString &text, bool bSelect=false, int imageId=wxNOT_FOUND)

  Inserts a sub page under the specified page.
- virtual bool IsNodeExpanded (size_t pageId) const

  Returns true if the page represented by pageId is expanded.

Additional Inherited Members

21.805.2 Constructor & Destructor Documentation

wxTreebook::wxTreebook ( )

Default constructor.

wxTreebook::wxTreebook (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxBK_DEFAULT, const wxString & *name* = wxEmptyString)

Creates an empty wxTreebook.

Parameters

parent	The parent window. Must be non-NULL.
id	The window identifier.
pos	The window position.
size	The window size.
style	The window style. See wxNotebook.
name	The name of the control (used only under Motif).

virtual wxTreebook::~wxTreebook ( ) [virtual]

Destroys the wxTreebook object.

Also deletes all the pages owned by the control (inserted previously into it).

21.805.3 Member Function Documentation

virtual bool wxTreebook::AddPage (wxWindow * *page*, const wxString & *text*, bool *bSelect* = false, int *imageId* = wxNOT_FOUND) [virtual]

Adds a new page.

The page is placed at the topmost level after all other pages. NULL could be specified for page to create an empty page.

Reimplemented from wxBookCtrlBase.

virtual bool wxTreebook::AddSubPage (wxWindow * *page*, const wxString & *text*, bool *bSelect* = false, int *imageId* = wxNOT_FOUND) [virtual]

Adds a new child-page to the last top-level page.

NULL could be specified for page to create an empty page.

bool wxTreebook::CollapseNode (size_t *pageId*)

Shortcut for ExpandNode(*pageId*, false).

bool wxTreebook::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxBK_DEFAULT, const wxString & *name* = wxEmptyString)

Creates a treebook control.

See wxTreebook::wxTreebook() for the description of the parameters.

virtual bool wxTreebook::DeletePage (size_t *pagePos*) [virtual]

Deletes the page at the specified position and all its children.

Could trigger page selection change in a case when selected page is removed. In that case its parent is selected (or the next page if no parent).

Reimplemented from wxBookCtrlBase.

virtual bool wxTreebook::ExpandNode (size_t *pageId*, bool *expand* = true) [virtual]

Expands (collapses) the *pageId* node.

Returns the previous state. May generate page changing events (if selected page is under the collapsed branch, then its parent is autoselected).

int wxTreebook::GetPageParent (size_t *page*) const

Returns the parent page of the given one or wxNOT_FOUND if this is a top-level page.

virtual int wxTreebook::GetSelection ( ) const [virtual]

Returns the currently selected page, or wxNOT_FOUND if none was selected.

Note

This method may return either the previously or newly selected page when called from the EVT_TREEBOOK_PAGE_CHANGED() handler depending on the platform and so wxBookCtrlEvent::GetSelection() should be used instead in this case.

Implements wxBookCtrlBase.

virtual bool wxTreebook::InsertPage (size_t *pagePos*, wxWindow * *page*, const wxString & *text*, bool *bSelect* = false, int *imageId* = wxNOT_FOUND) [virtual]

Inserts a new page just before the page indicated by *pagePos*.

The new page is placed before *pagePos* page and on the same level. NULL could be specified for page to create an empty page.

Implements wxBookCtrlBase.

virtual bool wxTreebook::InsertSubPage (size_t *pagePos*, wxWindow * *page*, const wxString & *text*, bool *bSelect* = false, int *imageId* = wxNOT_FOUND) [virtual]

Inserts a sub page under the specified page.

NULL could be specified for page to create an empty page.

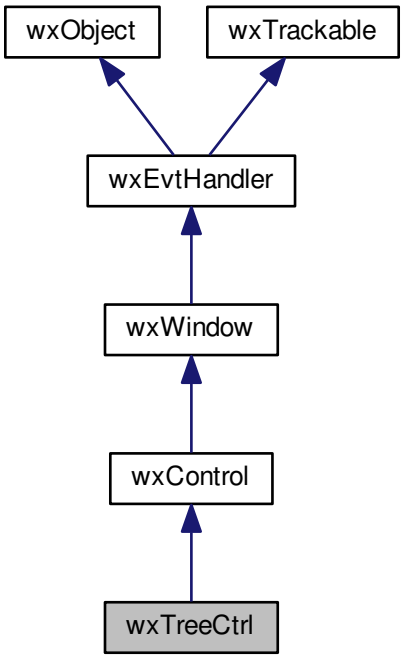
virtual bool wxTreebook::IsNodeExpanded (size_t *pageId*) const [virtual]

Returns true if the page represented by *pageId* is expanded.

21.806 wxTreeCtrl Class Reference

`#include <wx/treectrl.h>`

Inheritance diagram for wxTreeCtrl:



21.806.1 Detailed Description

A tree control presents information as a hierarchy, with items that may be expanded to show further items.

Items in a tree control are referenced by wxTreeItemId handles, which may be tested for validity by calling wxTreeItemId::IsOk().

A similar control with a fully native implementation for GTK+ and OS X as well is wxDataViewTreeCtrl.

To intercept events from a tree control, use the event table macros described in wxTreeEvent.

Styles

This class supports the following styles:

- wxTR_EDIT_LABELS: Use this style if you wish the user to be able to edit labels in the tree control.
- wxTR_NO_BUTTONS: For convenience to document that no buttons are to be drawn.
- wxTR_HAS_BUTTONS: Use this style to show + and - buttons to the left of parent items.
- wxTR_TWIST_BUTTONS: Selects alternative style of `+/- buttons` and shows rotating ("twisting") arrows instead. Currently this style is only implemented under Microsoft Windows Vista and later Windows versions and is ignored under the other platforms. Notice that under Vista this style results in the same appearance

as used by the tree control in Explorer and other built-in programs and so using it may be preferable to the default style.

- wxTR_NO_LINES: Use this style to hide vertical level connectors.
- wxTR_FULL_ROW_HIGHLIGHT: Use this style to have the background colour and the selection highlight extend over the entire horizontal row of the tree control window. (This flag is ignored under Windows unless you specify `wxTR_NO_LINES` as well.)
- wxTR_LINES_AT_ROOT: Use this style to show lines between root nodes. Only applicable if `wxTR_HIDE_ROOT` is set and `wxTR_NO_LINES` is not set.
- wxTR_HIDE_ROOT: Use this style to suppress the display of the root node, effectively causing the first-level nodes to appear as a series of root nodes.
- wxTR_ROW_LINES: Use this style to draw a contrasting border between displayed rows.
- wxTR_HAS_VARIABLE_ROW_HEIGHT: Use this style to cause row heights to be just big enough to fit the content. If not set, all rows use the largest row height. The default is that this flag is unset. Generic only.
- wxTR_SINGLE: For convenience to document that only one item may be selected at a time. Selecting another item causes the current selection, if any, to be deselected. This is the default.
- wxTR_MULTIPLE: Use this style to allow a range of items to be selected. If a second range is selected, the current range, if any, is deselected.
- wxTR_DEFAULT_STYLE: The set of flags that are closest to the defaults for the native control for a particular toolkit.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxTreeEvent& event)

Event macros for events emitted by this class:

- EVT_TREE_BEGIN_DRAG(id, func): Begin dragging with the left mouse button. If you want to enable left-dragging you need to intercept this event and explicitly call wxTreeEvent::Allow(), as it's vetoed by default. Processes a `wxEVT_TREE_BEGIN_DRAG` event type.
- EVT_TREE_BEGIN_RDRAG(id, func): Begin dragging with the right mouse button. If you want to enable right-dragging you need to intercept this event and explicitly call wxTreeEvent::Allow(), as it's vetoed by default. Processes a `wxEVT_TREE_BEGIN_RDRAG` event type.
- EVT_TREE_END_DRAG(id, func): End dragging with the left or right mouse button. Processes a `wxEVT_TREE_END_DRAG` event type.
- EVT_TREE_BEGIN_LABEL_EDIT(id, func): Begin editing a label. This can be prevented by calling Veto(). Processes a `wxEVT_TREE_BEGIN_LABEL_EDIT` event type.
- EVT_TREE_END_LABEL_EDIT(id, func): Finish editing a label. This can be prevented by calling Veto(). Processes a `wxEVT_TREE_END_LABEL_EDIT` event type.
- EVT_TREE_DELETE_ITEM(id, func): An item was deleted. Processes a `wxEVT_TREE_DELETE_ITEM` event type.
- EVT_TREE_GET_INFO(id, func): Request information from the application. Processes a `wxEVT_TREE_GET_INFO` event type.
- EVT_TREE_SET_INFO(id, func): Information is being supplied. Processes a `wxEVT_TREE_SET_INFO` event type.
- EVT_TREE_ITEM_ACTIVATED(id, func): The item has been activated, i.e. chosen by double clicking it with mouse or from keyboard. Processes a `wxEVT_TREE_ITEM_ACTIVATED` event type.

- EVT_TREE_ITEM_COLLAPSED(id, func): The item has been collapsed. Processes a wxEVT_TREE_ITEM_COLLAPSED event type.
- EVT_TREE_ITEM_COLLAPSING(id, func): The item is being collapsed. This can be prevented by calling Veto(). Processes a wxEVT_TREE_ITEM_COLLAPSING event type.
- EVT_TREE_ITEM_EXPANDED(id, func): The item has been expanded. Processes a wxEVT_TREE_ITEM_EXPANDED event type.
- EVT_TREE_ITEM_EXPANDING(id, func): The item is being expanded. This can be prevented by calling Veto(). Processes a wxEVT_TREE_ITEM_EXPANDING event type.
- EVT_TREE_ITEM_RIGHT_CLICK(id, func): The user has clicked the item with the right mouse button. Processes a wxEVT_TREE_ITEM_RIGHT_CLICK event type.
- EVT_TREE_ITEM_MIDDLE_CLICK(id, func): The user has clicked the item with the middle mouse button. This is only supported by the generic control. Processes a wxEVT_TREE_ITEM_MIDDLE_CLICK event type.
- EVT_TREE_SEL_CHANGED(id, func): Selection has changed. Processes a wxEVT_TREE_SEL_CHANGED event type.
- EVT_TREE_SEL_CHANGING(id, func): Selection is changing. This can be prevented by calling Veto(). Processes a wxEVT_TREE_SEL_CHANGING event type.
- EVT_TREE_KEY_DOWN(id, func): A key has been pressed. Processes a wxEVT_TREE_KEY_DOWN event type.
- EVT_TREE_ITEM_GETTOOLTIP(id, func): The opportunity to set the item tooltip is being given to the application (call wxTreeEvent::SetToolTip). Windows only. Processes a wxEVT_TREE_ITEM_GETTOOLTIP event type.
- EVT_TREE_ITEM_MENU(id, func): The context menu for the selected item has been requested, either by a right click or by using the menu key. Processes a wxEVT_TREE_ITEM_MENU event type.
- EVT_TREE_STATE_IMAGE_CLICK(id, func): The state image has been clicked. Processes a wxEVT_TREE_STATE_IMAGE_CLICK event type.

See also Window Styles.

Win32 notes:

wxTreeCtrl class uses the standard common treeview control under Win32 implemented in the system library comctl32.dll. Some versions of this library are known to have bugs with handling the tree control colours: the usual symptom is that the expanded items leave black (or otherwise incorrectly coloured) background behind them, especially for the controls using non-default background colour. The recommended solution is to upgrade the comctl32.dll to a newer version: see http://www.microsoft.com/downloads/details.aspx?familyid=cb2cf3a2-8025-4e8f-8511-9b476a8d35d2

Library: wxCore

Category: Controls

See also

wxDataViewTreeCtrl, wxTreeEvent, wxTreeItemData, wxTreeCtrl Overview, wxListBox, wxListCtrl, wxImageList

Public Member Functions

- wxTreeCtrl ()

  Default Constructor.
- wxTreeCtrl (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTR_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxTreeCtrlNameStr)

  Constructor, creating and showing a tree control.
- virtual ~wxTreeCtrl ()

  Destructor, destroying the tree control.
- virtual wxTreeItemId AddRoot (const wxString &text, int image=-1, int selImage=-1, wxTreeItemData *data=NULL)

  Adds the root node to the tree, returning the new item.
- wxTreeItemId AppendItem (const wxTreeItemId &parent, const wxString &text, int image=-1, int selImage=-1, wxTreeItemData *data=NULL)

  Appends an item to the end of the branch identified by parent, return a new item id.
- void AssignButtonsImageList (wxImageList *imageList)

  Sets the buttons image list.
- void AssignImageList (wxImageList *imageList)

  Sets the normal image list.
- void AssignStateImageList (wxImageList *imageList)

  Sets the state image list.
- virtual void Collapse (const wxTreeItemId &item)

  Collapses the given item.
- void CollapseAll ()

  Collapses the root item.
- void CollapseAllChildren (const wxTreeItemId &item)

  Collapses this item and all of its children, recursively.
- virtual void CollapseAndReset (const wxTreeItemId &item)

  Collapses the given item and removes all children.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTR_DEFAULT_STYLE, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxTreeCtrlNameStr)

  Creates the tree control.
- virtual void Delete (const wxTreeItemId &item)

  Deletes the specified item.
- virtual void DeleteAllItems ()

  Deletes all items in the control.
- virtual void DeleteChildren (const wxTreeItemId &item)

  Deletes all children of the given item (but not the item itself).
- virtual wxTextCtrl * EditLabel (const wxTreeItemId &item, wxClassInfo *textCtrlClass=wxCLASSINFO(wxTextCtrl))

  Starts editing the label of the given item.
- void EnableBellOnNoMatch (bool on=true)

  Enable or disable a beep if there is no match for the currently entered text when searching for the item from keyboard.
- virtual void EndEditLabel (const wxTreeItemId &item, bool discardChanges=false)

  Ends label editing.
- virtual void EnsureVisible (const wxTreeItemId &item)

  Scrolls and/or expands items to ensure that the given item is visible.
- virtual void Expand (const wxTreeItemId &item)

  Expands the given item.
- void ExpandAll ()

Expands all items in the tree.

- void ExpandAllChildren (const wxTreeItemId &item)

  Expands the given item and all its children recursively.
- virtual bool GetBoundingRect (const wxTreeItemId &item, wxRect &rect, bool textOnly=false) const

  Retrieves the rectangle bounding the item.
- wxImageList * GetButtonsImageList () const

  Returns the buttons image list (from which application-defined button images are taken).
- virtual size_t GetChildrenCount (const wxTreeItemId &item, bool recursively=true) const

  Returns the number of items in the branch.
- virtual unsigned int GetCount () const

  Returns the number of items in the control.
- virtual wxTextCtrl * GetEditControl () const

  Returns the edit control being currently used to edit a label.
- virtual wxTreeItemId GetFirstChild (const wxTreeItemId &item, wxTreeItemIdValue &cookie) const

  Returns the first child; call GetNextChild() for the next child.
- virtual wxTreeItemId GetFirstVisibleItem () const

  Returns the first visible item.
- virtual wxTreeItemId GetFocusedItem () const

  Returns the item last clicked or otherwise selected.
- virtual void ClearFocusedItem ()

  Clears the currently focused item.
- virtual void SetFocusedItem (const wxTreeItemId &item)

  Sets the currently focused item.
- wxImageList * GetImageList () const

  Returns the normal image list.
- virtual unsigned int GetIndent () const

  Returns the current tree control indentation.
- virtual wxColour GetItemBackgroundColour (const wxTreeItemId &item) const

  Returns the background colour of the item.
- virtual wxTreeItemData * GetItemData (const wxTreeItemId &item) const

  Returns the tree item data associated with the item.
- virtual wxFont GetItemFont (const wxTreeItemId &item) const

  Returns the font of the item label.
- virtual int GetItemImage (const wxTreeItemId &item, wxTreeItemIcon which=wxTreeItemIcon_Normal) const

  Gets the specified item image.
- virtual wxTreeItemId GetItemParent (const wxTreeItemId &item) const

  Returns the item's parent.
- int GetItemState (const wxTreeItemId &item) const

  Gets the specified item state.
- virtual wxString GetItemText (const wxTreeItemId &item) const

  Returns the item label.
- virtual wxColour GetItemTextColour (const wxTreeItemId &item) const

  Returns the colour of the item label.
- virtual wxTreeItemId GetLastChild (const wxTreeItemId &item) const

  Returns the last child of the item (or an invalid tree item if this item has no children).
- virtual wxTreeItemId GetNextChild (const wxTreeItemId &item, wxTreeItemIdValue &cookie) const

  Returns the next child; call GetFirstChild() for the first child.
- virtual wxTreeItemId GetNextSibling (const wxTreeItemId &item) const

  Returns the next sibling of the specified item; call GetPrevSibling() for the previous sibling.
- virtual wxTreeItemId GetNextVisible (const wxTreeItemId &item) const

  Returns the next visible item or an invalid item if this item is the last visible one.

- virtual wxTreeItemId GetPrevSibling (const wxTreeItemId &item) const

  Returns the previous sibling of the specified item; call GetNextSibling() for the next sibling.
- virtual wxTreeItemId GetPrevVisible (const wxTreeItemId &item) const

  Returns the previous visible item or an invalid item if this item is the first visible one.
- bool GetQuickBestSize () const

  Returns true if the control will use a quick calculation for the best size, looking only at the first and last items.
- virtual wxTreeItemId GetRootItem () const

  Returns the root item for the tree control.
- virtual wxTreeItemId GetSelection () const

  Returns the selection, or an invalid item if there is no selection.
- virtual size_t GetSelections (wxArrayTreeItemIds &selection) const

  Fills the array of tree items passed in with the currently selected items.
- wxImageList * GetStateImageList () const

  Returns the state image list (from which application-defined state images are taken).
- wxTreeItemId HitTest (const wxPoint &point, int &flags) const

  Calculates which (if any) item is under the given point, returning the tree item id at this point plus extra information flags.
- wxTreeItemId InsertItem (const wxTreeItemId &parent, const wxTreeItemId &previous, const wxString &text, int image=-1, int selImage=-1, wxTreeItemData *data=NULL)

  Inserts an item after a given one (previous).
- wxTreeItemId InsertItem (const wxTreeItemId &parent, size_t pos, const wxString &text, int image=-1, int selImage=-1, wxTreeItemData *data=NULL)

  Inserts an item before one identified by its position (pos).
- virtual bool IsBold (const wxTreeItemId &item) const

  Returns true if the given item is in bold state.
- bool IsEmpty () const

  Returns true if the control is empty (i.e. has no items, even no root one).
- virtual bool IsExpanded (const wxTreeItemId &item) const

  Returns true if the item is expanded (only makes sense if it has children).
- virtual bool IsSelected (const wxTreeItemId &item) const

  Returns true if the item is selected.
- virtual bool IsVisible (const wxTreeItemId &item) const

  Returns true if the item is visible on the screen.
- virtual bool ItemHasChildren (const wxTreeItemId &item) const

  Returns true if the item has children.
- virtual int OnCompareItems (const wxTreeItemId &item1, const wxTreeItemId &item2)

  Override this function in the derived class to change the sort order of the items in the tree control.
- wxTreeItemId PrependItem (const wxTreeItemId &parent, const wxString &text, int image=-1, int selImage=-1, wxTreeItemData *data=NULL)

  Appends an item as the first child of parent, return a new item id.
- virtual void ScrollTo (const wxTreeItemId &item)

  Scrolls the specified item into view.
- virtual void SelectItem (const wxTreeItemId &item, bool select=true)

  Selects the given item.
- void SetButtonsImageList (wxImageList *imageList)

  Sets the buttons image list (from which application-defined button images are taken).
- virtual void SetImageList (wxImageList *imageList)

  Sets the normal image list.
- virtual void SetIndent (unsigned int indent)

  Sets the indentation for the tree control.
- virtual void SetItemBackgroundColour (const wxTreeItemId &item, const wxColour &col)

Sets the colour of the item's background.

- virtual void SetItemBold (const wxTreeItemId &item, bool bold=true)

  Makes item appear in bold font if bold parameter is true or resets it to the normal state.
- virtual void SetItemData (const wxTreeItemId &item, wxTreeItemData *data)

  Sets the item client data.
- virtual void SetItemDropHighlight (const wxTreeItemId &item, bool highlight=true)

  Gives the item the visual feedback for Drag'n'Drop actions, which is useful if something is dragged from the outside onto the tree control (as opposed to a DnD operation within the tree control, which already is implemented internally).
- virtual void SetItemFont (const wxTreeItemId &item, const wxFont &font)

  Sets the item's font.
- virtual void SetItemHasChildren (const wxTreeItemId &item, bool hasChildren=true)

  Force appearance of the button next to the item.
- virtual void SetItemImage (const wxTreeItemId &item, int image, wxTreeItemIcon which=wxTreeItemIcon_Normal)

  Sets the specified item's image.
- void SetItemState (const wxTreeItemId &item, int state)

  Sets the specified item state.
- virtual void SetItemText (const wxTreeItemId &item, const wxString &text)

  Sets the item label.
- virtual void SetItemTextColour (const wxTreeItemId &item, const wxColour &col)

  Sets the colour of the item's text.
- void SetQuickBestSize (bool quickBestSize)

  If true is passed, specifies that the control will use a quick calculation for the best size, looking only at the first and last items.
- virtual void SetStateImageList (wxImageList *imageList)

  Sets the state image list (from which application-defined state images are taken).
- void SetWindowStyle (long styles)

  Sets the mode flags associated with the display of the tree control.
- virtual void SortChildren (const wxTreeItemId &item)

  Sorts the children of the given item using OnCompareItems().
- virtual void Toggle (const wxTreeItemId &item)

  Toggles the given item between collapsed and expanded states.
- void ToggleItemSelection (const wxTreeItemId &item)

  Toggles the given item between selected and unselected states.
- virtual void Unselect ()

  Removes the selection from the currently selected item (if any).
- virtual void UnselectAll ()

  This function either behaves the same as Unselect() if the control doesn't have `wxTR_MULTIPLE` style, or removes the selection from all items if it does have this style.
- void UnselectItem (const wxTreeItemId &item)

  Unselects the given item.
- virtual void SelectChildren (const wxTreeItemId &parent)

  Select all the immediate children of the given parent.

Additional Inherited Members

21.806.2 Constructor & Destructor Documentation

wxTreeCtrl::wxTreeCtrl ( )

Default Constructor.

wxTreeCtrl::wxTreeCtrl (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTR_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = `wxTreeCtrlNameStr`)

Constructor, creating and showing a tree control.

Parameters

parent	Parent window. Must not be NULL.
id	Window identifier. The value `wxID_ANY` indicates a default value.
pos	Window position. If wxDefaultPosition is specified then a default position is chosen.
size	Window size. If wxDefaultSize is specified then the window is sized appropriately.
style	Window style. See wxTreeCtrl.
validator	Window validator.
name	Window name.

See also

Create(), wxValidator

virtual wxTreeCtrl::~wxTreeCtrl ( ) [virtual]

Destructor, destroying the tree control.

21.806.3 Member Function Documentation

virtual wxTreeItemId wxTreeCtrl::AddRoot (const wxString & *text*, int *image* = −1, int *selImage* = −1, wxTreeItemData * *data* = NULL) [virtual]

Adds the root node to the tree, returning the new item.

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* -1 and *selImage* is -1, the same image is used for both selected and unselected items.

wxTreeItemId wxTreeCtrl::AppendItem (const wxTreeItemId & *parent*, const wxString & *text*, int *image* = −1, int *selImage* = −1, wxTreeItemData * *data* = NULL)

Appends an item to the end of the branch identified by *parent*, return a new item id.

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* > -1 and *selImage* is -1, the same image is used for both selected and unselected items.

void wxTreeCtrl::AssignButtonsImageList (wxImageList * *imageList*)

Sets the buttons image list.

The button images assigned with this method will be automatically deleted by wxTreeCtrl as appropriate (i.e. it takes ownership of the list).

Setting or assigning the button image list enables the display of image buttons. Once enabled, the only way to disable the display of button images is to set the button image list to NULL.

This function is only available in the generic version.

See also

SetButtonsImageList().

void wxTreeCtrl::AssignImageList (wxImageList * *imageList*)

Sets the normal image list.

The image list assigned with this method will be automatically deleted by wxTreeCtrl as appropriate (i.e. it takes ownership of the list).

See also

SetImageList().

void wxTreeCtrl::AssignStateImageList (wxImageList * *imageList*)

Sets the state image list.

Image list assigned with this method will be automatically deleted by wxTreeCtrl as appropriate (i.e. it takes ownership of the list).

See also

SetStateImageList().

virtual void wxTreeCtrl::ClearFocusedItem ( ) [virtual]

Clears the currently focused item.

Since

2.9.1

virtual void wxTreeCtrl::Collapse (const wxTreeItemId & *item*) [virtual]

Collapses the given item.

void wxTreeCtrl::CollapseAll ( )

Collapses the root item.

See also

ExpandAll()

void wxTreeCtrl::CollapseAllChildren (const wxTreeItemId & *item*)

Collapses this item and all of its children, recursively.

See also

ExpandAllChildren()

virtual void wxTreeCtrl::CollapseAndReset (const wxTreeItemId & *item*) [virtual]

Collapses the given item and removes all children.

bool wxTreeCtrl::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTR_DEFAULT_STYLE, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxTreeCtrlNameStr)

Creates the tree control.

See wxTreeCtrl::wxTreeCtrl() for further details.

virtual void wxTreeCtrl::Delete (const wxTreeItemId & *item*) [virtual]

Deletes the specified item.

A EVT_TREE_DELETE_ITEM event will be generated.

This function may cause a subsequent call to GetNextChild() to fail.

virtual void wxTreeCtrl::DeleteAllItems ( ) [virtual]

Deletes all items in the control.

Note that this may not generate EVT_TREE_DELETE_ITEM events under some Windows versions although normally such event is generated for each removed item.

virtual void wxTreeCtrl::DeleteChildren (const wxTreeItemId & *item*) [virtual]

Deletes all children of the given item (but not the item itself).

Note that this will **not** generate any events unlike Delete() method.

If you have called SetItemHasChildren(), you may need to call it again since DeleteChildren() does not automatically clear the setting.

virtual wxTextCtrl* wxTreeCtrl::EditLabel (const wxTreeItemId & *item*, wxClassInfo * *textCtrlClass* = wxCLASSINFO(wxTextCtrl)) [virtual]

Starts editing the label of the given *item*.

This function generates a EVT_TREE_BEGIN_LABEL_EDIT event which can be vetoed so that no text control will appear for in-place editing.

If the user changed the label (i.e. s/he does not press ESC or leave the text control without changes, a EVT_TREE_END_LABEL_EDIT event will be sent which can be vetoed as well.

See also

EndEditLabel(), wxTreeEvent

void wxTreeCtrl::EnableBellOnNoMatch (bool *on* = true)

Enable or disable a beep if there is no match for the currently entered text when searching for the item from keyboard.

The default is to not beep in this case except in wxMSW where the beep is always generated by the native control and cannot be disabled, i.e. calls to this function do nothing there.

Since

2.9.5

virtual void wxTreeCtrl::EndEditLabel (const wxTreeItemId & *item,* bool *discardChanges* = `false`) `[virtual]`

Ends label editing.

If *cancelEdit* is true, the edit will be cancelled.

Note

This function is currently supported under Windows only.

See also

EditLabel()

virtual void wxTreeCtrl::EnsureVisible (const wxTreeItemId & *item*) `[virtual]`

Scrolls and/or expands items to ensure that the given item is visible.

virtual void wxTreeCtrl::Expand (const wxTreeItemId & *item*) `[virtual]`

Expands the given item.

void wxTreeCtrl::ExpandAll ( )

Expands all items in the tree.

void wxTreeCtrl::ExpandAllChildren (const wxTreeItemId & *item*)

Expands the given item and all its children recursively.

virtual bool wxTreeCtrl::GetBoundingRect (const wxTreeItemId & *item,* wxRect & *rect,* bool *textOnly* = `false`) const `[virtual]`

Retrieves the rectangle bounding the *item*.

If *textOnly* is true, only the rectangle around the item's label will be returned, otherwise the item's image is also taken into account.

The return value is true if the rectangle was successfully retrieved or false if it was not (in this case *rect* is not changed) – for example, if the item is currently invisible.

Notice that the rectangle coordinates are logical, not physical ones. So, for example, the x coordinate may be negative if the tree has a horizontal scrollbar and its position is not 0.

wxPerl Note: In wxPerl this method only takes the *item* and *textOnly* parameters and returns a `Wx::Rect` (or `undef`).

wxImageList* wxTreeCtrl::GetButtonsImageList ( ) const

Returns the buttons image list (from which application-defined button images are taken).

This function is only available in the generic version.

virtual size_t wxTreeCtrl::GetChildrenCount (const wxTreeItemId & *item,* bool *recursively* = `true`) const `[virtual]`

Returns the number of items in the branch.

If *recursively* is true, returns the total number of descendants, otherwise only one level of children is counted.

virtual unsigned int wxTreeCtrl::GetCount ( ) const `[virtual]`

Returns the number of items in the control.

virtual wxTextCtrl* wxTreeCtrl::GetEditControl ( ) const `[virtual]`

Returns the edit control being currently used to edit a label.

Returns NULL if no label is being edited.

Note

This is currently only implemented for wxMSW.

virtual wxTreeItemId wxTreeCtrl::GetFirstChild (const wxTreeItemId & *item*, wxTreeItemIdValue & *cookie*) const `[virtual]`

Returns the first child; call GetNextChild() for the next child.

For this enumeration function you must pass in a 'cookie' parameter which is opaque for the application but is necessary for the library to make these functions reentrant (i.e. allow more than one enumeration on one and the same object simultaneously). The cookie passed to GetFirstChild() and GetNextChild() should be the same variable.

Returns an invalid tree item (i.e. wxTreeItemId::IsOk() returns false) if there are no further children.

wxPerl Note: In wxPerl this method only takes the *item* parameter, and returns a 2-element list (item, cookie).

See also

GetNextChild(), GetNextSibling()

virtual wxTreeItemId wxTreeCtrl::GetFirstVisibleItem ( ) const `[virtual]`

Returns the first visible item.

virtual wxTreeItemId wxTreeCtrl::GetFocusedItem ( ) const `[virtual]`

Returns the item last clicked or otherwise selected.

Unlike GetSelection(), it can be used whether or not the control has the `wxTR_MULTIPLE` style.

Since

2.9.1

wxImageList* wxTreeCtrl::GetImageList ( ) const

Returns the normal image list.

virtual unsigned int wxTreeCtrl::GetIndent ( ) const `[virtual]`

Returns the current tree control indentation.

virtual **wxColour** wxTreeCtrl::GetItemBackgroundColour (const **wxTreeItemId** & *item*) const [virtual]

Returns the background colour of the item.

virtual **wxTreeItemData**∗ wxTreeCtrl::GetItemData (const **wxTreeItemId** & *item*) const [virtual]

Returns the tree item data associated with the item.

See also

wxTreeItemData

wxPerl Note: wxPerl provides the following shortcut method:

- GetPlData(item): returns the Perl data associated with the Wx::TreeItemData. It is just the same as tree->GetItemData(item)->GetData().

virtual **wxFont** wxTreeCtrl::GetItemFont (const **wxTreeItemId** & *item*) const [virtual]

Returns the font of the item label.

If the font hadn't been explicitly set for the specified *item* with SetItemFont(), returns an invalid wxNullFont font. GetFont() can be used to retrieve the global tree control font used for the items without any specific font.

virtual int wxTreeCtrl::GetItemImage (const **wxTreeItemId** & *item,* **wxTreeItemIcon** *which* = **wxTreeItemIcon_Normal**) const [virtual]

Gets the specified item image.

The value of *which* may be:

- wxTreeItemIcon_Normal: to get the normal item image.
- wxTreeItemIcon_Selected: to get the selected item image (i.e. the image which is shown when the item is currently selected).
- wxTreeItemIcon_Expanded: to get the expanded image (this only makes sense for items which have children - then this image is shown when the item is expanded and the normal image is shown when it is collapsed).
- wxTreeItemIcon_SelectedExpanded: to get the selected expanded image (which is shown when an expanded item is currently selected).

virtual **wxTreeItemId** wxTreeCtrl::GetItemParent (const **wxTreeItemId** & *item*) const [virtual]

Returns the item's parent.

int wxTreeCtrl::GetItemState (const **wxTreeItemId** & *item*) const

Gets the specified item state.

virtual **wxString** wxTreeCtrl::GetItemText (const **wxTreeItemId** & *item*) const [virtual]

Returns the item label.

virtual **wxColour** wxTreeCtrl::GetItemTextColour (const **wxTreeItemId** & *item*) const [virtual]

Returns the colour of the item label.

virtual **wxTreeItemId** wxTreeCtrl::GetLastChild (const **wxTreeItemId** & *item*) const [virtual]

Returns the last child of the item (or an invalid tree item if this item has no children).

See also

GetFirstChild(), GetNextSibling(), GetLastChild()

virtual **wxTreeItemId** wxTreeCtrl::GetNextChild (const **wxTreeItemId** & *item,* wxTreeItemIdValue & *cookie*) const [virtual]

Returns the next child; call GetFirstChild() for the first child.

For this enumeration function you must pass in a 'cookie' parameter which is opaque for the application but is necessary for the library to make these functions reentrant (i.e. allow more than one enumeration on one and the same object simultaneously). The cookie passed to GetFirstChild() and GetNextChild() should be the same.

Returns an invalid tree item if there are no further children.

wxPerl Note: In wxPerl this method returns a 2-element list (item, cookie) instead of modifying its parameters.

See also

GetFirstChild()

virtual **wxTreeItemId** wxTreeCtrl::GetNextSibling (const **wxTreeItemId** & *item*) const [virtual]

Returns the next sibling of the specified item; call GetPrevSibling() for the previous sibling.

Returns an invalid tree item if there are no further siblings.

See also

GetPrevSibling()

virtual **wxTreeItemId** wxTreeCtrl::GetNextVisible (const **wxTreeItemId** & *item*) const [virtual]

Returns the next visible item or an invalid item if this item is the last visible one.

Note

The *item* itself must be visible.

virtual **wxTreeItemId** wxTreeCtrl::GetPrevSibling (const **wxTreeItemId** & *item*) const [virtual]

Returns the previous sibling of the specified item; call GetNextSibling() for the next sibling.

Returns an invalid tree item if there are no further children.

See also

GetNextSibling()

virtual **wxTreeItemId** **wxTreeCtrl::GetPrevVisible** (const **wxTreeItemId** & *item*) const [virtual]

Returns the previous visible item or an invalid item if this item is the first visible one.

Note

The *item* itself must be visible.

bool wxTreeCtrl::GetQuickBestSize ( ) const

Returns true if the control will use a quick calculation for the best size, looking only at the first and last items.

The default is false.

See also

SetQuickBestSize()

virtual **wxTreeItemId** **wxTreeCtrl::GetRootItem** ( ) const [virtual]

Returns the root item for the tree control.

virtual **wxTreeItemId** **wxTreeCtrl::GetSelection** ( ) const [virtual]

Returns the selection, or an invalid item if there is no selection.

This function only works with the controls without `wxTR_MULTIPLE` style, use GetSelections() for the controls which do have this style or, if a single item is wanted, use GetFocusedItem().

virtual **size_t** **wxTreeCtrl::GetSelections** (**wxArrayTreeItemIds** & *selection*) const [virtual]

Fills the array of tree items passed in with the currently selected items.

This function can be called only if the control has the `wxTR_MULTIPLE` style.

Returns the number of selected items.

wxPerl Note: In wxPerl this method takes no parameters and returns a list of `Wx::TreeItemId`.

wxImageList* **wxTreeCtrl::GetStateImageList** ( ) const

Returns the state image list (from which application-defined state images are taken).

wxTreeItemId **wxTreeCtrl::HitTest** (const **wxPoint** & *point,* int & *flags*) const

Calculates which (if any) item is under the given *point*, returning the tree item id at this point plus extra information *flags*.

flags is a bitlist of the following:

- `wxTREE_HITTEST_ABOVE`: Above the client area.
- `wxTREE_HITTEST_BELOW`: Below the client area.
- `wxTREE_HITTEST_NOWHERE`: In the client area but below the last item.
- `wxTREE_HITTEST_ONITEMBUTTON`: On the button associated with an item.

- `wxTREE_HITTEST_ONITEMICON`: On the bitmap associated with an item.
- `wxTREE_HITTEST_ONITEMINDENT`: In the indentation associated with an item.
- `wxTREE_HITTEST_ONITEMLABEL`: On the label (string) associated with an item.
- `wxTREE_HITTEST_ONITEMRIGHT`: In the area to the right of an item.
- `wxTREE_HITTEST_ONITEMSTATEICON`: On the state icon for a tree view item that is in a user-defined state.
- `wxTREE_HITTEST_TOLEFT`: To the right of the client area.
- `wxTREE_HITTEST_TORIGHT`: To the left of the client area.

wxPerl Note: In wxPerl this method only takes the *point* parameter and returns a 2-element list (item, flags).

wxTreeItemId wxTreeCtrl::InsertItem (const wxTreeItemId & *parent,* const wxTreeItemId & *previous,* const wxString & *text,* int *image* = −1, int *selImage* = −1, wxTreeItemData * *data* = NULL)

Inserts an item after a given one (*previous*).

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* -1 and *selImage* is -1, the same image is used for both selected and unselected items.

wxTreeItemId wxTreeCtrl::InsertItem (const wxTreeItemId & *parent,* size_t *pos,* const wxString & *text,* int *image* = −1, int *selImage* = −1, wxTreeItemData * *data* = NULL)

Inserts an item before one identified by its position (*pos*).

pos must be less than or equal to the number of children.

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* -1 and *selImage* is -1, the same image is used for both selected and unselected items.

virtual bool wxTreeCtrl::IsBold (const wxTreeItemId & *item*) const [virtual]

Returns true if the given item is in bold state.

See also

SetItemBold()

bool wxTreeCtrl::IsEmpty ( ) const

Returns true if the control is empty (i.e. has no items, even no root one).

virtual bool wxTreeCtrl::IsExpanded (const wxTreeItemId & *item*) const [virtual]

Returns true if the item is expanded (only makes sense if it has children).

virtual bool wxTreeCtrl::IsSelected (const wxTreeItemId & *item*) const [virtual]

Returns true if the item is selected.

virtual bool wxTreeCtrl::IsVisible (const wxTreeItemId & *item*) const [virtual]

Returns true if the item is visible on the screen.

virtual bool wxTreeCtrl::ItemHasChildren (const wxTreeItemId & *item*) const [virtual]

Returns true if the item has children.

virtual int wxTreeCtrl::OnCompareItems (const wxTreeItemId & *item1*, const wxTreeItemId & *item2*) [virtual]

Override this function in the derived class to change the sort order of the items in the tree control.

The function should return a negative, zero or positive value if the first item is less than, equal to or greater than the second one.

Please note that you **must** use wxRTTI macros wxDECLARE_DYNAMIC_CLASS() and wxIMPLEMENT_DYNAMIC_CLASS() if you override this function because otherwise the base class considers that it is not overridden and uses the default comparison, i.e. sorts the items alphabetically, which allows it optimize away the calls to the virtual function completely.

See also

SortChildren()

wxTreeItemId wxTreeCtrl::PrependItem (const wxTreeItemId & *parent*, const wxString & *text*, int *image* = -1, int *selImage* = -1, wxTreeItemData * *data* = NULL)

Appends an item as the first child of *parent*, return a new item id.

The *image* and *selImage* parameters are an index within the normal image list specifying the image to use for unselected and selected items, respectively. If *image* -1 and *selImage* is -1, the same image is used for both selected and unselected items.

virtual void wxTreeCtrl::ScrollTo (const wxTreeItemId & *item*) [virtual]

Scrolls the specified item into view.

virtual void wxTreeCtrl::SelectChildren (const wxTreeItemId & *parent*) [virtual]

Select all the immediate children of the given parent.

This function can be used with multiselection controls only.

Since

2.9.1

virtual void wxTreeCtrl::SelectItem (const wxTreeItemId & *item*, bool *select* = true) [virtual]

Selects the given item.

In multiple selection controls, can be also used to deselect a currently selected item if the value of *select* is false.

Notice that calling this method will generate wxEVT_TREE_SEL_CHANGING and wxEVT_TREE_SEL_CHANGED events and that the change could be vetoed by the former event handler.

void wxTreeCtrl::SetButtonsImageList (wxImageList * *imageList*)

Sets the buttons image list (from which application-defined button images are taken).

The button images assigned with this method will **not** be deleted by wxTreeCtrl's destructor, you must delete it yourself. Setting or assigning the button image list enables the display of image buttons. Once enabled, the only way to disable the display of button images is to set the button image list to NULL.

Note

This function is only available in the generic version.

See also

AssignButtonsImageList().

virtual void wxTreeCtrl::SetFocusedItem (const wxTreeItemId & *item*) [virtual]

Sets the currently focused item.

Parameters

item	The item to make the current one. It must be valid.

Since

2.9.1

virtual void wxTreeCtrl::SetImageList (wxImageList * *imageList*) [virtual]

Sets the normal image list.

The image list assigned with this method will **not** be deleted by wxTreeCtrl's destructor, you must delete it yourself.

See also

AssignImageList().

virtual void wxTreeCtrl::SetIndent (unsigned int *indent*) [virtual]

Sets the indentation for the tree control.

virtual void wxTreeCtrl::SetItemBackgroundColour (const wxTreeItemId & *item*, const wxColour & *col*) [virtual]

Sets the colour of the item's background.

virtual void wxTreeCtrl::SetItemBold (const wxTreeItemId & *item*, bool *bold* = true) [virtual]

Makes item appear in bold font if *bold* parameter is true or resets it to the normal state.

See also

IsBold()

virtual void wxTreeCtrl::SetItemData (const wxTreeItemId & *item,* wxTreeItemData * *data*) [virtual]

Sets the item client data.

Notice that the client data previously associated with the *item* (if any) is *not* freed by this function and so calling this function multiple times for the same item will result in memory leaks unless you delete the old item data pointer yourself.

wxPerl Note: wxPerl provides the following shortcut method:

- SetPlData(item, data): sets the Perl data associated with the `Wx::TreeItemData`. It is just the same as tree->GetItemData(item)->SetData(data).

virtual void wxTreeCtrl::SetItemDropHighlight (const wxTreeItemId & *item,* bool *highlight* = `true`) [virtual]

Gives the item the visual feedback for Drag'n'Drop actions, which is useful if something is dragged from the outside onto the tree control (as opposed to a DnD operation within the tree control, which already is implemented internally).

virtual void wxTreeCtrl::SetItemFont (const wxTreeItemId & *item,* const wxFont & *font*) [virtual]

Sets the item's font.

All items in the tree should have the same height to avoid text clipping, so the fonts height should be the same for all of them, although font attributes may vary.

See also

SetItemBold()

virtual void wxTreeCtrl::SetItemHasChildren (const wxTreeItemId & *item,* bool *hasChildren* = `true`) [virtual]

Force appearance of the button next to the item.

This is useful to allow the user to expand the items which don't have any children now, but instead adding them only when needed, thus minimizing memory usage and loading time.

virtual void wxTreeCtrl::SetItemImage (const wxTreeItemId & *item,* int *image,* wxTreeItemIcon *which* = wxTreeItemIcon_Normal) [virtual]

Sets the specified item's image.

See GetItemImage() for the description of the *which* parameter.

void wxTreeCtrl::SetItemState (const wxTreeItemId & *item,* int *state*)

Sets the specified item state.

The value of *state* may be:

- `wxTREE_ITEMSTATE_NONE`: to disable the item state (the state image will be not displayed).
- `wxTREE_ITEMSTATE_NEXT`: to set the next item state.
- `wxTREE_ITEMSTATE_PREV`: to set the previous item state.

virtual void wxTreeCtrl::SetItemText (const wxTreeItemId & *item,* const wxString & *text*) [virtual]

Sets the item label.

virtual void wxTreeCtrl::SetItemTextColour (const wxTreeItemId & *item,* const wxColour & *col*) [virtual]

Sets the colour of the item's text.

void wxTreeCtrl::SetQuickBestSize (bool *quickBestSize*)

If true is passed, specifies that the control will use a quick calculation for the best size, looking only at the first and last items.

Otherwise, it will look at all items. The default is false.

See also

GetQuickBestSize()

virtual void wxTreeCtrl::SetStateImageList (wxImageList * *imageList*) [virtual]

Sets the state image list (from which application-defined state images are taken).

Image list assigned with this method will **not** be deleted by wxTreeCtrl's destructor, you must delete it yourself.

See also

AssignStateImageList().

void wxTreeCtrl::SetWindowStyle (long *styles*)

Sets the mode flags associated with the display of the tree control.

The new mode takes effect immediately.

Note

Generic only; MSW ignores changes.

virtual void wxTreeCtrl::SortChildren (const wxTreeItemId & *item*) [virtual]

Sorts the children of the given item using OnCompareItems().

You should override that method to change the sort order (the default is ascending case-sensitive alphabetical order).

See also

wxTreeItemData, OnCompareItems()

virtual void wxTreeCtrl::Toggle (const wxTreeItemId & *item*) [virtual]

Toggles the given item between collapsed and expanded states.

void wxTreeCtrl::ToggleItemSelection (const wxTreeItemId & *item*)

Toggles the given item between selected and unselected states.

For multiselection controls only.

virtual void wxTreeCtrl::Unselect ( ) [virtual]

Removes the selection from the currently selected item (if any).

virtual void wxTreeCtrl::UnselectAll ( ) [virtual]

This function either behaves the same as Unselect() if the control doesn't have `wxTR_MULTIPLE` style, or removes the selection from all items if it does have this style.

void wxTreeCtrl::UnselectItem (const wxTreeItemId & *item*)

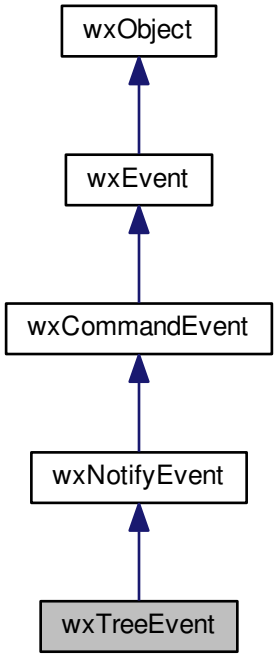
Unselects the given item.

This works in multiselection controls only.

21.807 wxTreeEvent Class Reference

`#include <wx/treectrl.h>`

Inheritance diagram for wxTreeEvent:



21.807.1 Detailed Description

A tree event holds information about events associated with wxTreeCtrl objects.

To process input from a tree control, use these event handler macros to direct input to member functions that take a wxTreeEvent argument.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxTreeEvent& event)

Event macros:

- EVT_TREE_BEGIN_DRAG(id, func): Begin dragging with the left mouse button. If you want to enable left-dragging you need to intercept this event and explicitly call wxTreeEvent::Allow(), as it's vetoed by default. Also notice that the control must have an associated image list (see SetImageList()) to drag its items under MSW.
- EVT_TREE_BEGIN_RDRAG(id, func): Begin dragging with the right mouse button. If you want to enable right-dragging you need to intercept this event and explicitly call wxTreeEvent::Allow(), as it's vetoed by default.
- EVT_TREE_END_DRAG(id, func): End dragging with the left or right mouse button.
- EVT_TREE_BEGIN_LABEL_EDIT(id, func): Begin editing a label. This can be prevented by calling Veto().
- EVT_TREE_END_LABEL_EDIT(id, func): Finish editing a label. This can be prevented by calling Veto().
- EVT_TREE_DELETE_ITEM(id, func): Delete an item.
- EVT_TREE_GET_INFO(id, func): Request information from the application.
- EVT_TREE_SET_INFO(id, func): Information is being supplied.
- EVT_TREE_ITEM_ACTIVATED(id, func): The item has been activated, i.e. chosen by double clicking it with mouse or from keyboard.
- EVT_TREE_ITEM_COLLAPSED(id, func): The item has been collapsed.
- EVT_TREE_ITEM_COLLAPSING(id, func): The item is being collapsed. This can be prevented by calling Veto().
- EVT_TREE_ITEM_EXPANDED(id, func): The item has been expanded.
- EVT_TREE_ITEM_EXPANDING(id, func): The item is being expanded. This can be prevented by calling Veto().
- EVT_TREE_ITEM_RIGHT_CLICK(id, func): The user has clicked the item with the right mouse button.
- EVT_TREE_ITEM_MIDDLE_CLICK(id, func): The user has clicked the item with the middle mouse button.
- EVT_TREE_SEL_CHANGED(id, func): Selection has changed.
- EVT_TREE_SEL_CHANGING(id, func): Selection is changing. This can be prevented by calling Veto().
- EVT_TREE_KEY_DOWN(id, func): A key has been pressed.
- EVT_TREE_ITEM_GETTOOLTIP(id, func): The opportunity to set the item tooltip is being given to the application (call SetToolTip()). Windows only.
- EVT_TREE_ITEM_MENU(id, func): The context menu for the selected item has been requested, either by a right click or by using the menu key.
- EVT_TREE_STATE_IMAGE_CLICK(id, func): The state image has been clicked.

Library: wxCore

Category: Events

See also

wxTreeCtrl

Public Member Functions

- wxTreeEvent (wxEventType commandType, wxTreeCtrl *tree, const wxTreeItemId &item=wxTreeItemId())

  Constructor, used by wxWidgets itself only.
- wxTreeItemId GetItem () const

  Returns the item (valid for all events).
- int GetKeyCode () const

  Returns the key code if the event is a key event.
- const wxKeyEvent & GetKeyEvent () const

  Returns the key event for `EVT_TREE_KEY_DOWN` events.
- const wxString & GetLabel () const

  Returns the label if the event is a begin or end edit label event.
- wxTreeItemId GetOldItem () const

  Returns the old item index (valid for `EVT_TREE_ITEM_CHANGING` and `EVT_TREE_ITEM_CHANGED` events).
- wxPoint GetPoint () const

  Returns the position of the mouse pointer if the event is a drag or menu-context event.
- bool IsEditCancelled () const

  Returns true if the label edit was cancelled.
- void SetToolTip (const wxString &tooltip)

  Set the tooltip for the item (valid for `EVT_TREE_ITEM_GETTOOLTIP` events).

Additional Inherited Members

21.807.2 Constructor & Destructor Documentation

wxTreeEvent::wxTreeEvent (wxEventType *commandType,* wxTreeCtrl * *tree,* const wxTreeItemId & *item =* wxTreeItemId ())

Constructor, used by wxWidgets itself only.

21.807.3 Member Function Documentation

wxTreeItemId wxTreeEvent::GetItem ( ) const

Returns the item (valid for all events).

int wxTreeEvent::GetKeyCode ( ) const

Returns the key code if the event is a key event.

Use GetKeyEvent() to get the values of the modifier keys for this event (i.e. Shift or Ctrl).

const wxKeyEvent& wxTreeEvent::GetKeyEvent ( ) const

Returns the key event for `EVT_TREE_KEY_DOWN` events.

const wxString& wxTreeEvent::GetLabel ( ) const

Returns the label if the event is a begin or end edit label event.

wxTreeItemId wxTreeEvent::GetOldItem ( ) const

Returns the old item index (valid for `EVT_TREE_ITEM_CHANGING` and `EVT_TREE_ITEM_CHANGED` events).

wxPoint wxTreeEvent::GetPoint ( ) const

Returns the position of the mouse pointer if the event is a drag or menu-context event.

In both cases the position is in client coordinates - i.e. relative to the wxTreeCtrl window (so that you can pass it directly to e.g. wxWindow::PopupMenu()).

bool wxTreeEvent::IsEditCancelled ( ) const

Returns true if the label edit was cancelled.

This should be called from within an `EVT_TREE_END_LABEL_EDIT` handler.

void wxTreeEvent::SetToolTip (const wxString & *tooltip*)

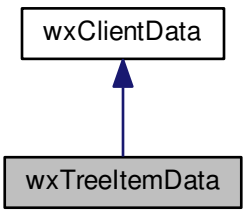
Set the tooltip for the item (valid for `EVT_TREE_ITEM_GETTOOLTIP` events).

Windows only.

21.808 wxTreeItemData Class Reference

`#include <wx/treebase.h>`

Inheritance diagram for wxTreeItemData:



21.808.1 Detailed Description

wxTreeItemData is some (arbitrary) user class associated with some item.

The main advantage of having this class is that wxTreeItemData objects are destroyed automatically by the tree and, as this class has virtual destructor, it means that the memory and any other resources associated with a tree item will be automatically freed when it is deleted. Note that we don't use wxObject as the base class for wxTreeItemData because the size of this class is critical: in many applications, each tree leaf will have wxTreeItemData associated with it and the number of leaves may be quite big.

Also please note that because the objects of this class are deleted by the tree using the operator `delete`, they must always be allocated on the heap using `new`.

Library: wxCore

Category: Containers

See also

wxTreeCtrl

Public Member Functions

- wxTreeItemData ()

  Default constructor.
- virtual ~wxTreeItemData ()

  Virtual destructor.
- const wxTreeItemId & GetId () const

  Returns the item associated with this node.
- void SetId (const wxTreeItemId &id)

  Sets the item associated with this node.

21.808.2 Constructor & Destructor Documentation

wxTreeItemData::wxTreeItemData ( )

Default constructor.

wxPerl Note: In wxPerl the constructor accepts a scalar as an optional parameter and stores it as client data; use

- GetData() to retrieve the value.
- SetData(data) to set it.

virtual wxTreeItemData::~wxTreeItemData ( ) [virtual]

Virtual destructor.

21.808.3 Member Function Documentation

const wxTreeItemId& wxTreeItemData::GetId ( ) const

Returns the item associated with this node.

void wxTreeItemData::SetId (const wxTreeItemId & *id*)

Sets the item associated with this node.

Notice that this function is automatically called by wxTreeCtrl methods associating an object of this class with a tree control item such as wxTreeCtrl::AppendItem(), wxTreeCtrl::InsertItem() and wxTreeCtrl::SetItemData() so there is usually no need to call it yourself.

21.809 wxTreeItemId Class Reference

`#include <wx/treebase.h>`

21.809.1 Detailed Description

An opaque reference to a tree item.

Library: wxCore

Category: Data Structures

See also

wxTreeCtrl, wxTreeItemData, wxTreeCtrl Overview

Public Member Functions

- wxTreeItemId ()

  Default constructor.
- bool IsOk () const

  Returns true if this instance is referencing a valid tree item.
- void * GetID () const
- void Unset ()

21.809.2 Constructor & Destructor Documentation

wxTreeItemId::wxTreeItemId ( )

Default constructor.

A wxTreeItemId is not meant to be constructed explicitly by the user; only those returned by the wxTreeCtrl functions should be used.

21.809.3 Member Function Documentation

void* wxTreeItemId::GetID ( ) const

bool wxTreeItemId::IsOk ( ) const

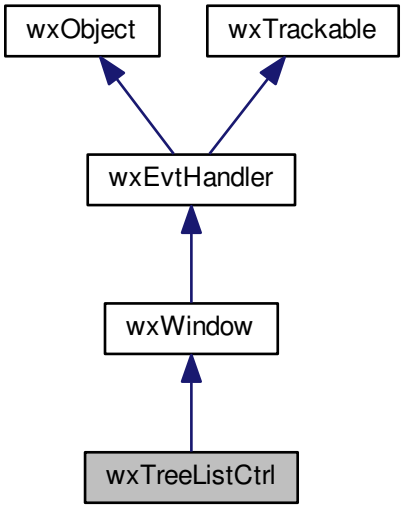
Returns true if this instance is referencing a valid tree item.

void wxTreeItemId::Unset ( )

21.810 wxTreeListCtrl Class Reference

`#include <wx/treelist.h>`

Inheritance diagram for wxTreeListCtrl:



21.810.1 Detailed Description

A control combining wxTreeCtrl and wxListCtrl features.

This is a multi-column tree control optionally supporting images and checkboxes for the items in the first column.

It is currently implemented using wxDataViewCtrl internally but provides a much simpler interface for the common use case it addresses. Thus, one of the design principles for this control is simplicity and intentionally doesn't provide all the features of wxDataViewCtrl. Most importantly, this class stores all its data internally and doesn't require you to define a custom model for it.

Instead, this controls works like wxTreeCtrl or non-virtual wxListCtrl and allows you to simply add items to it using wxTreeListCtrl::AppendItem() and related methods. Typically, you start by setting up the columns (you must have at least one) by calling wxTreeListCtrl::AppendColumn() and then add the items. While only the text of the first column can be specified when adding them, you can use wxTreeListCtrl::SetItemText() to set the text of the other columns.

Unlike wxTreeCtrl or wxListCtrl this control can sort its items on its own. To allow user to sort the control contents by clicking on some column you should use wxCOL_SORTABLE flag when adding that column to the control. When a column with this flag is clicked, the control resorts itself using the values in this column. By default the sort is done using alphabetical order comparison of the items text, which is not always correct (e.g. this doesn't work for the numeric columns). To change this you may use SetItemComparator() method to provide a custom comparator, i.e. simply an object that implements comparison between the two items. The treelist sample shows an example of doing this. And if you need to sort the control programmatically, you can call SetSortColumn() method.

Here are the styles supported by this control. Notice that using wxTL_USER_3STATE implies wxTL_3STATE and wxTL_3STATE in turn implies wxTL_CHECKBOX.

Styles

This class supports the following styles:

- wxTL_SINGLE: Single selection, this is the default.
- wxTL_MULTIPLE: Allow multiple selection, see GetSelections().
- wxTL_CHECKBOX: Show the usual, 2 state, checkboxes for the items in the first column.
- wxTL_3STATE: Show the checkboxes that can possibly be set by the program, but not the user, to a third, undetermined, state, for the items in the first column. Implies wxTL_CHECKBOX.
- wxTL_USER_3STATE: Same as wxTL_3STATE but the user can also set the checkboxes to the undetermined state. Implies wxTL_3STATE.
- wxTL_DEFAULT_STYLE: Style used by the control by default, just wxTL_SINGLE currently.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxTreeListEvent& event)

Event macros:

- EVT_TREELIST_SELECTION_CHANGED(id, func): Process `wxEVT_TREELIST_SELECTION_CHANGED` event and notifies about the selection change in the control. In the single selection case the item indicated by the event has been selected and previously selected item, if any, was deselected. In multiple selection case, the selection of this item has just changed (it may have been either selected or deselected) but notice that the selection of other items could have changed as well, use wxTreeListCtrl::GetSelections() to retrieve the new selection if necessary.
- EVT_TREELIST_ITEM_EXPANDING(id, func): Process `wxEVT_TREELIST_ITEM_EXPANDING` event notifying about the given branch being expanded. This event is sent before the expansion occurs and can be vetoed to prevent it from happening.
- EVT_TREELIST_ITEM_EXPANDED(id, func): Process `wxEVT_TREELIST_ITEM_EXPANDED` event notifying about the expansion of the given branch. This event is sent after the expansion occurs and can't be vetoed.
- EVT_TREELIST_ITEM_CHECKED(id, func): Process `wxEVT_TREELIST_ITEM_CHECKED` event notifying about the user checking or unchecking the item. You can use wxTreeListCtrl::GetCheckedState() to retrieve the new item state and wxTreeListEvent::GetOldCheckedState() to get the previous one.
- EVT_TREELIST_ITEM_ACTIVATED(id, func): Process `wxEVT_TREELIST_ITEM_ACTIVATED` event notifying about the user double clicking the item or activating it from keyboard.
- EVT_TREELIST_ITEM_CONTEXT_MENU(id, func): Process `wxEVT_TREELIST_ITEM_CONTEXT_MENU` event indicating that the popup menu for the given item should be displayed.
- EVT_TREELIST_COLUMN_SORTED(id, func): Process `wxEVT_TREELIST_COLUMN_SORTED` event indicating that the control contents has just been resorted using the specified column. The event doesn't carry the sort direction, use GetSortColumn() method if you need to know it.

Library: wxAdvanced

Category: Controls

Since

2.9.3

See also

wxTreeCtrl, wxDataViewCtrl

- static const int NO_IMAGE = -1

  Image list methods.
- void AssignImageList (wxImageList *imageList)

  Sets the image list and gives its ownership to the control.
- void SetImageList (wxImageList *imageList)

  Sets the image list.

Public Member Functions

- wxTreeListCtrl ()

  Default constructor, call Create() later.
- wxTreeListCtrl (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTL_DEFAULT_STYLE, const wxString &name=wxTreeListCtrlNameStr)

  Full constructing, creating the object and its window.
- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=wxTL_DEFAULT_STYLE, const wxString &name=wxTreeListCtrlNameStr)

  Create the control window.

- int AppendColumn (const wxString &title, int width=wxCOL_WIDTH_AUTOSIZE, wxAlignment align=wxALIGN_LEFT, int flags=wxCOL_RESIZABLE)

  Column methods.
- unsigned GetColumnCount () const

  Return the total number of columns.
- bool DeleteColumn (unsigned col)

  Delete the column with the given index.
- void ClearColumns ()

  Delete all columns.
- void SetColumnWidth (unsigned col, int width)

  Change the width of the given column.
- int GetColumnWidth (unsigned col) const

  Get the current width of the given column in pixels.
- int WidthFor (const wxString &text) const

  Get the width appropriate for showing the given text.

- wxTreeListItem AppendItem (wxTreeListItem parent, const wxString &text, int imageClosed=NO_IMAGE, int imageOpened=NO_IMAGE, wxClientData *data=NULL)

  Adding and removing items.
- wxTreeListItem InsertItem (wxTreeListItem parent, wxTreeListItem previous, const wxString &text, int imageClosed=NO_IMAGE, int imageOpened=NO_IMAGE, wxClientData *data=NULL)

  Insert a new item into the tree.
- wxTreeListItem PrependItem (wxTreeListItem parent, const wxString &text, int imageClosed=NO_IMAGE, int imageOpened=NO_IMAGE, wxClientData *data=NULL)

  Same as InsertItem() with wxTLI_FIRST.
- void DeleteItem (wxTreeListItem item)

Delete the specified item.

- void DeleteAllItems ()

  Delete all tree items.

- wxTreeListItem GetRootItem () const

  Methods for the tree navigation.
- wxTreeListItem GetItemParent (wxTreeListItem item) const

  Return the parent of the given item.
- wxTreeListItem GetFirstChild (wxTreeListItem item) const

  Return the first child of the given item.
- wxTreeListItem GetNextSibling (wxTreeListItem item) const

  Return the next sibling of the given item.
- wxTreeListItem GetFirstItem () const

  Return the first item in the tree.
- wxTreeListItem GetNextItem (wxTreeListItem item) const

  Get item after the given one in the depth-first tree-traversal order.

- const wxString & GetItemText (wxTreeListItem item, unsigned col=0) const

  Items attributes.
- void SetItemText (wxTreeListItem item, unsigned col, const wxString &text)

  Set the text of the specified column of the given item.
- void SetItemText (wxTreeListItem item, const wxString &text)

  Set the text of the first column of the given item.
- void SetItemImage (wxTreeListItem item, int closed, int opened=NO_IMAGE)

  Set the images for the given item.
- wxClientData * GetItemData (wxTreeListItem item) const

  Get the data associated with the given item.
- void SetItemData (wxTreeListItem item, wxClientData *data)

  Set the data associated with the given item.

- void Expand (wxTreeListItem item)

  Expanding and collapsing tree branches.
- void Collapse (wxTreeListItem item)

  Collapse the given tree branch.
- bool IsExpanded (wxTreeListItem item) const

  Return whether the given item is expanded.

- wxTreeListItem GetSelection () const

  Selection methods.
- unsigned GetSelections (wxTreeListItems &selections) const

  Fill in the provided array with all the selected items.
- void Select (wxTreeListItem item)

  Select the given item.
- void Unselect (wxTreeListItem item)

  Deselect the given item.
- bool IsSelected (wxTreeListItem item) const

  Return true if the item is selected.
- void SelectAll ()

  Select all the control items.
- void UnselectAll ()

  Deselect all the control items.

- void CheckItem (wxTreeListItem item, wxCheckBoxState state=wxCHK_CHECKED)

  Checkbox handling.
- void CheckItemRecursively (wxTreeListItem item, wxCheckBoxState state=wxCHK_CHECKED)

  Change the checked state of the given item and all its children.
- void UncheckItem (wxTreeListItem item)

  Uncheck the given item.
- void UpdateItemParentStateRecursively (wxTreeListItem item)

  Update the state of the parent item to reflect the checked state of its children.
- wxCheckBoxState GetCheckedState (wxTreeListItem item) const

  Return the checked state of the item.
- bool AreAllChildrenInState (wxTreeListItem item, wxCheckBoxState state) const

  Return true if all children of the given item are in the specified state.

- void SetSortColumn (unsigned col, bool ascendingOrder=true)

  Sorting.
- bool GetSortColumn (unsigned *col, bool *ascendingOrder=NULL)

  Return the column currently used for sorting, if any.
- void SetItemComparator (wxTreeListItemComparator *comparator)

  Set the object to use for comparing the items.

- wxWindow * GetView () const

  View window.
- wxDataViewCtrl * GetDataView () const

  Return the view part of this control as wxDataViewCtrl.

Additional Inherited Members

21.810.2 Constructor & Destructor Documentation

wxTreeListCtrl::wxTreeListCtrl ( )

Default constructor, call Create() later.

This constructor is used during two-part construction process when it is impossible or undesirable to create the window when constructing the object.

wxTreeListCtrl::wxTreeListCtrl (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTL_DEFAULT_STYLE, const wxString & *name* = `wxTreeListCtrlNameStr`)

Full constructing, creating the object and its window.

See Create() for the parameters description.

21.810.3 Member Function Documentation

int wxTreeListCtrl::AppendColumn (const wxString & *title*, int *width* = wxCOL_WIDTH_AUTOSIZE, wxAlignment *align* = wxALIGN_LEFT, int *flags* = wxCOL_RESIZABLE)

Column methods.

Add a column with the given title and attributes.

Parameters

title	The column label.
width	The width of the column in pixels or the special wxCOL_WIDTH_AUTOSIZE value indicating that the column should adjust to its contents. Notice that the first column is special and will be always resized to fill all the space not taken by the other columns, i.e. the width specified here is ignored for it.
align	Alignment of both the column header and its items.
flags	Column flags, currently can include wxCOL_RESIZABLE to allow the user to resize the column and wxCOL_SORTABLE to allow the user to resort the control contents by clicking on this column.

Returns

Index of the new column or -1 on failure.

wxTreeListItem wxTreeListCtrl::AppendItem (wxTreeListItem *parent,* const wxString & *text,* int *imageClosed* = NO_IMAGE, int *imageOpened* = NO_IMAGE, wxClientData * *data* = NULL)

Adding and removing items.

When adding items, the parent and text of the first column of the new item must always be specified, the rest is optional.

Each item can have two images: one used for closed state and another for opened one. Only the first one is ever used for the items that don't have children. And both are not set by default.

It is also possible to associate arbitrary client data pointer with the new item. It will be deleted by the control when the item is deleted (either by an explicit DeleteItem() call or because the entire control is destroyed).Same as InsertItem() with wxTLI_LAST.

bool wxTreeListCtrl::AreAllChildrenInState (wxTreeListItem *item,* wxCheckBoxState *state*) const

Return true if all children of the given item are in the specified state.

This is especially useful for the controls with wxTL_3STATE style to allow to decide whether the parent effective state should be the same *state*, if all its children are in it, or wxCHK_UNDETERMINED.

See also

UpdateItemParentStateRecursively()

void wxTreeListCtrl::AssignImageList (wxImageList * *imageList*)

Sets the image list and gives its ownership to the control.

The image list assigned with this method will be automatically deleted by wxTreeCtrl as appropriate (i.e. it takes ownership of the list).

See also

SetImageList().

void wxTreeListCtrl::CheckItem (wxTreeListItem *item,* wxCheckBoxState *state* = wxCHK_CHECKED)

Checkbox handling.

Methods in this section can only be used with the controls created with wxTL_CHECKBOX style. Change the item checked state.

Parameters

item	Valid non-root tree item.
state	One of wxCHK_CHECKED, wxCHK_UNCHECKED or, for the controls with wxTL_3STATE or wxTL_USER_3STATE styles, wxCHK_UNDETERMINED.

void wxTreeListCtrl::CheckItemRecursively (wxTreeListItem *item*, wxCheckBoxState *state* = wxCHK_CHECKED)

Change the checked state of the given item and all its children.

This is the same as CheckItem() but checks or unchecks not only this item itself but all its children recursively as well.

void wxTreeListCtrl::ClearColumns ( )

Delete all columns.

See also

DeleteAllItems()

void wxTreeListCtrl::Collapse (wxTreeListItem *item*)

Collapse the given tree branch.

bool wxTreeListCtrl::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = wxTL_DEFAULT_STYLE, const wxString & *name* = `wxTreeListCtrlNameStr`)

Create the control window.

Can be only called for the objects created using the default constructor and exactly once.

Parameters

parent	The parent window, must be non-NULL.
id	The window identifier, may be wxID_ANY.
pos	The initial window position, usually unused.
size	The initial window size, usually unused.
style	The window style, see their description in the class documentation.
name	The name of the window.

void wxTreeListCtrl::DeleteAllItems ( )

Delete all tree items.

bool wxTreeListCtrl::DeleteColumn (unsigned *col*)

Delete the column with the given index.

Parameters

col	Column index in 0 to GetColumnCount() (exclusive) range.

Returns

True if the column was deleted, false if index is invalid or deleting the column failed for some other reason.

void wxTreeListCtrl::DeleteItem (wxTreeListItem *item*)

Delete the specified item.

void wxTreeListCtrl::Expand (wxTreeListItem *item*)

Expanding and collapsing tree branches.

Notice that calling neither Expand() nor Collapse() method generates any events. Expand the given tree branch.

wxCheckBoxState wxTreeListCtrl::GetCheckedState (wxTreeListItem *item*) const

Return the checked state of the item.

The return value can be wxCHK_CHECKED, wxCHK_UNCHECKED or wxCHK_UNDETERMINED.

unsigned wxTreeListCtrl::GetColumnCount ( ) const

Return the total number of columns.

int wxTreeListCtrl::GetColumnWidth (unsigned *col*) const

Get the current width of the given column in pixels.

wxDataViewCtrl* wxTreeListCtrl::GetDataView ( ) const

Return the view part of this control as wxDataViewCtrl.

This method may return NULL in the future, non wxDataViewCtrl-based, versions of this class, use GetView() unless you really need to use wxDataViewCtrl methods on the returned object.

wxTreeListItem wxTreeListCtrl::GetFirstChild (wxTreeListItem *item*) const

Return the first child of the given item.

Item may be the root item.

Return value may be invalid if the item doesn't have any children.

wxTreeListItem wxTreeListCtrl::GetFirstItem ( ) const

Return the first item in the tree.

This is the first child of the root item.

See also

GetNextItem()

wxClientData* wxTreeListCtrl::GetItemData (wxTreeListItem *item*) const

Get the data associated with the given item.

The returned pointer may be NULL.

It must not be deleted by the caller as this will be done by the control itself.

wxTreeListItem wxTreeListCtrl::GetItemParent (wxTreeListItem *item*) const

Return the parent of the given item.

All the tree items visible in the tree have valid parent items, only the never shown root item has no parent.

const wxString& wxTreeListCtrl::GetItemText (wxTreeListItem *item*, unsigned *col* = 0) const

Items attributes.

Return the text of the given item.

By default, returns the text of the first column but any other one can be specified using *col* argument.

wxTreeListItem wxTreeListCtrl::GetNextItem (wxTreeListItem *item*) const

Get item after the given one in the depth-first tree-traversal order.

Calling this function starting with the result of GetFirstItem() allows iterating over all items in the tree.

The iteration stops when this function returns an invalid item, i.e.

```
for ( wxTreeListItem item = tree->GetFirstItem();
      item.IsOk();
      item = tree->GetNextItem(item) )
{
    ... Do something with every tree item ...
}
```

wxTreeListItem wxTreeListCtrl::GetNextSibling (wxTreeListItem *item*) const

Return the next sibling of the given item.

Return value may be invalid if there are no more siblings.

wxTreeListItem wxTreeListCtrl::GetRootItem ( ) const

Methods for the tree navigation.

The tree has an invisible root item which is the hidden parent of all top-level items in the tree. Starting from it it is possible to iterate over all tree items using GetNextItem().

It is also possible to iterate over just the children of the given item by using GetFirstChild() to get the first of them and then calling GetNextSibling() to retrieve all the others.Return the (never shown) root item.

wxTreeListItem wxTreeListCtrl::GetSelection ( ) const

Selection methods.

The behaviour of the control is different in single selection mode (the default) and multi-selection mode (if wxTL↩_MULTIPLE was specified when creating it). Not all methods can be used in both modes and some of those that can don't behave in the same way in two cases. Return the currently selected item.

This method can't be used with multi-selection controls, use GetSelections() instead.

The return value may be invalid if no item has been selected yet. Once an item in a single selection control was selected, it will keep a valid selection.

unsigned wxTreeListCtrl::GetSelections (wxTreeListItems & *selections*) const

Fill in the provided array with all the selected items.

This method can be used in both single and multi-selection case.

The previous array contents is destroyed.

Returns the number of selected items.

bool wxTreeListCtrl::GetSortColumn (unsigned * *col*, bool * *ascendingOrder* = NULL)

Return the column currently used for sorting, if any.

If the control is currently unsorted, the function simply returns false and doesn't modify any of its output parameters.

Parameters

col	Receives the index of the column used for sorting if non-NULL.
ascendingOrder	Receives true or false depending on whether the items are sorted in ascending or descending order.

Returns

true if the control is sorted or false if it isn't sorted at all.

wxWindow* wxTreeListCtrl::GetView ( ) const

View window.

This control itself is entirely covered by the "view window" which is currently a wxDataViewCtrl but if you want to avoid relying on this to allow your code to work with later versions which might not be wxDataViewCtrl-based, use GetView() function only and only use GetDataView() if you really need to call wxDataViewCtrl methods on it. Return the view part of this control as a wxWindow.

This method always returns non-NULL pointer once the window was created.

wxTreeListItem wxTreeListCtrl::InsertItem (wxTreeListItem *parent*, wxTreeListItem *previous*, const wxString & *text*, int *imageClosed* = NO_IMAGE, int *imageOpened* = NO_IMAGE, wxClientData * *data* = NULL)

Insert a new item into the tree.

Parameters

parent	The item parent. Must be valid, may be GetRootItem().
previous	The previous item that this one should be inserted immediately after. It must be valid but may be one of the special values wxTLI_FIRST or wxTLI_LAST indicating that the item should be either inserted before the first child of its parent (if any) or after the last one.

text	The item text.
imageClosed	The normal item image, may be NO_IMAGE to not show any image.
imageOpened	The item image shown when it's in the expanded state.
data	Optional client data pointer that can be later retrieved using GetItemData() and will be deleted by the tree when the item itself is deleted.

bool wxTreeListCtrl::IsExpanded (wxTreeListItem *item*) const

Return whether the given item is expanded.

bool wxTreeListCtrl::IsSelected (wxTreeListItem *item*) const

Return true if the item is selected.

This method can be used in both single and multiple selection modes.

wxTreeListItem wxTreeListCtrl::PrependItem (wxTreeListItem *parent,* const wxString & *text,* int *imageClosed* = NO_IMAGE, int *imageOpened* = NO_IMAGE, wxClientData * *data* = NULL)

Same as InsertItem() with wxTLI_FIRST.

void wxTreeListCtrl::Select (wxTreeListItem *item*)

Select the given item.

In single selection mode, deselects any other selected items, in multi-selection case it adds to the selection.

void wxTreeListCtrl::SelectAll ( )

Select all the control items.

Can be only used in multi-selection mode.

void wxTreeListCtrl::SetColumnWidth (unsigned *col,* int *width*)

Change the width of the given column.

Set column width to either the given value in pixels or to the value large enough to fit all of the items if width is wxCOL_WIDTH_AUTOSIZE.

Notice that setting the width of the first column is ignored as this column is always resized to fill the space left by the other columns.

void wxTreeListCtrl::SetImageList (wxImageList * *imageList*)

Sets the image list.

The image list assigned with this method will **not** be deleted by the control itself and you will need to delete it yourself, use AssignImageList() to give the image list ownership to the control.

Parameters

imageList	Image list to use, may be NULL to not show any images any more.

void wxTreeListCtrl::SetItemComparator (wxTreeListItemComparator * *comparator*)

Set the object to use for comparing the items.

This object will be used when the control is being sorted because the user clicked on a sortable column or SetSortColumn() was called.

The provided pointer is stored by the control so the object it points to must have a life-time equal or greater to that of the control itself. In addition, the pointer can be NULL to stop using custom comparator and revert to the default alphabetical comparison.

void wxTreeListCtrl::SetItemData (wxTreeListItem *item*, wxClientData * *data*)

Set the data associated with the given item.

Previous client data, if any, is deleted when this function is called so it may be used to delete the current item data object and reset it by passing NULL as *data* argument.

void wxTreeListCtrl::SetItemImage (wxTreeListItem *item*, int *closed*, int *opened* = NO_IMAGE)

Set the images for the given item.

See InsertItem() for the images parameters descriptions.

void wxTreeListCtrl::SetItemText (wxTreeListItem *item*, unsigned *col*, const wxString & *text*)

Set the text of the specified column of the given item.

void wxTreeListCtrl::SetItemText (wxTreeListItem *item*, const wxString & *text*)

Set the text of the first column of the given item.

void wxTreeListCtrl::SetSortColumn (unsigned *col*, bool *ascendingOrder* = `true`)

Sorting.

If some control columns were added with wxCOL_SORTABLE flag, clicking on them will automatically resort the control using the custom comparator set by SetItemComparator() or by doing alphabetical comparison by default.

In any case, i.e. even if the user can't sort the control by clicking on its header, you may call SetSortColumn() to sort it programmatically and call GetSortColumn() to determine whether it's sorted now and, if so, by which column and in which order. Set the column to use for sorting and the order in which to sort.

Calling this method resorts the control contents using the values of the items in the specified column. Sorting uses custom comparator set with SetItemComparator() or alphabetical comparison of items texts if none was specified.

Notice that currently there is no way to reset sort order.

Parameters

col	A valid column index.

ascendingOrder	Indicates whether the items should be sorted in ascending (A to Z) or descending (Z to A) order.

void wxTreeListCtrl::UncheckItem (wxTreeListItem *item*)

Uncheck the given item.

This is synonymous with CheckItem(wxCHK_UNCHECKED).

void wxTreeListCtrl::Unselect (wxTreeListItem *item*)

Deselect the given item.

This method can be used in multiple selection mode only.

void wxTreeListCtrl::UnselectAll ( )

Deselect all the control items.

Can be only used in multi-selection mode.

void wxTreeListCtrl::UpdateItemParentStateRecursively (wxTreeListItem *item*)

Update the state of the parent item to reflect the checked state of its children.

This method updates the parent of this item recursively: if this item and all its siblings are checked, the parent will become checked as well. If this item and all its siblings are unchecked, the parent will be unchecked. And if the siblings of this item are not all in the same state, the parent will be switched to indeterminate state. And then the same logic will be applied to the parents parent and so on recursively.

This is typically called when the state of the given item has changed from EVT_TREELIST_ITEM_CHECKED() handler in the controls which have wxTL_3STATE flag. Notice that without this flag this function can't work as it would be unable to set the state of a parent with both checked and unchecked items so it's only allowed to call it when this flag is set.

int wxTreeListCtrl::WidthFor (const wxString & *text*) const

Get the width appropriate for showing the given text.

This is typically used as second argument for AppendColumn() or with SetColumnWidth().

21.810.4 Member Data Documentation

const int wxTreeListCtrl::NO_IMAGE = -1 `[static]`

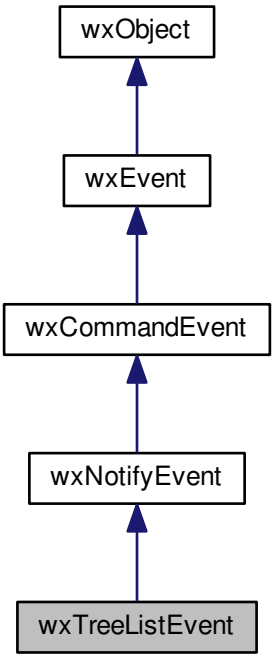
Image list methods.

Like wxTreeCtrl and wxListCtrl this class uses wxImageList so if you intend to use item icons with it, you must construct wxImageList containing them first and then specify the indices of the icons in this image list when adding the items later.A constant indicating that no image should be used for an item.

21.811 wxTreeListEvent Class Reference

`#include <wx/treelist.h>`

Inheritance diagram for wxTreeListEvent:



21.811.1 Detailed Description

Event generated by wxTreeListCtrl.

Since

2.9.3

Public Member Functions

- wxTreeListEvent ()
- wxTreeListItem GetItem () const

  Return the item affected by the event.
- wxCheckBoxState GetOldCheckedState () const

  Return the previous state of the item checkbox.
- unsigned GetColumn () const

  Return the column affected by the event.

Additional Inherited Members

21.811.2 Constructor & Destructor Documentation

wxTreeListEvent::wxTreeListEvent ( )

21.811.3 Member Function Documentation

unsigned wxTreeListEvent::GetColumn ( ) const

Return the column affected by the event.

This is currently only used with `wxEVT_TREELIST_COLUMN_SORTED` event.

wxTreeListItem wxTreeListEvent::GetItem ( ) const

Return the item affected by the event.

This is the item being selected, expanded, checked or activated (depending on the event type).

wxCheckBoxState wxTreeListEvent::GetOldCheckedState ( ) const

Return the previous state of the item checkbox.

This method can be used with `wxEVT_TREELIST_ITEM_CHECKED` events only.

Notice that the new state of the item can be retrieved using wxTreeListCtrl::GetCheckedState().

21.812 wxTreeListItem Class Reference

```
#include <wx/treelist.h>
```

21.812.1 Detailed Description

Unique identifier of an item in wxTreeListCtrl.

This is an opaque class which can't be used by the application in any other way than receiving or passing it to wxTreeListCtrl and checking for validity.

See also

wxTreeListCtrl

Library: wxAdvanced

Category: Controls

Since

2.9.3

Public Member Functions

- wxTreeListItem ()

  Only the default constructor is publicly accessible.
- bool IsOk () const

  Return true if the item is valid.

21.812.2 Constructor & Destructor Documentation

wxTreeListItem::wxTreeListItem ( )

Only the default constructor is publicly accessible.

Default constructing a wxTreeListItem creates an invalid item.

21.812.3 Member Function Documentation

bool wxTreeListItem::IsOk ( ) const

Return true if the item is valid.

21.813 wxTreeListItemComparator Class Reference

`#include <wx/treelist.h>`

21.813.1 Detailed Description

Class defining sort order for the items in wxTreeListCtrl.

See also

wxTreeListCtrl

Library: wxAdvanced

Category: Controls

Since

2.9.3

Public Member Functions

- wxTreeListItemComparator ()

  Default constructor.
- virtual int Compare (wxTreeListCtrl *treelist, unsigned column, wxTreeListItem first, wxTreeListItem second)=0

  Pure virtual function which must be overridden to define sort order.
- virtual ~wxTreeListItemComparator ()

  Trivial but virtual destructor.

21.813.2 Constructor & Destructor Documentation

wxTreeListItemComparator::wxTreeListItemComparator ( )

Default constructor.

Notice that this class is not copyable, comparators are not passed by value.

virtual wxTreeListItemComparator::~wxTreeListItemComparator ( ) [virtual]

Trivial but virtual destructor.

Although this class is not used polymorphically by wxWidgets itself, provide virtual dtor in case it's used like this in the user code.

21.813.3 Member Function Documentation

virtual int wxTreeListItemComparator::Compare (wxTreeListCtrl * *treelist,* unsigned *column,* wxTreeListItem *first,* wxTreeListItem *second*) [pure virtual]

Pure virtual function which must be overridden to define sort order.

The comparison function should return negative, null or positive value depending on whether the first item is less than, equal to or greater than the second one. The items should be compared using their values for the given column.

Parameters

treelist	The control whose contents is being sorted.
column	The column of this control used for sorting.
first	First item to compare.
second	Second item to compare.

Returns

A negative value if the first item is less than (i.e. should appear above) the second one, zero if the two items are equal or a positive value if the first item is greater than (i.e. should appear below) the second one.

21.814 wxUIActionSimulator Class Reference

`#include <wx/uiaction.h>`

21.814.1 Detailed Description

wxUIActionSimulator is a class used to simulate user interface actions such as a mouse click or a key press.

Common usage for this class would be to provide playback and record (aka macro recording) functionality for users, or to drive unit tests by simulating user sessions.

See the wxUIActionSimulator Sample for an example of using this class.

Since

2.9.2

Library: wxCore

Public Member Functions

- wxUIActionSimulator ()

  Default constructor.
- bool MouseMove (long x, long y)

Move the mouse to the specified coordinates.

- bool MouseMove (const wxPoint &point)

  Move the mouse to the specified coordinates.
- bool MouseDown (int button=wxMOUSE_BTN_LEFT)

  Press a mouse button.
- bool MouseUp (int button=wxMOUSE_BTN_LEFT)

  Release a mouse button.
- bool MouseClick (int button=wxMOUSE_BTN_LEFT)

  Click a mouse button.
- bool MouseDblClick (int button=wxMOUSE_BTN_LEFT)

  Double-click a mouse button.
- bool MouseDragDrop (long x1, long y1, long x2, long y2, int button=wxMOUSE_BTN_LEFT)

  Perform a drag and drop operation.
- bool KeyDown (int keycode, int modifiers=wxMOD_NONE)

  Press a key.
- bool KeyUp (int keycode, int modifiers=wxMOD_NONE)

  Release a key.
- bool Char (int keycode, int modifiers=wxMOD_NONE)

  Press and release a key.
- bool Select (const wxString &text)

  Simulate selection of an item with the given text.
- bool Text (const wxString &text)

  Emulate typing in the keys representing the given string.

21.814.2 Constructor & Destructor Documentation

wxUIActionSimulator::wxUIActionSimulator ( )

Default constructor.

21.814.3 Member Function Documentation

bool wxUIActionSimulator::Char (int *keycode*, int *modifiers* = wxMOD_NONE)

Press and release a key.

Parameters

keycode	Key to operate on, as an integer. It is interpreted as a wxKeyCode.
modifiers	A combination of wxKeyModifier flags to be pressed with the given keycode.

bool wxUIActionSimulator::KeyDown (int *keycode*, int *modifiers* = wxMOD_NONE)

Press a key.

If you are using modifiers then it needs to be paired with an identical KeyUp or the modifiers will not be released (MSW and OSX).

Parameters

keycode	Key to operate on, as an integer. It is interpreted as a wxKeyCode.
modifiers	A combination of wxKeyModifier flags to be pressed with the given keycode.

bool wxUIActionSimulator::KeyUp (int *keycode,* int *modifiers* = wxMOD_NONE)

Release a key.

Parameters

keycode	Key to operate on, as an integer. It is interpreted as a wxKeyCode.
modifiers	A combination of wxKeyModifier flags to be pressed with the given keycode.

bool wxUIActionSimulator::MouseClick (int *button* = wxMOUSE_BTN_LEFT)

Click a mouse button.

Parameters

button	Button to press. See wxUIActionSimulator::MouseDown for a list of valid constants.

bool wxUIActionSimulator::MouseDblClick (int *button* = wxMOUSE_BTN_LEFT)

Double-click a mouse button.

Parameters

button	Button to press. See wxUIActionSimulator::MouseDown for a list of valid constants.

bool wxUIActionSimulator::MouseDown (int *button* = wxMOUSE_BTN_LEFT)

Press a mouse button.

Parameters

button	Button to press. Valid constants are `wxMOUSE_BTN_LEFT`, `wxMOUSE_BTN_MIDDLE`, and `wxMOUSE_BTN_RIGHT`.

bool wxUIActionSimulator::MouseDragDrop (long *x1,* long *y1,* long *x2,* long *y2,* int *button* = wxMOUSE_BTN_LEFT)

Perform a drag and drop operation.

Parameters

x1	x start coordinate, in screen coordinates.
y1	y start coordinate, in screen coordinates.
x2	x destination coordinate, in screen coordinates.
y2	y destination coordinate, in screen coordinates.
button	Button to press. See wxUIActionSimulator::MouseDown for a list of valid constants.

bool wxUIActionSimulator::MouseMove (long *x,* long *y*)

Move the mouse to the specified coordinates.

Parameters

x	x coordinate to move to, in screen coordinates.
y	y coordinate to move to, in screen coordinates.

bool wxUIActionSimulator::MouseMove (const wxPoint & *point*)

Move the mouse to the specified coordinates.

Parameters

point	Point to move to, in screen coordinates.

bool wxUIActionSimulator::MouseUp (int *button* = wxMOUSE_BTN_LEFT)

Release a mouse button.

Parameters

button	Button to press. See wxUIActionSimulator::MouseDown for a list of valid constants.

bool wxUIActionSimulator::Select (const wxString & *text*)

Simulate selection of an item with the given text.

This method selects an item in the currently focused wxChoice, wxComboBox, wxListBox and similar controls. It does it by simulating keyboard events, so the behaviour should be the same as if the item was really selected by the user.

Notice that the implementation of this method uses wxYield() and so events can be dispatched from it.

Parameters

text	The text of the item to select.

Returns

true if the item *text* was successfully selected or false if the currently focused window is not one of the controls allowing item selection or if the item with the given text was not found in it.

Since

3.1.0

bool wxUIActionSimulator::Text (const wxString & *text*)

Emulate typing in the keys representing the given string.

Currently only the ASCII letters, digits and characters for the definition of numbers (i.e. characters `a-z A-Z 0-9 + - . , 'space'`) are supported.

Parameters

text	The string to type.

21.815 wxULongLong Class Reference

`#include <wx/longlong.h>`

21.815.1 Detailed Description

This class represents an unsigned 64 bit long number.

Since wxULongLong has exactly the same API as wxLongLong, please refer to wxLongLong documentation (this page exists only as redirection).

Library: wxBase

Category: Data Structures

21.816 wxUniChar Class Reference

`#include <wx/unichar.h>`

21.816.1 Detailed Description

This class represents a single Unicode character.

It can be converted to and from `char` or `wchar_t` and implements commonly used character operations.

Library: wxBase

Category: Data Structures

Public Types

- typedef wxUint32 value_type

  A type capable of holding any Unicode code point.

Public Member Functions

- wxUniChar ()

  Default ctor.
- wxUniChar (int c)
- wxUniChar (unsigned int c)
- wxUniChar (long int c)
- wxUniChar (unsigned long int c)
- wxUniChar (short int c)

- wxUniChar (unsigned short int c)
- wxUniChar (wxLongLong_t c)
- wxUniChar (wxULongLong_t c)
- wxUniChar (const wxUniCharRef &c)
- value_type GetValue () const

  Returns Unicode code point value of the character.
- bool IsAscii () const

  Returns true if the character is an ASCII character (i.e. if its value is less than 128).
- bool GetAsChar (char *c) const

  Returns true if the character is representable as a single byte in the current locale encoding.

- wxUniChar (char c)

  Create a character from the 8-bit character value c using the current locale encoding.
- wxUniChar (unsigned char c)

  Create a character from the 8-bit character value c using the current locale encoding.

- operator char () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator unsigned char () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator wchar_t () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator int () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator unsigned int () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator long int () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator unsigned long int () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator short int () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator unsigned short int () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator wxLongLong_t () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.
- operator wxULongLong_t () const

  Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

- wxUniChar & operator= (const wxUniChar &c)

  Assignment operators.
- wxUniChar & operator= (const wxUniCharRef &c)

Assignment operators.

- wxUniChar & operator= (char c)

  Assignment operators.

- wxUniChar & operator= (unsigned char c)

  Assignment operators.

- wxUniChar & operator= (wchar_t c)

  Assignment operators.

- wxUniChar & operator= (int c)

  Assignment operators.

- wxUniChar & operator= (unsigned int c)

  Assignment operators.

- wxUniChar & operator= (long int c)

  Assignment operators.

- wxUniChar & operator= (unsigned long int c)

  Assignment operators.

- wxUniChar & operator= (short int c)

  Assignment operators.

- wxUniChar & operator= (unsigned short int c)

  Assignment operators.

- wxUniChar & operator= (wxLongLong_t c)

  Assignment operators.

- wxUniChar & operator= (wxULongLong_t c)

  Assignment operators.

21.816.2 Member Typedef Documentation

typedef wxUint32 wxUniChar::value_type

A type capable of holding any Unicode code point.

We do not use wchar_t as it cannot do the job on Win32, where wchar_t is a 16-bit type (wchar_t* is encoded using UTF-16 on Win32).

21.816.3 Constructor & Destructor Documentation

wxUniChar::wxUniChar ( )

Default ctor.

wxUniChar::wxUniChar (char *c*)

Create a character from the 8-bit character value *c* using the current locale encoding.

wxUniChar::wxUniChar (unsigned char *c*)

Create a character from the 8-bit character value *c* using the current locale encoding.

wxUniChar::wxUniChar (int *c*)

wxUniChar::wxUniChar (unsigned int *c*)

wxUniChar::wxUniChar (long int *c*)

wxUniChar::wxUniChar (unsigned long int *c*)

wxUniChar::wxUniChar (short int *c*)

wxUniChar::wxUniChar (unsigned short int *c*)

wxUniChar::wxUniChar (wxLongLong_t *c*)

wxUniChar::wxUniChar (wxULongLong_t *c*)

wxUniChar::wxUniChar (const wxUniCharRef & *c*)

21.816.4 Member Function Documentation

bool wxUniChar::GetAsChar (char * *c*) const

Returns true if the character is representable as a single byte in the current locale encoding.

This function only returns true if the character can be converted in exactly one byte, e.g. it only returns true for 7 bit ASCII characters when the encoding used is UTF-8.

It is mostly useful to test if the character can be passed to functions taking a char and is used by wxWidgets itself for this purpose.

Parameters

c	An output pointer to the value of this Unicode character as a `char`. Must be non-NULL.

Returns

true if the object is an 8 bit char and *c* was filled with its value as char or false otherwise (*c* won't be modified then).

See also

IsAscii()

Since

2.9.1

value_type wxUniChar::GetValue ( ) const

Returns Unicode code point value of the character.

bool wxUniChar::IsAscii ( ) const

Returns true if the character is an ASCII character (i.e. if its value is less than 128).

wxUniChar::operator char ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator int ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator long int ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator short int ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator unsigned char ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator unsigned int ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator unsigned long int ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator unsigned short int ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator wchar_t ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator wxLongLong_t ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar::operator wxULongLong_t ( ) const

Conversions to char and wchar_t types: all of those are needed to be able to pass wxUniChars to various standard narrow and wide character functions.

wxUniChar& wxUniChar::operator= (const wxUniChar & *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (const wxUniCharRef & *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (char *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (unsigned char *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (wchar_t *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (int *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (unsigned int *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (long int *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (unsigned long int *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (short int *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (unsigned short int *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (wxLongLong_t *c*)

Assignment operators.

wxUniChar& wxUniChar::operator= (wxULongLong_t *c*)

Assignment operators.

21.817 wxUniCharRef Class Reference

`#include <wx/unichar.h>`

21.817.1 Detailed Description

Writeable reference to a character in wxString.

This class can be used in the same way wxChar is used, except that changing its value updates the underlying string object.

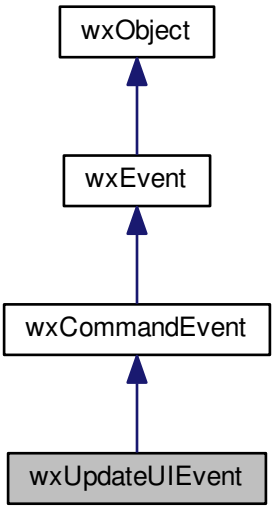
Library: wxBase

Category: Data Structures

21.818 wxUpdateUIEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxUpdateUIEvent:



21.818.1 Detailed Description

This class is used for pseudo-events which are called by wxWidgets to give an application the chance to update various user interface elements.

Without update UI events, an application has to work hard to check/uncheck, enable/disable, show/hide, and set the text for elements such as menu items and toolbar buttons. The code for doing this has to be mixed up with the code that is invoked when an action is invoked for a menu item or button.

With update UI events, you define an event handler to look at the state of the application and change UI elements accordingly. wxWidgets will call your member functions in idle time, so you don't have to worry where to call this code.

In addition to being a clearer and more declarative method, it also means you don't have to worry whether you're updating a toolbar or menubar identifier. The same handler can update a menu item and toolbar button, if the identifier is the same. Instead of directly manipulating the menu or button, you call functions in the event object, such as wxUpdateUIEvent::Check. wxWidgets will determine whether such a call has been made, and which UI element to update.

These events will work for popup menus as well as menubars. Just before a menu is popped up, wxMenu::UpdateUI is called to process any UI events for the window that owns the menu.

If you find that the overhead of UI update processing is affecting your application, you can do one or both of the following:

- Call wxUpdateUIEvent::SetMode with a value of wxUPDATE_UI_PROCESS_SPECIFIED, and set the extra style wxWS_EX_PROCESS_UI_UPDATES for every window that should receive update events. No other windows will receive update events.
- Call wxUpdateUIEvent::SetUpdateInterval with a millisecond value to set the delay between updates. You may need to call wxWindow::UpdateWindowUI at critical points, for example when a dialog is about to be shown, in case the user sees a slight delay before windows are updated.

Note that although events are sent in idle time, defining a wxIdleEvent handler for a window does not affect this because the events are sent from wxWindow::OnInternalIdle which is always called in idle time.

wxWidgets tries to optimize update events on some platforms. On Windows and GTK+, events for menubar items are only sent when the menu is about to be shown, and not in idle time.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxUpdateUIEvent& event)

Event macros:

- EVT_UPDATE_UI(id, func): Process a `wxEVT_UPDATE_UI` event for the command with the given id.
- EVT_UPDATE_UI_RANGE(id1, id2, func): Process a `wxEVT_UPDATE_UI` event for any command with id included in the given range.

Library: wxCore

Category: Events

See also

Events and Event Handling

Public Member Functions

- wxUpdateUIEvent (wxWindowID commandId=0)

  Constructor.
- void Check (bool check)

  Check or uncheck the UI element.
- void Enable (bool enable)

  Enable or disable the UI element.
- bool GetChecked () const

  Returns true if the UI element should be checked.
- bool GetEnabled () const

  Returns true if the UI element should be enabled.
- bool GetSetChecked () const

  Returns true if the application has called Check().
- bool GetSetEnabled () const

  Returns true if the application has called Enable().
- bool GetSetShown () const

  Returns true if the application has called Show().
- bool GetSetText () const

  Returns true if the application has called SetText().
- bool GetShown () const

  Returns true if the UI element should be shown.
- wxString GetText () const

  Returns the text that should be set for the UI element.
- void SetText (const wxString &text)

  Sets the text for this UI element.
- void Show (bool show)

  Show or hide the UI element.

Static Public Member Functions

- static bool CanUpdate (wxWindow *window)

  Returns true if it is appropriate to update (send UI update events to) this window.
- static wxUpdateUIMode GetMode ()

  Static function returning a value specifying how wxWidgets will send update events: to all windows, or only to those which specify that they will process the events.
- static long GetUpdateInterval ()

  Returns the current interval between updates in milliseconds.
- static void ResetUpdateTime ()

  Used internally to reset the last-updated time to the current time.
- static void SetMode (wxUpdateUIMode mode)

  Specify how wxWidgets will send update events: to all windows, or only to those which specify that they will process the events.
- static void SetUpdateInterval (long updateInterval)

  Sets the interval between updates in milliseconds.

Additional Inherited Members

21.818.2 Constructor & Destructor Documentation

wxUpdateUIEvent::wxUpdateUIEvent (wxWindowID *commandId* = 0)

Constructor.

21.818.3 Member Function Documentation

static bool wxUpdateUIEvent::CanUpdate (wxWindow * *window*) [static]

Returns true if it is appropriate to update (send UI update events to) this window.

This function looks at the mode used (see wxUpdateUIEvent::SetMode), the wxWS_EX_PROCESS_UI_UPDATES flag in *window*, the time update events were last sent in idle time, and the update interval, to determine whether events should be sent to this window now. By default this will always return true because the update mode is initially wxUPDATE_UI_PROCESS_ALL and the interval is set to 0; so update events will be sent as often as possible. You can reduce the frequency that events are sent by changing the mode and/or setting an update interval.

See also

ResetUpdateTime(), SetUpdateInterval(), SetMode()

void wxUpdateUIEvent::Check (bool *check*)

Check or uncheck the UI element.

void wxUpdateUIEvent::Enable (bool *enable*)

Enable or disable the UI element.

bool wxUpdateUIEvent::GetChecked ( ) const

Returns true if the UI element should be checked.

bool wxUpdateUIEvent::GetEnabled ( ) const

Returns true if the UI element should be enabled.

static wxUpdateUIMode wxUpdateUIEvent::GetMode ( ) [static]

Static function returning a value specifying how wxWidgets will send update events: to all windows, or only to those which specify that they will process the events.

See also

SetMode()

bool wxUpdateUIEvent::GetSetChecked ( ) const

Returns true if the application has called Check().

For wxWidgets internal use only.

bool wxUpdateUIEvent::GetSetEnabled ( ) const

Returns true if the application has called Enable().

For wxWidgets internal use only.

bool wxUpdateUIEvent::GetSetShown ( ) const

Returns true if the application has called Show().

For wxWidgets internal use only.

bool wxUpdateUIEvent::GetSetText ( ) const

Returns true if the application has called SetText().

For wxWidgets internal use only.

bool wxUpdateUIEvent::GetShown ( ) const

Returns true if the UI element should be shown.

wxString wxUpdateUIEvent::GetText ( ) const

Returns the text that should be set for the UI element.

static long wxUpdateUIEvent::GetUpdateInterval ( ) [static]

Returns the current interval between updates in milliseconds.

The value -1 disables updates, 0 updates as frequently as possible.

See also

SetUpdateInterval().

static void wxUpdateUIEvent::ResetUpdateTime ( ) [static]

Used internally to reset the last-updated time to the current time.

It is assumed that update events are normally sent in idle time, so this is called at the end of idle processing.

See also

CanUpdate(), SetUpdateInterval(), SetMode()

static void wxUpdateUIEvent::SetMode (wxUpdateUIMode *mode*) [static]

Specify how wxWidgets will send update events: to all windows, or only to those which specify that they will process the events.

Parameters

mode	this parameter may be one of the wxUpdateUIMode enumeration values. The default mode is wxUPDATE_UI_PROCESS_ALL.

void wxUpdateUIEvent::SetText (const wxString & *text*)

Sets the text for this UI element.

static void wxUpdateUIEvent::SetUpdateInterval (long *updateInterval*) `[static]`

Sets the interval between updates in milliseconds.

Set to -1 to disable updates, or to 0 to update as frequently as possible. The default is 0.

Use this to reduce the overhead of UI update events if your application has a lot of windows. If you set the value to -1 or greater than 0, you may also need to call wxWindow::UpdateWindowUI at appropriate points in your application, such as when a dialog is about to be shown.

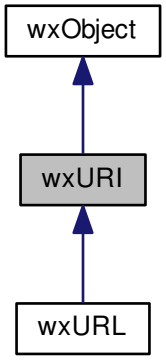
void wxUpdateUIEvent::Show (bool *show*)

Show or hide the UI element.

21.819 wxURI Class Reference

`#include <wx/uri.h>`

Inheritance diagram for wxURI:



21.819.1 Detailed Description

wxURI is used to extract information from a URI (Uniform Resource Identifier).

For information about URIs, see RFC 3986 (http://www.ietf.org/rfc/rfc3986.txt).

In short, a URL is a URI. In other words, URL is a subset of a URI - all acceptable URLs are also acceptable URIs.

wxURI automatically escapes invalid characters in a string, so there is no chance of wxURI "failing" on construction/creation.

wxURI supports copy construction and standard assignment operators. wxURI can also be inherited from to provide further functionality.

To obtain individual components you can use one of the GetXXX() methods. However, you should check HasXXX() before calling a get method, which determines whether or not the component referred to by the method is defined according to RFC 2396. Consider an undefined component equivalent to a NULL C string.

Example:

```
// protocol will hold the http protocol (i.e. "http")
wxString protocol;
wxURI myuri("http://mysite.com");
if( myuri.HasScheme() )
    protocol = myuri.GetScheme();
```

Note

On URIs with a "file" scheme wxURI does not parse the userinfo, server, or port portion. This is to keep compatibility with wxFileSystem, the old wxURL, and older url specifications.

Library: wxBase

Category: Networking

See also

wxURL

Public Member Functions

- wxURI ()

  Creates an empty URI.
- wxURI (const wxString &uri)

  Constructor for quick creation.
- wxURI (const wxURI &uri)

  Copies this URI from another URI.
- wxString BuildURI () const

  Builds the URI from its individual components and adds proper separators.
- wxString BuildUnescapedURI () const

  Builds the URI from its individual components, adds proper separators, and returns escape sequences to normal characters.
- bool Create (const wxString &uri)

  Creates this URI from the uri string.
- const wxString & GetFragment () const

  Obtains the fragment of this URI.
- wxURIHostType GetHostType () const

  Obtains the host type of this URI, which is one of wxURIHostType.
- wxString GetPassword () const

  Returns the password part of the userinfo component of this URI.
- const wxString & GetPath () const

  Returns the (normalized) path of the URI.
- const wxString & GetPort () const

  Returns a string representation of the URI's port.
- const wxString & GetQuery () const

  Returns the Query component of the URI.
- const wxString & GetScheme () const

  Returns the Scheme component of the URI.
- const wxString & GetServer () const

  Returns the Server component of the URI.
- wxString GetUser () const

  Returns the username part of the userinfo component of this URI.

- const wxString & GetUserInfo () const

  Returns the UserInfo component of the URI.
- bool HasFragment () const

  Returns true if the Fragment component of the URI exists.
- bool HasPath () const

  Returns true if the Path component of the URI exists.
- bool HasPort () const

  Returns true if the Port component of the URI exists.
- bool HasQuery () const

  Returns true if the Query component of the URI exists.
- bool HasScheme () const

  Returns true if the Scheme component of the URI exists.
- bool HasServer () const

  Returns true if the Server component of the URI exists.
- bool HasUserInfo () const

  Returns true if the User component of the URI exists.
- bool IsReference () const

  Returns true if a valid [absolute] URI, otherwise this URI is a URI reference and not a full URI, and this function returns false.
- void Resolve (const wxURI &base, int flags=wxURI_STRICT)

  Inherits this URI from a base URI - components that do not exist in this URI are copied from the base, and if this URI's path is not an absolute path (prefixed by a '/'), then this URI's path is merged with the base's path.
- bool operator== (const wxURI &uricomp) const

  Compares this URI to another URI, and returns true if this URI equals uricomp, otherwise it returns false.

Static Public Member Functions

- static wxString Unescape (const wxString &uri)

  Translates all escape sequences (normal characters and returns the result.

Additional Inherited Members

21.819.2 Constructor & Destructor Documentation

wxURI::wxURI ( )

Creates an empty URI.

wxURI::wxURI (const wxString & *uri*)

Constructor for quick creation.

Parameters

uri	URI (Uniform Resource Identifier) to initialize with.

wxURI::wxURI (const wxURI & *uri*)

Copies this URI from another URI.

Parameters

uri	URI (Uniform Resource Identifier) to initialize with.

21.819.3 Member Function Documentation

wxString wxURI::BuildUnescapedURI ( ) const

Builds the URI from its individual components, adds proper separators, and returns escape sequences to normal characters.

Note

It is preferred to call this over Unescape(BuildURI()) since BuildUnescapedURI() performs some optimizations over the plain method.

wxString wxURI::BuildURI ( ) const

Builds the URI from its individual components and adds proper separators.

If the URI is not a reference or is not resolved, the URI that is returned is the same one passed to the constructor or Create().

bool wxURI::Create (const wxString & *uri*)

Creates this URI from the *uri* string.

Returns true if this instance was correctly initialized.

Parameters

uri	String to initialize from.

const wxString& wxURI::GetFragment ( ) const

Obtains the fragment of this URI.

The fragment of a URI is the last value of the URI, and is the value after a "#" character after the path of the URI.

`"http://mysite.com/mypath#<fragment>"`

wxURIHostType wxURI::GetHostType ( ) const

Obtains the host type of this URI, which is one of wxURIHostType.

wxString wxURI::GetPassword ( ) const

Returns the password part of the userinfo component of this URI.

Note that this is explicitly depreciated by RFC 1396 and should generally be avoided if possible.

`"http://<user>:<password>@mysite.com/mypath"`

const wxString& wxURI::GetPath ( ) const

Returns the (normalized) path of the URI.

The path component of a URI comes directly after the scheme component if followed by zero or one slashes ('/'), or after the server/port component.

Absolute paths include the leading '/' character.

"http://mysite.com<path>"

const wxString& wxURI::GetPort ( ) const

Returns a string representation of the URI's port.

The Port of a URI is a value after the server, and must come after a colon (:).

"http://mysite.com:<port>"

Note

You can easily get the numeric value of the port by using wxAtoi() or wxString::Format().

const wxString& wxURI::GetQuery ( ) const

Returns the Query component of the URI.

The query component is what is commonly passed to a cgi application, and must come after the path component, and after a '?' character.

"http://mysite.com/mypath?<query>"

const wxString& wxURI::GetScheme ( ) const

Returns the Scheme component of the URI.

The first part of the URI.

"<scheme>://mysite.com"

const wxString& wxURI::GetServer ( ) const

Returns the Server component of the URI.

The server of the URI can be a server name or a type of IP address. See GetHostType() for the possible values for the host type of the server component.

"http://<server>/mypath"

wxString wxURI::GetUser ( ) const

Returns the username part of the userinfo component of this URI.

Note that this is explicitly depreciated by RFC 1396 and should generally be avoided if possible.

"http://<user>:<password>@mysite.com/mypath"

const wxString& wxURI::GetUserInfo ( ) const

Returns the UserInfo component of the URI.

The component of a URI before the server component that is postfixed by a '@' character.

"http://<userinfo>@mysite.com/mypath"

bool wxURI::HasFragment ( ) const

Returns true if the Fragment component of the URI exists.

bool wxURI::HasPath ( ) const

Returns true if the Path component of the URI exists.

bool wxURI::HasPort ( ) const

Returns true if the Port component of the URI exists.

bool wxURI::HasQuery ( ) const

Returns true if the Query component of the URI exists.

bool wxURI::HasScheme ( ) const

Returns true if the Scheme component of the URI exists.

bool wxURI::HasServer ( ) const

Returns true if the Server component of the URI exists.

bool wxURI::HasUserInfo ( ) const

Returns true if the User component of the URI exists.

bool wxURI::IsReference ( ) const

Returns true if a valid [absolute] URI, otherwise this URI is a URI reference and not a full URI, and this function returns false.

bool wxURI::operator== (const wxURI & *uricomp*) const

Compares this URI to another URI, and returns true if this URI equals *uricomp*, otherwise it returns false.

Parameters

uricomp	URI to compare to.

void wxURI::Resolve (const wxURI & *base,* int *flags* = `wxURI_STRICT`)

Inherits this URI from a base URI - components that do not exist in this URI are copied from the base, and if this URI's path is not an absolute path (prefixed by a '/'), then this URI's path is merged with the base's path.

For instance, resolving "../mydir" from "http://mysite.com/john/doe" results in the scheme (http) and server ("mysite.com") being copied into this URI, since they do not exist. In addition, since the path of this URI is not absolute (does not begin with '/'), the path of the base's is merged with this URI's path, resulting in the URI "http://mysite.com/john/mydir".

Parameters

base	Base URI to inherit from. Must be a full URI and not a reference.
flags	Currently either wxURI_STRICT or 0, in non-strict mode some compatibility layers are enabled to allow loopholes from RFCs prior to 2396.

static wxString wxURI::Unescape (const wxString & *uri*) `[static]`

Translates all escape sequences (normal characters and returns the result.

If you want to unescape an entire wxURI, use BuildUnescapedURI() instead, as it performs some optimizations over this method.

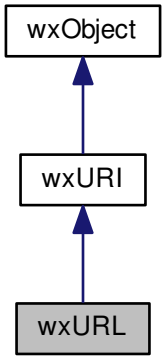
Parameters

uri	String with escaped characters to convert.

21.820 wxURL Class Reference

`#include <wx/url.h>`

Inheritance diagram for wxURL:



21.820.1 Detailed Description

wxURL is a specialization of wxURI for parsing URLs.

Please look at wxURI documentation for more info about the functions you can use to retrieve the various parts of the URL (scheme, server, port, etc).

Supports standard assignment operators, copy constructors, and comparison operators.

Library: wxNet

Category: Networking

See also

wxSocketBase, wxProtocol

Public Member Functions

- wxURL (const wxString &url=wxEmptyString)

  Constructs a URL object from the string.
- virtual ~wxURL ()

  Destroys the URL object.
- wxURLError GetError () const

  Returns the last error.
- wxInputStream * GetInputStream ()

  Creates a new input stream on the specified URL.
- wxProtocol & GetProtocol ()

  Returns a reference to the protocol which will be used to get the URL.
- bool IsOk () const

  Returns true if this object is correctly initialized, i.e. if GetError() returns wxURL_NOERR.
- void SetProxy (const wxString &url_proxy)

  Sets the proxy to use for this URL.
- wxURLError SetURL (const wxString &url)

  Initializes this object with the given URL and returns wxURL_NOERR if it's valid (see GetError() for more info).

Static Public Member Functions

- static void SetDefaultProxy (const wxString &url_proxy)

  Sets the default proxy server to use to get the URL.

Additional Inherited Members

21.820.2 Constructor & Destructor Documentation

wxURL::wxURL (const wxString & *url* = wxEmptyString)

Constructs a URL object from the string.

The URL must be valid according to RFC 1738. In particular, file URLs must be of the format "`file://hostname/path/to/file`", otherwise GetError() will return a value different from wxURL_NOERR.

It is valid to leave out the hostname but slashes must remain in place, in other words, a file URL without a hostname must contain three consecutive slashes (e.g. "`file:///somepath/myfile`").

Parameters

url	Url string to parse.

virtual wxURL::~wxURL ( ) [virtual]

Destroys the URL object.

21.820.3 Member Function Documentation

wxURLError wxURL::GetError ( ) const

Returns the last error.

This error refers to the URL parsing or to the protocol. It can be one of wxURLError.

wxInputStream* wxURL::GetInputStream ( )

Creates a new input stream on the specified URL.

You can use all but seek functionality of wxStream. Seek isn't available on all streams. For example, HTTP or FTP streams don't deal with it.

Note that this method is somewhat deprecated, all future wxWidgets applications should use wxFileSystem instead.

Example:

```
wxURL url("http://a.host/a.dir/a.file");
if (url.GetError() == wxURL_NOERR)
{
    wxInputStream *in_stream;

    in_stream = url.GetInputStream();
    // Then, you can use all IO calls of in_stream (See wxStream)
}
```

Returns

Returns the initialized stream. You will have to delete it yourself.

See also

wxInputStream

wxProtocol& wxURL::GetProtocol ( )

Returns a reference to the protocol which will be used to get the URL.

bool wxURL::IsOk ( ) const

Returns true if this object is correctly initialized, i.e. if GetError() returns wxURL_NOERR.

static void wxURL::SetDefaultProxy (const wxString & *url_proxy*) [static]

Sets the default proxy server to use to get the URL.

The string specifies the proxy like this: "<hostname>:<port number>".

Parameters

url_proxy	Specifies the proxy to use.

See also

SetProxy()

void wxURL::SetProxy (const wxString & *url_proxy*)

Sets the proxy to use for this URL.

See also

SetDefaultProxy()

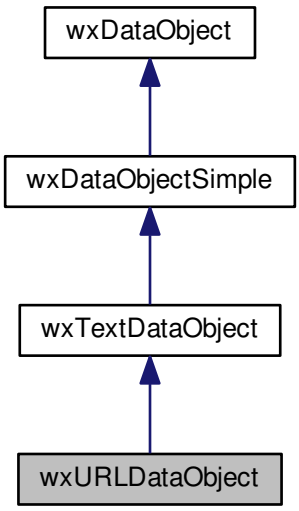
wxURLError wxURL::SetURL (const wxString & *url*)

Initializes this object with the given URL and returns wxURL_NOERR if it's valid (see GetError() for more info).

21.821 wxURLDataObject Class Reference

`#include <wx/dataobj.h>`

Inheritance diagram for wxURLDataObject:



21.821.1 Detailed Description

wxURLDataObject is a wxDataObject containing an URL and can be used e.g.

when you need to put an URL on or retrieve it from the clipboard:

```
wxTheClipboard->SetData(new wxURLDataObject(url));
```

Note

This class is derived from wxDataObjectComposite on Windows rather than wxTextDataObject on all other platforms.

Library: wxCore

Category: Clipboard and Drag & Drop

See also

Drag and Drop Overview, wxDataObject

Public Member Functions

- wxURLDataObject (const wxString &url=wxEmptyString)

  Constructor, may be used to initialize the URL.

- wxString GetURL () const

  Returns the URL stored by this object, as a string.

- void SetURL (const wxString &url)

  Sets the URL stored by this object.

Additional Inherited Members

21.821.2 Constructor & Destructor Documentation

wxURLDataObject::wxURLDataObject (const wxString & *url* = wxEmptyString)

Constructor, may be used to initialize the URL.

If *url* is empty, SetURL() can be used later.

21.821.3 Member Function Documentation

wxString wxURLDataObject::GetURL ( ) const

Returns the URL stored by this object, as a string.

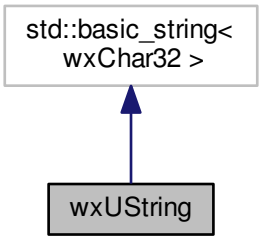
void wxURLDataObject::SetURL (const wxString & *url*)

Sets the URL stored by this object.

21.822 wxUString Class Reference

`#include <wx/ustring.h>`

Inheritance diagram for wxUString:



21.822.1 Detailed Description

wxUString is a class representing a Unicode character string where each character is stored using a 32-bit value.

This is different from wxString which may store a character either as a UTF-8 or as a UTF-16 sequence and different from `std::string` which stores a string as a sequence of simple 8-bit characters and also different from `std::wstring` which stores the string differently depending on the definition of wchar_t.

The main purpose of wxUString is a to give users a Unicode string class that has O(1) access to its content, to be identical on all platforms and to be easily convertable to wxString as well as other ways to store strings (C string literals, wide character string literals, character buffer, etc) by providing several overloads and built-in conversions to and from the various string formats.

wxUString derives from `std::basic_string<wxChar32>` and therefore offers the complete API of `std::string`.

Library: wxBase

Category: Data Structures

See also

wxString, wxString overview, Unicode overview

Public Member Functions

- wxUString ()

  Default constructor.
- wxUString (const wxUString &str)

  Copy constructor.
- wxUString (const wxChar32 *str)

  Constructs a string from a 32-bit string literal.
- wxUString (const wxU32CharBuffer &buf)

  Constructs a string from 32-bit string buffer.
- wxUString (const char *str)

  Constructs a string from C string literal using wxConvLibc to convert it to Unicode.
- wxUString (const wxCharBuffer &buf)

Constructs a string from C string buffer using wxConvLibc to convert it to Unicode.

- wxUString (const char *str, const wxMBConv &conv)

  Constructs a string from C string literal using conv to convert it to Unicode.
- wxUString (const wxCharBuffer &buf, const wxMBConv &conv)

  Constructs a string from C string literal using conv to convert it to Unicode.
- wxUString (const wxChar16 *str)

  Constructs a string from UTF-16 string literal.
- wxUString (const wxU16CharBuffer &buf)

  Constructs a string from UTF-16 string buffer.
- wxUString (const wxString &str)

  Constructs a string from wxString.
- wxUString (char ch)

  Constructs a string from using wxConvLibc to convert it to Unicode.
- wxUString (wxChar16 ch)

  Constructs a string from a UTF-16 character.
- wxUString (wxChar32 ch)

  Constructs a string from 32-bit Unicode character.
- wxUString (wxUniChar ch)

  Constructs a string from wxUniChar (returned by wxString's access operator)
- wxUString (wxUniCharRef ch)

  Constructs a string from wxUniCharRef (returned by wxString's access operator)
- wxUString (size_t n, char ch)

  Constructs a string from n characters ch.
- wxUString (size_t n, wxChar16 ch)

  Constructs a string from n characters ch.
- wxUString (size_t n, wxChar32 ch)

  Constructs a string from n characters ch.
- wxUString (size_t n, wxUniChar ch)

  Constructs a string from n characters ch.
- wxUString (size_t n, wxUniCharRef ch)

  Constructs a string from n characters ch.
- wxUString & assignFromAscii (const char *str)

  Assignment from a 7-bit ASCII string literal.
- wxUString & assignFromAscii (const char *str, size_t n)

  Assignment from a 7-bit ASCII string literal.
- wxUString & assignFromUTF8 (const char *str)

  Assignment from a UTF-8 string literal.
- wxUString & assignFromUTF8 (const char *str, size_t n)

  Assignment from a UTF-8 string literal.
- wxUString & assignFromUTF16 (const wxChar16 *str)

  Assignment from a UTF-16 string literal.
- wxUString & assignFromUTF16 (const wxChar16 *str, size_t n)

  Assignment from a UTF-16 string literal.
- wxUString & assignFromCString (const char *str)

  Assignment from a C string literal using wxConvLibc.
- wxUString & assignFromCString (const char *str, const wxMBConv &conv)

  Assignment from a C string literal using conv.
- wxCharBuffer utf8_str () const

  Conversion to a UTF-8 string.
- wxU16CharBuffer utf16_str () const

  Conversion to a UTF-16 string.

- wxWCharBuffer wc_str () const

  Conversion to a wide character string (either UTF-16 or UCS-4, depending on the size of wchar_t).
- operator wxString () const

  Implicit conversion to wxString.
- wxUString & assign (const wxUString &str)

  wxUString assignment.
- wxUString & append (const wxUString &s)

  Appending.
- wxUString & insert (size_t pos, const wxUString &s)

  Insertion.
- wxUString & operator= (const wxUString &s)

  Assignment operator.
- wxUString & operator+= (const wxUString &s)

  Concatenation operator.

Static Public Member Functions

- static wxUString FromAscii (const char *str, size_t n)

  Static construction of a wxUString from a 7-bit ASCII string.
- static wxUString FromAscii (const char *str)

  Static construction of a wxUString from a 7-bit ASCII string.
- static wxUString FromUTF8 (const char *str, size_t n)

  Static construction of a wxUString from a UTF-8 encoded string.
- static wxUString FromUTF8 (const char *str)

  Static construction of a wxUString from a UTF-8 encoded string.
- static wxUString FromUTF16 (const wxChar16 *str, size_t n)

  Static construction of a wxUString from a UTF-16 encoded string.
- static wxUString FromUTF16 (const wxChar16 *str)

  Static construction of a wxUString from a UTF-16 encoded string.

21.822.2 Constructor & Destructor Documentation

wxUString::wxUString ( )

Default constructor.

wxUString::wxUString (const wxUString & *str*)

Copy constructor.

wxUString::wxUString (const wxChar32 * *str*)

Constructs a string from a 32-bit string literal.

wxUString::wxUString (const wxU32CharBuffer & *buf*)

Constructs a string from 32-bit string buffer.

wxUString::wxUString (const char * *str*)

Constructs a string from C string literal using wxConvLibc to convert it to Unicode.

wxUString::wxUString (const wxCharBuffer & *buf*)

Constructs a string from C string buffer using wxConvLibc to convert it to Unicode.

wxUString::wxUString (const char * *str,* const wxMBConv & *conv*)

Constructs a string from C string literal using *conv* to convert it to Unicode.

wxUString::wxUString (const wxCharBuffer & *buf,* const wxMBConv & *conv*)

Constructs a string from C string literal using *conv* to convert it to Unicode.

wxUString::wxUString (const wxChar16 * *str*)

Constructs a string from UTF-16 string literal.

wxUString::wxUString (const wxU16CharBuffer & *buf*)

Constructs a string from UTF-16 string buffer.

wxUString::wxUString (const wxString & *str*)

Constructs a string from wxString.

wxUString::wxUString (char *ch*)

Constructs a string from using wxConvLibc to convert it to Unicode.

wxUString::wxUString (wxChar16 *ch*)

Constructs a string from a UTF-16 character.

wxUString::wxUString (wxChar32 *ch*)

Constructs a string from 32-bit Unicode character.

wxUString::wxUString (wxUniChar *ch*)

Constructs a string from wxUniChar (returned by wxString's access operator)

wxUString::wxUString (wxUniCharRef *ch*)

Constructs a string from wxUniCharRef (returned by wxString's access operator)

wxUString::wxUString (size_t *n,* char *ch*)

Constructs a string from *n* characters *ch*.

wxUString::wxUString (size_t *n,* wxChar16 *ch*)

Constructs a string from *n* characters *ch*.

wxUString::wxUString (size_t *n,* wxChar32 *ch*)

Constructs a string from *n* characters *ch*.

wxUString::wxUString (size_t *n,* wxUniChar *ch*)

Constructs a string from *n* characters *ch*.

wxUString::wxUString (size_t *n,* wxUniCharRef *ch*)

Constructs a string from *n* characters *ch*.

21.822.3 Member Function Documentation

wxUString& wxUString::append (const wxUString & *s*)

Appending.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single and repeated characters etc.

wxUString& wxUString::assign (const wxUString & *str*)

wxUString assignment.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single and repeated characters etc.

wxUString& wxUString::assignFromAscii (const char * *str*)

Assignment from a 7-bit ASCII string literal.

wxUString& wxUString::assignFromAscii (const char * *str,* size_t *n*)

Assignment from a 7-bit ASCII string literal.

wxUString& wxUString::assignFromCString (const char * *str*)

Assignment from a C string literal using wxConvLibc.

wxUString& wxUString::assignFromCString (const char * *str,* const wxMBConv & *conv*)

Assignment from a C string literal using *conv*.

wxUString& wxUString::assignFromUTF16 (const wxChar16 * *str*)

Assignment from a UTF-16 string literal.

wxUString& wxUString::assignFromUTF16 (const wxChar16 * *str,* size_t *n*)

Assignment from a UTF-16 string literal.

wxUString& wxUString::assignFromUTF8 (const char * *str*)

Assignment from a UTF-8 string literal.

wxUString& wxUString::assignFromUTF8 (const char * *str,* size_t *n*)

Assignment from a UTF-8 string literal.

static wxUString wxUString::FromAscii (const char * *str,* size_t *n*) `[static]`

Static construction of a wxUString from a 7-bit ASCII string.

static wxUString wxUString::FromAscii (const char * *str*) `[static]`

Static construction of a wxUString from a 7-bit ASCII string.

static wxUString wxUString::FromUTF16 (const wxChar16 * *str,* size_t *n*) `[static]`

Static construction of a wxUString from a UTF-16 encoded string.

static wxUString wxUString::FromUTF16 (const wxChar16 * *str*) `[static]`

Static construction of a wxUString from a UTF-16 encoded string.

static wxUString wxUString::FromUTF8 (const char * *str,* size_t *n*) `[static]`

Static construction of a wxUString from a UTF-8 encoded string.

static wxUString wxUString::FromUTF8 (const char * *str*) `[static]`

Static construction of a wxUString from a UTF-8 encoded string.

wxUString& wxUString::insert (size_t *pos,* const wxUString & *s*)

Insertion.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

wxUString::operator wxString ( ) const

Implicit conversion to wxString.

wxUString& wxUString::operator+= (const wxUString & *s*) `[inline]`

Concatenation operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

wxUString& wxUString::operator= (const wxUString & *s*) `[inline]`

Assignment operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

wxU16CharBuffer wxUString::utf16_str ( ) const

Conversion to a UTF-16 string.

wxCharBuffer wxUString::utf8_str ( ) const

Conversion to a UTF-8 string.

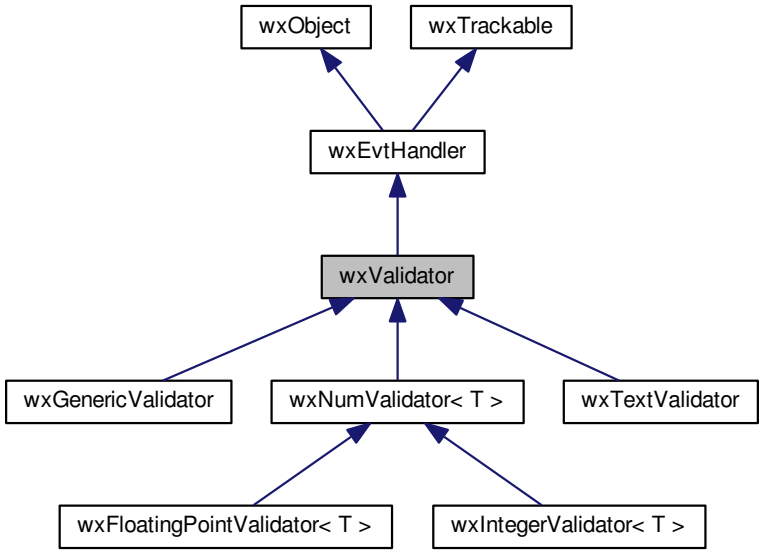
wxWCharBuffer wxUString::wc_str ( ) const

Conversion to a wide character string (either UTF-16 or UCS-4, depending on the size of wchar_t).

21.823 wxValidator Class Reference

`#include <wx/validate.h>`

Inheritance diagram for wxValidator:



21.823.1 Detailed Description

wxValidator is the base class for a family of validator classes that mediate between a class of control, and application data.

A validator has three major roles:

1. To transfer data from a C++ variable or own storage to and from a control.
2. To validate data in a control, and show an appropriate error message.
3. To filter events (such as keystrokes), thereby changing the behaviour of the associated control.

Validators can be plugged into controls dynamically.

To specify a default, "null" validator, use wxDefaultValidator.

For more information, please see wxValidator Overview.

Library: wxCore

Category: Validators

Predefined objects/pointers: wxDefaultValidator

See also

wxValidator Overview, wxTextValidator, wxGenericValidator, wxIntegerValidator, wxFloatingPointValidator

Public Member Functions

- wxValidator ()

  Constructor.
- virtual ~wxValidator ()

  Destructor.
- virtual wxObject * Clone () const

  All validator classes must implement the Clone() function, which returns an identical copy of itself.
- wxWindow * GetWindow () const

  Returns the window associated with the validator.
- void SetWindow (wxWindow *window)

  Associates a window with the validator.
- virtual bool TransferFromWindow ()

  This overridable function is called when the value in the window must be transferred to the validator.
- virtual bool TransferToWindow ()

  This overridable function is called when the value associated with the validator must be transferred to the window.
- virtual bool Validate (wxWindow *parent)

  This overridable function is called when the value in the associated window must be validated.

Static Public Member Functions

- static void SuppressBellOnError (bool suppress=true)

  This functions switches on or turns off the error sound produced by the validators if an invalid key is pressed.
- static bool IsSilent ()

  Returns if the error sound is currently disabled.

Additional Inherited Members

21.823.2 Constructor & Destructor Documentation

wxValidator::wxValidator ( )

Constructor.

virtual wxValidator::~wxValidator ( ) [virtual]

Destructor.

21.823.3 Member Function Documentation

virtual wxObject* wxValidator::Clone ( ) const [virtual]

All validator classes must implement the Clone() function, which returns an identical copy of itself.

This is because validators are passed to control constructors as references which must be copied. Unlike objects such as pens and brushes, it does not make sense to have a reference counting scheme to do this cloning because all validators should have separate data.

Returns

This base function returns NULL.

Reimplemented in wxGenericValidator, and wxTextValidator.

wxWindow* wxValidator::GetWindow ( ) const

Returns the window associated with the validator.

static bool wxValidator::IsSilent ( ) [static]

Returns if the error sound is currently disabled.

void wxValidator::SetWindow (wxWindow * *window*)

Associates a window with the validator.

This function is automatically called by wxWidgets when creating a wxWindow-derived class instance which takes a wxValidator reference.

E.g.

```
new wxTextCtrl(this, wxID_ANY, wxEmptyString,
      wxDefaultPosition, wxDefaultSize, 0,
               wxTextValidator(wxFILTER_ALPHA, &g_data.m_string));
```

will automatically link the wxTextValidator instance with the wxTextCtrl instance.

static void wxValidator::SuppressBellOnError (bool *suppress* = true) [static]

This functions switches on or turns off the error sound produced by the validators if an invalid key is pressed.

Since

2.9.1

Parameters

suppress	If true, error sound is not played when a validator detects an error. If false, error sound is enabled.

virtual bool wxValidator::TransferFromWindow ( ) [virtual]

This overridable function is called when the value in the window must be transferred to the validator.

Returns

false if there is a problem.

Reimplemented in wxTextValidator, wxGenericValidator, and wxNumValidator< T >.

virtual bool wxValidator::TransferToWindow ( ) [virtual]

This overridable function is called when the value associated with the validator must be transferred to the window.

Returns

false if there is a problem.

Reimplemented in wxTextValidator, wxGenericValidator, and wxNumValidator< T >.

virtual bool wxValidator::Validate (wxWindow * *parent*) [virtual]

This overridable function is called when the value in the associated window must be validated.

Parameters

parent	The parent of the window associated with the validator.

Returns

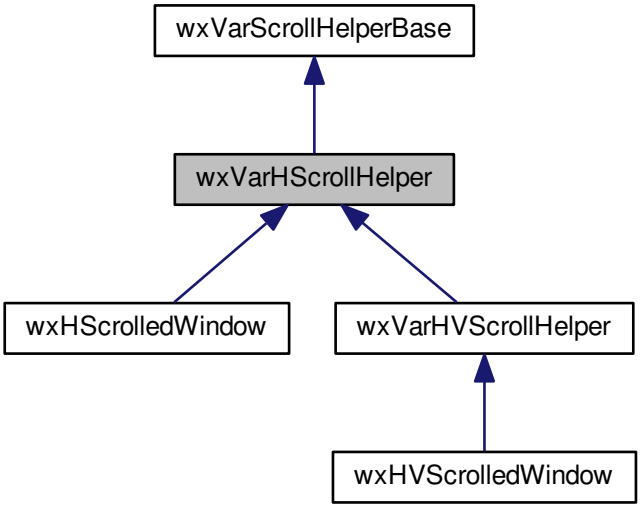
false if the value in the window is not valid; you may pop up an error dialog.

Reimplemented in wxTextValidator.

21.824 wxVarHScrollHelper Class Reference

`#include <wx/vscroll.h>`

Inheritance diagram for wxVarHScrollHelper:



21.824.1 Detailed Description

This class provides functions wrapping the wxVarScrollHelperBase class, targeted for horizontal-specific scrolling.

Like wxVarScrollHelperBase, this class is mostly only useful to those classes built into wxWidgets deriving from here, and this documentation is mostly only provided for referencing the functions provided by this class. You will likely want to derive your window from wxHScrolledWindow rather than from here directly.

Library: wxCore

Category: Miscellaneous Windows

See also

wxHScrolledWindow, wxHVScrolledWindow, wxVScrolledWindow

Public Member Functions

- wxVarHScrollHelper (wxWindow *winToScroll)

  Constructor taking the target window to be scrolled by this helper class.
- size_t GetColumnCount () const

  Returns the number of columns the target window contains.
- size_t GetVisibleColumnsBegin () const

  Returns the index of the first visible column based on the scroll position.
- size_t GetVisibleColumnsEnd () const

  Returns the index of the last visible column based on the scroll position.
- bool IsColumnVisible (size_t column) const

Returns true if the given column is currently visible (even if only partially visible) or false otherwise.

- virtual void RefreshColumn (size_t column)

  Triggers a refresh for just the given column's area of the window if it's visible.
- virtual void RefreshColumns (size_t from, size_t to)

  Triggers a refresh for the area between the specified range of columns given (inclusively).
- virtual bool ScrollColumnPages (int pages)

  Scroll by the specified number of pages which may be positive (to scroll right) or negative (to scroll left).
- virtual bool ScrollColumns (int columns)

  Scroll by the specified number of columns which may be positive (to scroll right) or negative (to scroll left).
- bool ScrollToColumn (size_t column)

  Scroll to the specified column.
- void SetColumnCount (size_t columnCount)

  Set the number of columns the window contains.

Protected Member Functions

- virtual wxCoord EstimateTotalWidth () const

  This class forwards calls from EstimateTotalSize() to this function so derived classes can override either just the height or the width estimation, or just estimate both differently if desired in any wxHVScrolledWindow derived class.
- virtual void OnGetColumnsWidthHint (size_t columnMin, size_t columnMax) const

  This function doesn't have to be overridden but it may be useful to do so if calculating the columns' sizes is a relatively expensive operation as it gives your code a chance to calculate several of them at once and cache the result if necessary.
- virtual wxCoord OnGetColumnWidth (size_t column) const =0

  This function must be overridden in the derived class, and should return the width of the given column in pixels.

21.824.2 Constructor & Destructor Documentation

wxVarHScrollHelper::wxVarHScrollHelper (wxWindow * *winToScroll*)

Constructor taking the target window to be scrolled by this helper class.

This will attach scroll event handlers to the target window to catch and handle scroll events appropriately.

21.824.3 Member Function Documentation

virtual wxCoord wxVarHScrollHelper::EstimateTotalWidth ( ) const `[protected], [virtual]`

This class forwards calls from EstimateTotalSize() to this function so derived classes can override either just the height or the width estimation, or just estimate both differently if desired in any wxHVScrolledWindow derived class.

Note

This function will not be called if EstimateTotalSize() is overridden in your derived class.

size_t wxVarHScrollHelper::GetColumnCount ( ) const

Returns the number of columns the target window contains.

See also

SetColumnCount()

size_t wxVarHScrollHelper::GetVisibleColumnsBegin ( ) const

Returns the index of the first visible column based on the scroll position.

size_t wxVarHScrollHelper::GetVisibleColumnsEnd ( ) const

Returns the index of the last visible column based on the scroll position.

This includes the last column even if it is only partially visible.

bool wxVarHScrollHelper::IsColumnVisible (size_t *column*) const

Returns true if the given column is currently visible (even if only partially visible) or false otherwise.

virtual void wxVarHScrollHelper::OnGetColumnsWidthHint (size_t *columnMin,* size_t *columnMax*) const `[protected], [virtual]`

This function doesn't have to be overridden but it may be useful to do so if calculating the columns' sizes is a relatively expensive operation as it gives your code a chance to calculate several of them at once and cache the result if necessary.

OnGetColumnsWidthHint() is normally called just before OnGetColumnWidth() but you shouldn't rely on the latter being called for all columns in the interval specified here. It is also possible that OnGetColumnWidth() will be called for units outside of this interval, so this is really just a hint, not a promise.

Finally, note that *columnMin* is inclusive, while *columnMax* is exclusive.

virtual wxCoord wxVarHScrollHelper::OnGetColumnWidth (size_t *column*) const `[protected], [pure virtual]`

This function must be overridden in the derived class, and should return the width of the given column in pixels.

virtual void wxVarHScrollHelper::RefreshColumn (size_t *column*) `[virtual]`

Triggers a refresh for just the given column's area of the window if it's visible.

virtual void wxVarHScrollHelper::RefreshColumns (size_t *from,* size_t *to*) `[virtual]`

Triggers a refresh for the area between the specified range of columns given (inclusively).

virtual bool wxVarHScrollHelper::ScrollColumnPages (int *pages*) `[virtual]`

Scroll by the specified number of pages which may be positive (to scroll right) or negative (to scroll left).

virtual bool wxVarHScrollHelper::ScrollColumns (int *columns*) `[virtual]`

Scroll by the specified number of columns which may be positive (to scroll right) or negative (to scroll left).

Returns

true if the window was scrolled, false otherwise (for example, if we're trying to scroll right but we are already showing the last column).

bool wxVarHScrollHelper::ScrollToColumn (size_t *column*)

Scroll to the specified column.

It will become the first visible column in the window.

Returns

true if we scrolled the window, false if nothing was done.

void wxVarHScrollHelper::SetColumnCount (size_t *columnCount*)

Set the number of columns the window contains.

The derived class must provide the widths for all columns with indices up to the one given here in it's OnGet↩ ColumnWidth() implementation.

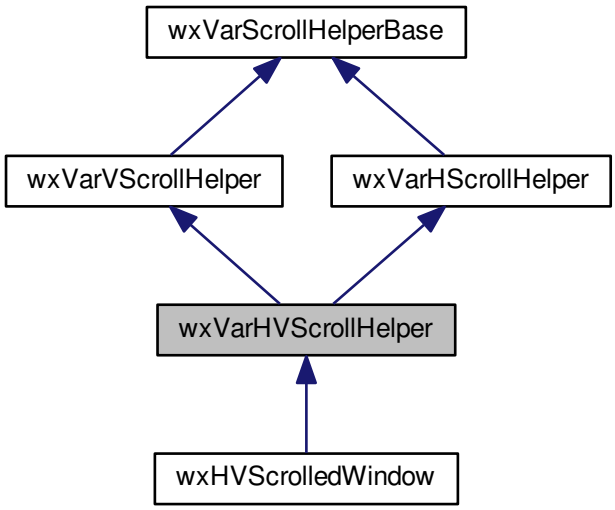
See also

GetColumnCount()

21.825 wxVarHVScrollHelper Class Reference

`#include <wx/vscroll.h>`

Inheritance diagram for wxVarHVScrollHelper:



21.825.1 Detailed Description

This class provides functions wrapping the wxVarHScrollHelper and wxVarVScrollHelper classes, targeted for scrolling a window in both axis.

Since this class is also the join class of the horizontal and vertical scrolling functionality, it also addresses some wrappers that help avoid the need to specify class scope in your wxHVScrolledWindow derived class when using wxVarScrollHelperBase functionality.

Like all three of it's scroll helper base classes, this class is mostly only useful to those classes built into wxWidgets deriving from here, and this documentation is mostly only provided for referencing the functions provided by this class. You will likely want to derive your window from wxHVScrolledWindow rather than from here directly.

Library: wxCore

Category: Miscellaneous Windows

See also

wxHScrolledWindow, wxHVScrolledWindow, wxVScrolledWindow

Public Member Functions

- wxVarHVScrollHelper (wxWindow *winToScroll)

  Constructor taking the target window to be scrolled by this helper class.
- void EnablePhysicalScrolling (bool vscrolling=true, bool hscrolling=true)

  With physical scrolling on (when this is true), the device origin is changed properly when a wxPaintDC is prepared, children are actually moved and laid out properly, and the contents of the window (pixels) are actually moved.
- wxSize GetRowColumnCount () const

  Returns the number of columns and rows the target window contains.
- wxPosition GetVisibleBegin () const

  Returns the index of the first visible column and row based on the current scroll position.
- wxPosition GetVisibleEnd () const

  Returns the index of the last visible column and row based on the scroll position.
- void SetRowColumnCount (size_t rowCount, size_t columnCount)

  Set the number of rows and columns the target window will contain.

- bool IsVisible (size_t row, size_t column) const

  Returns true if both the given row and column are currently visible (even if only partially visible) or false otherwise.
- bool IsVisible (const wxPosition &pos) const

  Returns true if both the given row and column are currently visible (even if only partially visible) or false otherwise.

- virtual void RefreshRowColumn (size_t row, size_t column)

  Triggers a refresh for just the area shared between the given row and column of the window if it is visible.
- virtual void RefreshRowColumn (const wxPosition &pos)

  Triggers a refresh for just the area shared between the given row and column of the window if it is visible.

- virtual void RefreshRowsColumns (size_t fromRow, size_t toRow, size_t fromColumn, size_t toColumn)

  Triggers a refresh for the visible area shared between all given rows and columns (inclusive) of the window.
- virtual void RefreshRowsColumns (const wxPosition &from, const wxPosition &to)

  Triggers a refresh for the visible area shared between all given rows and columns (inclusive) of the window.

- bool ScrollToRowColumn (size_t row, size_t column)

  Scroll to the specified row and column.
- bool ScrollToRowColumn (const wxPosition &pos)

  Scroll to the specified row and column.

- wxPosition VirtualHitTest (wxCoord x, wxCoord y) const

  Returns the virtual scroll unit under the device unit given accounting for scroll position or `wxNOT_FOUND` (for the row, column, or possibly both values) if none.

- wxPosition VirtualHitTest (const wxPoint &pos) const

  Returns the virtual scroll unit under the device unit given accounting for scroll position or `wxNOT_FOUND` (for the row, column, or possibly both values) if none.

Additional Inherited Members

21.825.2 Constructor & Destructor Documentation

wxVarHVScrollHelper::wxVarHVScrollHelper (wxWindow * *winToScroll*)

Constructor taking the target window to be scrolled by this helper class.

This will attach scroll event handlers to the target window to catch and handle scroll events appropriately.

21.825.3 Member Function Documentation

void wxVarHVScrollHelper::EnablePhysicalScrolling (bool *vscrolling* = `true`, bool *hscrolling* = `true`)

With physical scrolling on (when this is true), the device origin is changed properly when a wxPaintDC is prepared, children are actually moved and laid out properly, and the contents of the window (pixels) are actually moved.

When this is false, you are responsible for repainting any invalidated areas of the window yourself to account for the new scroll position.

Parameters

vscrolling	Specifies if physical scrolling should be turned on when scrolling vertically.
hscrolling	Specifies if physical scrolling should be turned on when scrolling horizontally.

wxSize wxVarHVScrollHelper::GetRowColumnCount ( ) const

Returns the number of columns and rows the target window contains.

See also

SetRowColumnCount()

wxPosition wxVarHVScrollHelper::GetVisibleBegin ( ) const

Returns the index of the first visible column and row based on the current scroll position.

wxPosition wxVarHVScrollHelper::GetVisibleEnd ( ) const

Returns the index of the last visible column and row based on the scroll position.

This includes any partially visible columns or rows.

bool wxVarHVScrollHelper::IsVisible (size_t *row*, size_t *column*) const

Returns true if both the given row and column are currently visible (even if only partially visible) or false otherwise.

bool wxVarHVScrollHelper::IsVisible (const wxPosition & *pos*) const

Returns true if both the given row and column are currently visible (even if only partially visible) or false otherwise.

virtual void wxVarHVScrollHelper::RefreshRowColumn (size_t *row,* size_t *column*) [virtual]

Triggers a refresh for just the area shared between the given row and column of the window if it is visible.

virtual void wxVarHVScrollHelper::RefreshRowColumn (const wxPosition & *pos*) [virtual]

Triggers a refresh for just the area shared between the given row and column of the window if it is visible.

virtual void wxVarHVScrollHelper::RefreshRowsColumns (size_t *fromRow,* size_t *toRow,* size_t *fromColumn,* size_t *toColumn*) [virtual]

Triggers a refresh for the visible area shared between all given rows and columns (inclusive) of the window.

If the target window for both orientations is the same, the rectangle of cells is refreshed; if the target windows differ, the entire client size opposite the orientation direction is refreshed between the specified limits.

virtual void wxVarHVScrollHelper::RefreshRowsColumns (const wxPosition & *from,* const wxPosition & *to*) [virtual]

Triggers a refresh for the visible area shared between all given rows and columns (inclusive) of the window.

If the target window for both orientations is the same, the rectangle of cells is refreshed; if the target windows differ, the entire client size opposite the orientation direction is refreshed between the specified limits.

bool wxVarHVScrollHelper::ScrollToRowColumn (size_t *row,* size_t *column*)

Scroll to the specified row and column.

It will become the first visible row and column in the window. Returns true if we scrolled the window, false if nothing was done.

bool wxVarHVScrollHelper::ScrollToRowColumn (const wxPosition & *pos*)

Scroll to the specified row and column.

It will become the first visible row and column in the window. Returns true if we scrolled the window, false if nothing was done.

void wxVarHVScrollHelper::SetRowColumnCount (size_t *rowCount,* size_t *columnCount*)

Set the number of rows and columns the target window will contain.

The derived class must provide the sizes for all rows and columns with indices up to the ones given here in it's OnGetRowHeight() and OnGetColumnWidth() implementations, respectively.

See also

GetRowColumnCount()

wxPosition wxVarHVScrollHelper::VirtualHitTest (wxCoord *x*, wxCoord *y*) const

Returns the virtual scroll unit under the device unit given accounting for scroll position or `wxNOT_FOUND` (for the row, column, or possibly both values) if none.

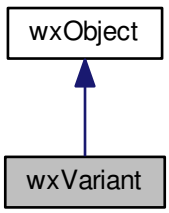
wxPosition wxVarHVScrollHelper::VirtualHitTest (const wxPoint & *pos*) const

Returns the virtual scroll unit under the device unit given accounting for scroll position or `wxNOT_FOUND` (for the row, column, or possibly both values) if none.

21.826 wxVariant Class Reference

`#include <wx/variant.h>`

Inheritance diagram for wxVariant:



21.826.1 Detailed Description

The wxVariant class represents a container for any type.

A variant's value can be changed at run time, possibly to a different type of value.

Note

> As of wxWidgets 2.9.1, wxAny has become the preferred variant class. While most controls still use wxVariant in their interface, you can start using wxAny in your code because of an implicit conversion layer. See below for more information.

As standard, wxVariant can store values of type bool, wxChar, double, long, string, string list, time, date, void pointer, list of strings, and list of variants. However, an application can extend wxVariant's capabilities by deriving from the class wxVariantData and using the wxVariantData form of the wxVariant constructor or assignment operator to assign this data to a variant. Actual values for user-defined types will need to be accessed via the wxVariantData object, unlike the case for basic data types where convenience functions such as GetLong() can be used.

Under Microsoft Windows, three additional wxVariantData-derived classes – wxVariantDataCurrency, wxVariantDataErrorCode and wxVariantDataSafeArray – are available for interoperation with OLE VARIANT when using wxAutomationObject.

Pointers to any wxObject derived class can also easily be stored in a wxVariant. wxVariant will then use wxWidgets' built-in RTTI system to set the type name (returned by GetType()) and to perform type-safety checks at runtime.

This class is useful for reducing the programming for certain tasks, such as an editor for different data types, or a remote procedure call protocol.

An optional name member is associated with a wxVariant. This might be used, for example, in CORBA or OLE automation classes, where named parameters are required.

Note that as of wxWidgets 2.7.1, wxVariant is reference counted. Additionally, the convenience macros DECLARE_VARIANT_OBJECT() and IMPLEMENT_VARIANT_OBJECT() were added so that adding (limited) support for conversion to and from wxVariant can be very easily implemented without modifying either wxVariant or the class to be stored by wxVariant. Since assignment operators cannot be declared outside the class, the shift left operators are used like this:

```
// in the header file
DECLARE_VARIANT_OBJECT(MyClass)

// in the implementation file
IMPLEMENT_VARIANT_OBJECT(MyClass)

// in the user code
wxVariant variant;
MyClass value;
variant << value;

// or
value << variant;
```

For this to work, MyClass must derive from wxObject, implement the wxWidgets RTTI system and support the assignment operator and equality operator for itself. Ideally, it should also be reference counted to make copying operations cheap and fast. This can be most easily implemented using the reference counting support offered by wxObject itself. By default, wxWidgets already implements the shift operator conversion for a few of its drawing related classes:

```
IMPLEMENT_VARIANT_OBJECT(wxColour)
IMPLEMENT_VARIANT_OBJECT(wxImage)
IMPLEMENT_VARIANT_OBJECT(wxIcon)
IMPLEMENT_VARIANT_OBJECT(wxBitmap)
```

Note that as of wxWidgets 2.9.0, wxVariantData no longer inherits from wxObject and wxVariant no longer uses the type-unsafe wxList class for list operations but the type-safe wxVariantList class. Also, wxVariantData now supports the wxVariantData::Clone() function for implementing the Unshare() function. wxVariantData::Clone() is implemented automatically by IMPLEMENT_VARIANT_OBJECT().

Since wxVariantData no longer derives from wxObject, any code that tests the type of the data using wxDynamicCast() will require adjustment. You can use the macro wxDynamicCastVariantData() with the same arguments as wxDynamicCast(), to use C++ RTTI type information instead of wxWidgets RTTI.

21.826.2 wxVariant to wxAny Conversion Layer

wxAny is a more modern, template-based variant class. It is not directly compatible with wxVariant, but there is a transparent conversion layer.

Following is an example how to use these conversions with wxPropertyGrid's property class wxPGProperty (which currently uses wxVariants both internally and in the public API):

```
// Get property value as wxAny instead of wxVariant
wxAny value = property->GetValue();

// Do something with it
DoSomethingWithString(value.As<wxString>());

// Write back new value to property
value = "New Value";
property->SetValue(value);
```

Some caveats:

- In wxAny, there are no separate types for handling integers of different sizes, so converting wxAny with 'long long' value will yield wxVariant of "long" type when the value is small enough to fit in without overflow. Otherwise, variant type "longlong" is used. Also note that wxAny holding unsigned integer will always be converted to "ulonglong" wxVariant type.

- Unlike wxVariant, wxAny does not store a (rarely needed) name string.

- Because of implicit conversion of wxVariant to wxAny, wxAny cannot usually contain value of type wxVariant. In other words, any.CheckType<wxVariant>() can never return true.

Supplied conversion functions will automatically work with all built-in wxVariant types, and also with all user-specified types generated using IMPLEMENT_VARIANT_OBJECT(). For hand-built wxVariantData classes, you will need to use supplied macros in a following manner:

```
// Declare wxVariantData for data type Foo
class wxVariantDataFoo: public wxVariantData
{
public:
    // interface
    // ...

    DECLARE_WXANY_CONVERSION()
protected:
    // data storage etc
    // ...
};

IMPLEMENT_TRIVIAL_WXANY_CONVERSION(Foo, wxVariantDataFoo)
```

Library: wxBase

Category: Data Structures

See also

wxVariantData, wxAny

Public Member Functions

- wxVariant ()

  Default constructor.
- wxVariant (wxVariantData *data, const wxString &name=wxEmptyString)

  Constructs a variant directly with a wxVariantData object.
- wxVariant (const wxVariant &variant)

  Constructs a variant from another variant by increasing the reference count.
- wxVariant (const wxAny &any)

  Constructs a variant by converting it from wxAny.
- wxVariant (const wxChar *value, const wxString &name=wxEmptyString)

  Constructs a variant from a wide string literal.
- wxVariant (const wxString &value, const wxString &name=wxEmptyString)

  Constructs a variant from a string.
- wxVariant (wxChar value, const wxString &name=wxEmptyString)

  Constructs a variant from a wide char.
- wxVariant (long value, const wxString &name=wxEmptyString)

  Constructs a variant from a long.
- wxVariant (bool value, const wxString &name=wxEmptyString)

  Constructs a variant from a bool.
- wxVariant (double value, const wxString &name=wxEmptyString)

  Constructs a variant from a double.
- wxVariant (wxLongLong value, const wxString &name=wxEmptyString)

  Constructs a variant from a wxLongLong.

- wxVariant (wxULongLong value, const wxString &name=wxEmptyString)

  Constructs a variant from a wxULongLong.
- wxVariant (const wxVariantList &value, const wxString &name=wxEmptyString)

  Constructs a variant from a list of variants.
- wxVariant (void ∗value, const wxString &name=wxEmptyString)

  Constructs a variant from a void pointer.
- wxVariant (wxObject ∗value, const wxString &name=wxEmptyString)

  Constructs a variant from a pointer to an wxObject derived class.
- wxVariant (const wxDateTime &val, const wxString &name=wxEmptyString)

  Constructs a variant from a wxDateTime.
- wxVariant (const wxArrayString &val, const wxString &name=wxEmptyString)

  Constructs a variant from a wxArrayString.
- virtual ∼wxVariant ()

  Destructor.
- wxAny GetAny () const

  Converts wxVariant into wxAny.
- wxArrayString GetArrayString () const

  Returns the string array value.
- bool GetBool () const

  Returns the boolean value.
- wxUniChar GetChar () const

  Returns the character value.
- wxVariantData ∗ GetData () const

  Returns a pointer to the internal variant data.
- wxDateTime GetDateTime () const

  Returns the date value.
- double GetDouble () const

  Returns the floating point value.
- long GetLong () const

  Returns the integer value.
- wxLongLong GetLongLong () const

  Returns the signed 64-bit integer value.
- const wxString & GetName () const

  Returns a constant reference to the variant name.
- wxString GetString () const

  Gets the string value.
- wxString GetType () const

  Returns the value type as a string.
- wxULongLong GetULongLong () const

  Returns the unsigned 64-bit integer value.
- void ∗ GetVoidPtr () const

  Gets the void pointer value.
- wxObject ∗ GetWxObjectPtr () const

  Gets the wxObject pointer value.
- bool IsNull () const

  Returns true if there is no data associated with this variant, false if there is data.
- bool IsType (const wxString &type) const

  Returns true if type matches the type of the variant, false otherwise.
- bool IsValueKindOf (const wxClassInfo ∗type) const

  Returns true if the data is derived from the class described by type, false otherwise.
- void MakeNull ()

Makes the variant null by deleting the internal data.

- wxString MakeString () const

  Makes a string representation of the variant value (for any type).
- bool Member (const wxVariant &value) const

  Returns true if value matches an element in the list.
- void SetData (wxVariantData *data)

  Sets the internal variant data, deleting the existing data if there is any.
- bool Unshare ()

  Makes sure that any data associated with this variant is not shared with other variants.
- void * operator void * () const

  Operator for implicit conversion to a pointer to a void, using GetVoidPtr().
- char operator wxChar () const

  Operator for implicit conversion to a wxChar, using GetChar().
- void * operator wxDateTime () const

  Operator for implicit conversion to a pointer to a wxDateTime, using GetDateTime().
- wxString operator wxString () const

  Operator for implicit conversion to a string, using MakeString().

List Functionality

- wxVariant operator[] (size_t idx) const

  Returns the value at idx (zero-based).
- wxVariant & operator[] (size_t idx)

  Returns a reference to the value at idx (zero-based).
- void Append (const wxVariant &value)

  Appends a value to the list.
- void Clear ()

  Makes the variant null by deleting the internal data and set the name to wxEmptyString.
- void ClearList ()

  Deletes the contents of the list.
- bool Delete (size_t item)

  Deletes the zero-based item from the list.
- size_t GetCount () const

  Returns the number of elements in the list.
- wxVariantList & GetList () const

  Returns a reference to the wxVariantList class used by wxVariant if this wxVariant is currently a list of variants.
- void Insert (const wxVariant &value)

  Inserts a value at the front of the list.
- void NullList ()

  Makes an empty list.

- bool Convert (long *value) const

  Retrieves and converts the value of this variant to the type that value is.
- bool Convert (bool *value) const

  Retrieves and converts the value of this variant to the type that value is.
- bool Convert (double *value) const

  Retrieves and converts the value of this variant to the type that value is.
- bool Convert (wxString *value) const

  Retrieves and converts the value of this variant to the type that value is.
- bool Convert (wxChar *value) const

  Retrieves and converts the value of this variant to the type that value is.
- bool Convert (wxLongLong *value) const

  Retrieves and converts the value of this variant to the type that value is.
- bool Convert (wxULongLong *value) const

Retrieves and converts the value of this variant to the type that value is.

- bool Convert (wxDateTime *value) const

  Retrieves and converts the value of this variant to the type that value is.

- bool operator!= (const wxVariant &value) const

  Inequality test operator.
- bool operator!= (const wxString &value) const

  Inequality test operator.
- bool operator!= (const wxChar *value) const

  Inequality test operator.
- bool operator!= (wxChar value) const

  Inequality test operator.
- bool operator!= (long value) const

  Inequality test operator.
- bool operator!= (bool value) const

  Inequality test operator.
- bool operator!= (double value) const

  Inequality test operator.
- bool operator!= (wxLongLong value) const

  Inequality test operator.
- bool operator!= (wxULongLong value) const

  Inequality test operator.
- bool operator!= (void *value) const

  Inequality test operator.
- bool operator!= (wxObject *value) const

  Inequality test operator.
- bool operator!= (const wxVariantList &value) const

  Inequality test operator.
- bool operator!= (const wxArrayString &value) const

  Inequality test operator.
- bool operator!= (const wxDateTime &value) const

  Inequality test operator.

- void operator= (const wxVariant &value)

  Assignment operator, using reference counting if possible.
- void operator= (wxVariantData *value)

  Assignment operator, using reference counting if possible.
- void operator= (const wxString &value)

  Assignment operator, using reference counting if possible.
- void operator= (const wxChar *value)

  Assignment operator, using reference counting if possible.
- void operator= (wxChar value)

  Assignment operator, using reference counting if possible.
- void operator= (long value)

  Assignment operator, using reference counting if possible.
- void operator= (bool value)

  Assignment operator, using reference counting if possible.
- void operator= (double value)

  Assignment operator, using reference counting if possible.
- bool operator= (wxLongLong value) const

  Assignment operator, using reference counting if possible.

- bool operator= (wxULongLong value) const

  Assignment operator, using reference counting if possible.
- void operator= (void *value)

  Assignment operator, using reference counting if possible.
- void operator= (wxObject *value)

  Assignment operator, using reference counting if possible.
- void operator= (const wxVariantList &value)

  Assignment operator, using reference counting if possible.
- void operator= (const wxDateTime &value)

  Assignment operator, using reference counting if possible.
- void operator= (const wxArrayString &value)

  Assignment operator, using reference counting if possible.

- bool operator== (const wxVariant &value) const

  Equality test operator.
- bool operator== (const wxString &value) const

  Equality test operator.
- bool operator== (const wxChar *value) const

  Equality test operator.
- bool operator== (wxChar value) const

  Equality test operator.
- bool operator== (long value) const

  Equality test operator.
- bool operator== (bool value) const

  Equality test operator.
- bool operator== (double value) const

  Equality test operator.
- bool operator== (wxLongLong value) const

  Equality test operator.
- bool operator== (wxULongLong value) const

  Equality test operator.
- bool operator== (void *value) const

  Equality test operator.
- bool operator== (wxObject *value) const

  Equality test operator.
- bool operator== (const wxVariantList &value) const

  Equality test operator.
- bool operator== (const wxArrayString &value) const

  Equality test operator.
- bool operator== (const wxDateTime &value) const

  Equality test operator.

- double operator double () const

  Operators for implicit conversion, using appropriate getter member function.
- long operator long () const

  Operators for implicit conversion, using appropriate getter member function.
- wxLongLong operator wxLongLong () const

  Operators for implicit conversion, using appropriate getter member function.
- wxULongLong operator wxULongLong () const

  Operators for implicit conversion, using appropriate getter member function.

Additional Inherited Members

21.826.3 Constructor & Destructor Documentation

wxVariant::wxVariant ( )

Default constructor.

wxVariant::wxVariant (wxVariantData * *data*, const wxString & *name* = wxEmptyString)

Constructs a variant directly with a wxVariantData object.

wxVariant will take ownership of the wxVariantData and will not increase its reference count.

wxVariant::wxVariant (const wxVariant & *variant*)

Constructs a variant from another variant by increasing the reference count.

wxVariant::wxVariant (const wxAny & *any*)

Constructs a variant by converting it from wxAny.

wxVariant::wxVariant (const wxChar * *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a wide string literal.

wxVariant::wxVariant (const wxString & *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a string.

wxVariant::wxVariant (wxChar *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a wide char.

wxVariant::wxVariant (long *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a long.

wxVariant::wxVariant (bool *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a bool.

wxVariant::wxVariant (double *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a double.

wxVariant::wxVariant (wxLongLong *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a wxLongLong.

wxVariant::wxVariant (wxULongLong *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a wxULongLong.

wxVariant::wxVariant (const wxVariantList & *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a list of variants.

wxVariant::wxVariant (void * *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a void pointer.

wxVariant::wxVariant (wxObject * *value*, const wxString & *name* = wxEmptyString)

Constructs a variant from a pointer to an wxObject derived class.

wxVariant::wxVariant (const wxDateTime & *val*, const wxString & *name* = wxEmptyString)

Constructs a variant from a wxDateTime.

wxVariant::wxVariant (const wxArrayString & *val*, const wxString & *name* = wxEmptyString)

Constructs a variant from a wxArrayString.

virtual wxVariant::~wxVariant ( ) [virtual]

Destructor.

Note

wxVariantData's destructor is protected, so wxVariantData cannot usually be deleted. Instead, wxVariantData::DecRef() should be called. See reference-counted object destruction for more info.

21.826.4 Member Function Documentation

void wxVariant::Append (const wxVariant & *value*)

Appends a value to the list.

void wxVariant::Clear ( )

Makes the variant null by deleting the internal data and set the name to wxEmptyString.

void wxVariant::ClearList ( )

Deletes the contents of the list.

bool wxVariant::Convert (long * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (bool * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (double * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (wxString * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (wxChar * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (wxLongLong * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (wxULongLong * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Convert (wxDateTime * *value*) const

Retrieves and converts the value of this variant to the type that *value* is.

bool wxVariant::Delete (size_t *item*)

Deletes the zero-based *item* from the list.

wxAny wxVariant::GetAny ( ) const

Converts wxVariant into wxAny.

wxArrayString wxVariant::GetArrayString ( ) const

Returns the string array value.

bool wxVariant::GetBool ( ) const

Returns the boolean value.

wxUniChar wxVariant::GetChar ( ) const

Returns the character value.

size_t wxVariant::GetCount ( ) const

Returns the number of elements in the list.

wxVariantData* wxVariant::GetData ( ) const

Returns a pointer to the internal variant data.

To take ownership of this data, you must call its wxVariantData::IncRef() method. When you stop using it, wxVariantData::DecRef() must be called as well.

wxDateTime wxVariant::GetDateTime ( ) const

Returns the date value.

double wxVariant::GetDouble ( ) const

Returns the floating point value.

wxVariantList& wxVariant::GetList ( ) const

Returns a reference to the wxVariantList class used by wxVariant if this wxVariant is currently a list of variants.

long wxVariant::GetLong ( ) const

Returns the integer value.

wxLongLong wxVariant::GetLongLong ( ) const

Returns the signed 64-bit integer value.

const wxString& wxVariant::GetName ( ) const

Returns a constant reference to the variant name.

wxString wxVariant::GetString ( ) const

Gets the string value.

wxString wxVariant::GetType ( ) const

Returns the value type as a string.

The built-in types are:

- "bool"
- "char"
- "datetime"
- "double"
- "list"

- "long"
- "longlong"
- "string"
- "ulonglong"
- "arrstring"
- "void*"

If the variant is null, the value type returned is the string "null" (not the empty string).

wxULongLong wxVariant::GetULongLong ( ) const

Returns the unsigned 64-bit integer value.

void* wxVariant::GetVoidPtr ( ) const

Gets the void pointer value.

Notice that this method can be used for null objects (i.e. those for which IsNull() returns true) and will return NULL for them.

wxObject* wxVariant::GetWxObjectPtr ( ) const

Gets the wxObject pointer value.

void wxVariant::Insert (const wxVariant & *value*)

Inserts a value at the front of the list.

bool wxVariant::IsNull ( ) const

Returns true if there is no data associated with this variant, false if there is data.

bool wxVariant::IsType (const wxString & *type*) const

Returns true if *type* matches the type of the variant, false otherwise.

bool wxVariant::IsValueKindOf (const wxClassInfo * *type*) const

Returns true if the data is derived from the class described by *type*, false otherwise.

void wxVariant::MakeNull ( )

Makes the variant null by deleting the internal data.

wxString wxVariant::MakeString ( ) const

Makes a string representation of the variant value (for any type).

bool wxVariant::Member (const wxVariant & *value*) const

Returns true if *value* matches an element in the list.

void wxVariant::NullList ( )

Makes an empty list.

This differs from a null variant which has no data; a null list is of type list, but the number of elements in the list is zero.

double wxVariant::operator double ( ) const

Operators for implicit conversion, using appropriate getter member function.

long wxVariant::operator long ( ) const

Operators for implicit conversion, using appropriate getter member function.

void* wxVariant::operator void * ( ) const

Operator for implicit conversion to a pointer to a void, using GetVoidPtr().

char wxVariant::operator wxChar ( ) const

Operator for implicit conversion to a wxChar, using GetChar().

void* wxVariant::operator wxDateTime ( ) const

Operator for implicit conversion to a pointer to a wxDateTime, using GetDateTime().

wxLongLong wxVariant::operator wxLongLong ( ) const

Operators for implicit conversion, using appropriate getter member function.

wxString wxVariant::operator wxString ( ) const

Operator for implicit conversion to a string, using MakeString().

wxULongLong wxVariant::operator wxULongLong ( ) const

Operators for implicit conversion, using appropriate getter member function.

bool wxVariant::operator!= (const wxVariant & *value*) const

Inequality test operator.

bool wxVariant::operator!= (const wxString & *value*) const

Inequality test operator.

bool wxVariant::operator!= (const wxChar * *value*) const

Inequality test operator.

bool wxVariant::operator!= (wxChar *value*) const

Inequality test operator.

bool wxVariant::operator!= (long *value*) const

Inequality test operator.

bool wxVariant::operator!= (bool *value*) const

Inequality test operator.

bool wxVariant::operator!= (double *value*) const

Inequality test operator.

bool wxVariant::operator!= (wxLongLong *value*) const

Inequality test operator.

bool wxVariant::operator!= (wxULongLong *value*) const

Inequality test operator.

bool wxVariant::operator!= (void * *value*) const

Inequality test operator.

bool wxVariant::operator!= (wxObject * *value*) const

Inequality test operator.

bool wxVariant::operator!= (const wxVariantList & *value*) const

Inequality test operator.

bool wxVariant::operator!= (const wxArrayString & *value*) const

Inequality test operator.

bool wxVariant::operator!= (const wxDateTime & *value*) const

Inequality test operator.

void wxVariant::operator= (const wxVariant & *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (wxVariantData * *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (const wxString & *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (const wxChar * *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (wxChar *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (long *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (bool *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (double *value*)

Assignment operator, using reference counting if possible.

bool wxVariant::operator= (wxLongLong *value*) const

Assignment operator, using reference counting if possible.

bool wxVariant::operator= (wxULongLong *value*) const

Assignment operator, using reference counting if possible.

void wxVariant::operator= (void * *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (wxObject * *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (const wxVariantList & *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (const wxDateTime & *value*)

Assignment operator, using reference counting if possible.

void wxVariant::operator= (const wxArrayString & *value*)

Assignment operator, using reference counting if possible.

bool wxVariant::operator== (const wxVariant & *value*) const

Equality test operator.

bool wxVariant::operator== (const wxString & *value*) const

Equality test operator.

bool wxVariant::operator== (const wxChar * *value*) const

Equality test operator.

bool wxVariant::operator== (wxChar *value*) const

Equality test operator.

bool wxVariant::operator== (long *value*) const

Equality test operator.

bool wxVariant::operator== (bool *value*) const

Equality test operator.

bool wxVariant::operator== (double *value*) const

Equality test operator.

bool wxVariant::operator== (wxLongLong *value*) const

Equality test operator.

bool wxVariant::operator== (wxULongLong *value*) const

Equality test operator.

bool wxVariant::operator== (void * *value*) const

Equality test operator.

bool wxVariant::operator== (wxObject * *value*) const

Equality test operator.

bool wxVariant::operator== (const wxVariantList & *value*) const

Equality test operator.

bool wxVariant::operator== (const wxArrayString & *value*) const

Equality test operator.

bool wxVariant::operator== (const wxDateTime & *value*) const

Equality test operator.

wxVariant wxVariant::operator[] (size_t *idx*) const

Returns the value at *idx* (zero-based).

wxVariant& wxVariant::operator[] (size_t *idx*)

Returns a reference to the value at *idx* (zero-based).

This can be used to change the value at this index.

void wxVariant::SetData (wxVariantData * *data*)

Sets the internal variant data, deleting the existing data if there is any.

bool wxVariant::Unshare ( )

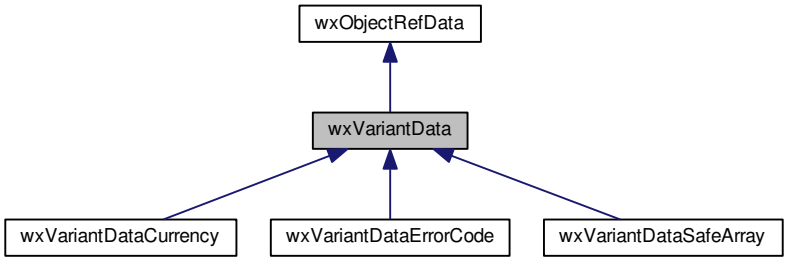
Makes sure that any data associated with this variant is not shared with other variants.

For this to work, wxVariantData::Clone() must be implemented for the data types you are working with. wxVariantData::Clone() is implemented for all the default data types.

21.827 wxVariantData Class Reference

`#include <wx/variant.h>`

Inheritance diagram for wxVariantData:



21.827.1 Detailed Description

The wxVariantData class is used to implement a new type for wxVariant.

Derive from wxVariantData, and override the pure virtual functions.

wxVariantData is reference counted, but you don't normally have to care about this, as wxVariant manages the count automatically. However, in case your application needs to take ownership of wxVariantData, be aware that the object is created with a reference count of 1, and passing it to wxVariant will not increase this. In other words, IncRef() needs to be called only if you both take ownership of wxVariantData and pass it to a wxVariant. Also note that the destructor is protected, so you can never explicitly delete a wxVariantData instance. Instead, DecRef() will delete the object automatically when the reference count reaches zero.

Library: wxBase

Category: Data Structures

See also

wxVariant, wxGetVariantCast()

Public Member Functions

- wxVariantData ()

  Default constructor.
- virtual wxVariantData * Clone () const

  This function can be overridden to clone the data.
- void DecRef ()

  Decreases reference count.
- virtual bool Eq (wxVariantData &data) const =0

  Returns true if this object is equal to data.
- virtual bool GetAny (wxAny *any) const

  Converts value to wxAny, if possible.
- virtual wxString GetType () const =0

  Returns the string type of the data.
- virtual wxClassInfo * GetValueClassInfo ()

If the data is a wxObject returns a pointer to the objects wxClassInfo structure, if the data isn't a wxObject the method returns NULL.

- void IncRef ()

  Increases reference count.

- virtual bool Read (istream &stream)

  Reads the data from stream.

- virtual bool Read (wxString &string)

  Reads the data from string.

- virtual bool Write (ostream &stream) const

  Writes the data to stream.

- virtual bool Write (wxString &string) const

  Writes the data to string.

21.827.2 Constructor & Destructor Documentation

wxVariantData::wxVariantData ( )

Default constructor.

21.827.3 Member Function Documentation

virtual wxVariantData* wxVariantData::Clone ( ) const `[virtual]`

This function can be overridden to clone the data.

You must implement this function in order for wxVariant::Unshare() to work for your data. This function is implemented for all built-in data types.

Reimplemented in wxVariantDataSafeArray, wxVariantDataErrorCode, and wxVariantDataCurrency.

void wxVariantData::DecRef ( )

Decreases reference count.

If the count reaches zero, the object is automatically deleted.

Note

The destructor of wxVariantData is protected, so delete cannot be used as normal. Instead, DecRef() should be called.

virtual bool wxVariantData::Eq (wxVariantData & *data*) const `[pure virtual]`

Returns true if this object is equal to *data*.

Implemented in wxVariantDataSafeArray, wxVariantDataErrorCode, and wxVariantDataCurrency.

virtual bool wxVariantData::GetAny (wxAny * *any*) const `[virtual]`

Converts value to wxAny, if possible.

Return true if successful.

virtual wxString wxVariantData::GetType ( ) const [pure virtual]

Returns the string type of the data.

Implemented in wxVariantDataSafeArray, wxVariantDataErrorCode, and wxVariantDataCurrency.

virtual wxClassInfo* wxVariantData::GetValueClassInfo ( ) [virtual]

If the data is a wxObject returns a pointer to the objects wxClassInfo structure, if the data isn't a wxObject the method returns NULL.

void wxVariantData::IncRef ( )

Increases reference count.

Note that initially wxVariantData has reference count of 1.

virtual bool wxVariantData::Read (istream & *stream*) [virtual]

Reads the data from *stream*.

virtual bool wxVariantData::Read (wxString & *string*) [virtual]

Reads the data from *string*.

virtual bool wxVariantData::Write (ostream & *stream*) const [virtual]

Writes the data to *stream*.

virtual bool wxVariantData::Write (wxString & *string*) const [virtual]

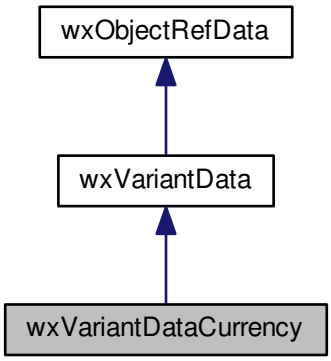
Writes the data to *string*.

Reimplemented in wxVariantDataSafeArray, wxVariantDataErrorCode, and wxVariantDataCurrency.

21.828 wxVariantDataCurrency Class Reference

```
#include <wx/msw/ole/automtn.h>
```

Inheritance diagram for wxVariantDataCurrency:



21.828.1 Detailed Description

This class represents a thin wrapper for Microsoft Windows CURRENCY type.

It is used for converting between wxVariant and OLE VARIANT with type set to VT_CURRENCY. When wxVariant stores wxVariantDataCurrency, it returns "currency" as its type.

An example of setting and getting CURRENCY value to and from wxVariant:

```
CURRENCY cy;
wxVariant variant;

// set wxVariant to currency type
if ( SUCCEEDED(VarCyFromR8(123.45, &cy)) )  // set cy to 123.45
{
    variant.SetData(new wxVariantDataCurrency(cy));

    // or instead of the line above you could write:
    // wxVariantDataCurrency wxCy;
    // wxCy.SetValue(cy);
    // variant.SetData(wxCy.Clone());
}

// get CURRENCY value from wxVariant
if ( variant.GetType() == "currency" )
{
    wxVariantDataCurrency*
        wxCy = wxDynamicCastVariantData(variant.GetData(),
      wxVariantDataCurrency);
    cy = wxCy->GetValue();
}
```

Availability: only available for the wxMSW port.

Since

2.9.5

Library: wxCore

Category: Data Structures

See also

wxAutomationObject, wxVariant, wxVariantData, wxVariantDataErrorCode

Include file:

```
#include <wx/msw/ole/oleutils.h>
```

Public Member Functions

- wxVariantDataCurrency ()

  Default constructor initializes the object to 0.0.
- wxVariantDataCurrency (CURRENCY value)

  Constructor from CURRENCY.
- CURRENCY GetValue () const

  Returns the stored CURRENCY value.
- void SetValue (CURRENCY value)

  Sets the stored value to value.
- virtual bool Eq (wxVariantData &data) const

  Returns true if data is of wxVariantDataCurency type and contains the same CURRENCY value.
- virtual bool Write (wxString &str) const

  Fills the provided string with the textual representation of this object.
- wxVariantData * Clone () const

  Returns a copy of itself.
- virtual wxString GetType () const

  Returns "currency".
- virtual bool GetAsAny (wxAny *any) const

  Converts the value of this object to wxAny.

21.828.2 Constructor & Destructor Documentation

wxVariantDataCurrency::wxVariantDataCurrency ( )

Default constructor initializes the object to 0.0.

wxVariantDataCurrency::wxVariantDataCurrency (CURRENCY *value*)

Constructor from CURRENCY.

21.828.3 Member Function Documentation

wxVariantData* wxVariantDataCurrency::Clone ( ) const `[virtual]`

Returns a copy of itself.

Reimplemented from wxVariantData.

virtual bool wxVariantDataCurrency::Eq (wxVariantData & *data*) const `[virtual]`

Returns true if *data* is of wxVariantDataCurency type and contains the same CURRENCY value.

Implements wxVariantData.

virtual bool wxVariantDataCurrency::GetAsAny (wxAny * *any*) const `[virtual]`

Converts the value of this object to wxAny.

virtual wxString wxVariantDataCurrency::GetType ( ) const `[virtual]`

Returns "currency".

Implements wxVariantData.

CURRENCY wxVariantDataCurrency::GetValue ( ) const

Returns the stored CURRENCY value.

void wxVariantDataCurrency::SetValue (CURRENCY *value*)

Sets the stored value to *value*.

virtual bool wxVariantDataCurrency::Write (wxString & *str*) const `[virtual]`

Fills the provided string with the textual representation of this object.

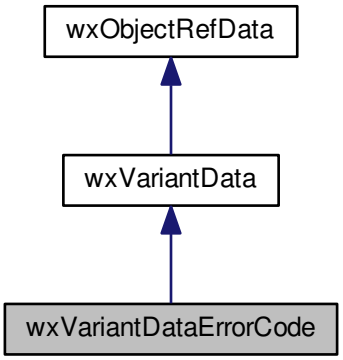
The implementation of this method using `VarBstrFromCy()` Windows API function with LOCALE_USER_DEFAULT.

Reimplemented from wxVariantData.

21.829 wxVariantDataErrorCode Class Reference

`#include <wx/msw/ole/automtn.h>`

Inheritance diagram for wxVariantDataErrorCode:



21.829.1 Detailed Description

This class represents a thin wrapper for Microsoft Windows SCODE type (which is the same as HRESULT).

It is used for converting between a wxVariant and OLE VARIANT with type set to VT_ERROR. When wxVariant stores wxVariantDataErrorCode, it returns "errorcode" as its type. This class can be used for returning error codes of automation calls or exchanging values with other applications: e.g. Microsoft Excel returns VARIANTs with VT_ERROR type for cell values with errors (one of XlCVError constants, displayed as e.g. "#DIV/0!" or "#REF!" there) etc. See wxVariantDataCurrency for an example of how to exchange values between wxVariant and a native type not directly supported by it.

Availability: only available for the wxMSW port.

Since

2.9.5

Library: wxCore

Category: Data Structures

See also

wxAutomationObject, wxVariant, wxVariantData, wxVariantDataCurrency

Include file:

```
#include <wx/msw/ole/oleutils.h>
```

Public Member Functions

- wxVariantDataErrorCode (SCODE value=S_OK)

  Constructor initializes the object to value or S_OK if no value was passed.
- SCODE GetValue () const

  Returns the stored SCODE value.
- void SetValue (SCODE value)

  Set the stored value to value.
- virtual bool Eq (wxVariantData &data) const

  Returns true if data is of wxVariantDataErrorCode type and contains the same SCODE value.
- virtual bool Write (wxString &str) const

  Fills the provided string with the textual representation of this object.
- wxVariantData * Clone () const

  Returns a copy of itself.
- virtual wxString GetType () const

  Returns "errorcode".
- virtual bool GetAsAny (wxAny *any) const

  Converts the value of this object to wxAny.

21.829.2 Constructor & Destructor Documentation

wxVariantDataErrorCode::wxVariantDataErrorCode (SCODE *value* = `S_OK`)

Constructor initializes the object to *value* or S_OK if no value was passed.

21.829.3 Member Function Documentation

wxVariantData* wxVariantDataErrorCode::Clone ( ) const [virtual]

Returns a copy of itself.

Reimplemented from wxVariantData.

virtual bool wxVariantDataErrorCode::Eq (wxVariantData & *data*) const [virtual]

Returns true if *data* is of wxVariantDataErrorCode type and contains the same SCODE value.

Implements wxVariantData.

virtual bool wxVariantDataErrorCode::GetAsAny (wxAny * *any*) const [virtual]

Converts the value of this object to wxAny.

virtual wxString wxVariantDataErrorCode::GetType ( ) const [inline], [virtual]

Returns "errorcode".

Implements wxVariantData.

SCODE wxVariantDataErrorCode::GetValue ( ) const

Returns the stored SCODE value.

void wxVariantDataErrorCode::SetValue (SCODE *value*)

Set the stored value to *value*.

virtual bool wxVariantDataErrorCode::Write (wxString & *str*) const [virtual]

Fills the provided string with the textual representation of this object.

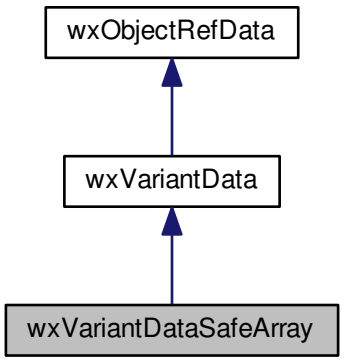
The error code is just a number, so it's output as such.

Reimplemented from wxVariantData.

21.830 wxVariantDataSafeArray Class Reference

`#include <wx/msw/ole/automtn.h>`

Inheritance diagram for wxVariantDataSafeArray:



21.830.1 Detailed Description

This class represents a thin wrapper for Microsoft Windows SAFEARRAY type.

It is used for converting between wxVariant and OLE VARIANT with type set to VT_ARRAY, which has more than one dimension. When wxVariant stores wxVariantDataSafeArray, it returns "safearray" as its type.

wxVariantDataSafeArray does NOT manage the SAFEARRAY it points to. If you want to pass it to a wxAutomationObject as a parameter:

1. Assign a SAFEARRAY pointer to it and store it in a wxVariant.
2. Call the wxAutomationObject method (CallMethod(), SetProperty() or Invoke())
3. wxAutomationObject will destroy the array after the approapriate automation call.

An example of creating a 2-dimensional SAFEARRAY containing VARIANTs and storing it in a wxVariant

```
SAFEARRAYBOUND bounds[2]; // 2 dimensions
wxSafeArray<VT_VARIANT> safeArray;
unsigned rowCount = 1000;
unsigned colCount = 20;

bounds[0].lLbound = 0; // elements start at 0
bounds[0].cElements = rowCount;
bounds[1].lLbound = 0; // elements start at 0
bounds[1].cElements = colCount;

if ( !safeArray.Create(bounds, 2) )
    return false;

long indices[2];

for ( unsigned row = 0; row < rowCount; row++ )
{
    indices[0] = row;
    for ( unsigned col = 0; col < colCount; col++ )
    {
        indices[1] = col;
        if ( !safeArray.SetElement(indices, wxString::Format("R%u C%u", row+1, col+1)) )
           return false;
    }
}
range.PutProperty("Value", wxVariant(new wxVariantDataSafeArray(safeArray.
      Detach())));
```

If you you received wxVariantDataSafeArray as a result of wxAutomationObject method call: (1) Get the data out of the array. (2) Destroy the array.

```
wxVariant result;
result = range.GetProperty("Value");
if ( result.GetType() == "safearray" )
{
    wxSafeArray<VT_VARIANT> safeArray;
    wxVariantDataSafeArray* const
        sa = wxStaticCastVariantData(variant.GetData(),
      wxVariantDataSafeArray);

    if ( !safeArray.Attach(sa.GetValue() )
    {
        if ( !safeArray.HasArray() )
            SafeArrayDestroy(sa.GetValue()); // we have to dispose the SAFEARRAY ourselves
        return false;
    }

    // get the data from the SAFEARRAY using wxSafeArray::GetElement()
    // SAFEARRAY will be disposed by safeArray's dtor
}
```

Availability: only available for the wxMSW port.

Since

2.9.5

Library: wxCore

Category: Data Structures

See also

wxAutomationObject, wxVariant, wxVariantData, wxVariantDataErrorCode

Include file:

```
#include <wx/msw/ole/oleutils.h>
```

Public Member Functions

- wxVariantDataSafeArray (SAFEARRAY *value=NULL)

  Constructor initializes the object to value.
- SAFEARRAY * GetValue () const

  Returns the stored array.
- void SetValue (SAFEARRAY *value)

  Set the stored array.
- virtual bool Eq (wxVariantData &data) const

  Returns true if data is of wxVariantDataSafeArray type and contains the same SAFEARRAY value.*
- virtual bool Write (wxString &str) const

  Fills the provided string with the textual representation of this object.
- wxVariantData * Clone () const

  Returns a copy of itself.
- virtual wxString GetType () const

  Returns "safearray".
- virtual bool GetAsAny (wxAny *any) const

  Converts the value of this object to wxAny.

21.830.2 Constructor & Destructor Documentation

wxVariantDataSafeArray::wxVariantDataSafeArray (SAFEARRAY * *value* = NULL) [explicit]

Constructor initializes the object to *value*.

21.830.3 Member Function Documentation

wxVariantData* wxVariantDataSafeArray::Clone ( ) const [virtual]

Returns a copy of itself.

Reimplemented from wxVariantData.

virtual bool wxVariantDataSafeArray::Eq (wxVariantData & *data*) const [virtual]

Returns true if *data* is of wxVariantDataSafeArray type and contains the same SAFEARRAY* value.

Implements wxVariantData.

virtual bool wxVariantDataSafeArray::GetAsAny (wxAny * *any*) const [virtual]

Converts the value of this object to wxAny.

virtual wxString wxVariantDataSafeArray::GetType ( ) const [virtual]

Returns "safearray".

Implements wxVariantData.

SAFEARRAY* wxVariantDataSafeArray::GetValue ( ) const

Returns the stored array.

void wxVariantDataSafeArray::SetValue (SAFEARRAY * *value*)

Set the stored array.

virtual bool wxVariantDataSafeArray::Write (wxString & *str*) const [virtual]

Fills the provided string with the textual representation of this object.

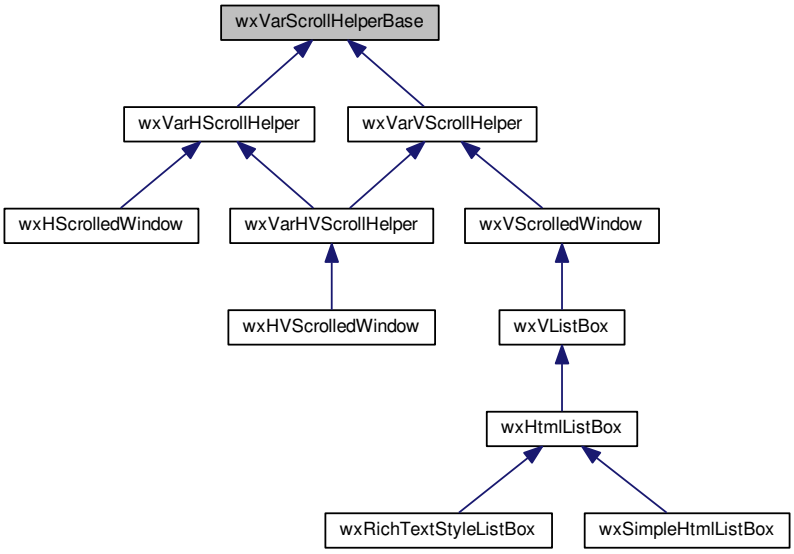
Only the address of SAFEARRAY pointer is output.

Reimplemented from wxVariantData.

21.831 wxVarScrollHelperBase Class Reference

```
#include <wx/vscroll.h>
```

Inheritance diagram for wxVarScrollHelperBase:



21.831.1 Detailed Description

This class provides all common base functionality for scroll calculations shared among all variable scrolled window implementations as well as automatic scrollbar functionality, saved scroll positions, controlling target windows to be scrolled, as well as defining all required virtual functions that need to be implemented for any orientation specific work.

Documentation of this class is provided specifically for referencing use of the functions provided by this class for use with the variable scrolled windows that derive from here. You will likely want to derive your window from one of the already implemented variable scrolled windows rather than from wxVarScrollHelperBase directly.

Library: wxCore

Category: Miscellaneous Windows

See also

wxHScrolledWindow, wxHVScrolledWindow, wxVScrolledWindow

Public Member Functions

- wxVarScrollHelperBase (wxWindow *winToScroll)

  Constructor taking the target window to be scrolled by this helper class.
- virtual ~wxVarScrollHelperBase ()

  Virtual destructor for detaching scroll event handlers attached with this helper class.
- int CalcScrolledPosition (int coord) const

  Translates the logical coordinate given to the current device coordinate.
- int CalcUnscrolledPosition (int coord) const

  Translates the device coordinate given to the corresponding logical coordinate.

- void EnablePhysicalScrolling (bool scrolling=true)

  With physical scrolling on (when this is true), the device origin is changed properly when a wxPaintDC is prepared, children are actually moved and laid out properly, and the contents of the window (pixels) are actually moved.
- virtual int GetNonOrientationTargetSize () const =0

  This function needs to be overridden in the in the derived class to return the window size with respect to the opposing orientation.
- virtual wxOrientation GetOrientation () const =0

  This function need to be overridden to return the orientation that this helper is working with, either `wxHORIZONTAL` or `wxVERTICAL`.
- virtual int GetOrientationTargetSize () const =0

  This function needs to be overridden in the in the derived class to return the window size with respect to the orientation this helper is working with.
- virtual wxWindow * GetTargetWindow () const

  This function will return the target window this helper class is currently scrolling.
- size_t GetVisibleBegin () const

  Returns the index of the first visible unit based on the scroll position.
- size_t GetVisibleEnd () const

  Returns the index of the last visible unit based on the scroll position.
- bool IsVisible (size_t unit) const

  Returns true if the given scroll unit is currently visible (even if only partially visible) or false otherwise.
- virtual void RefreshAll ()

  Recalculate all parameters and repaint all units.
- virtual void SetTargetWindow (wxWindow *target)

  Normally the window will scroll itself, but in some rare occasions you might want it to scroll (part of) another window (e.g.
- virtual void UpdateScrollbar ()

  Update the thumb size shown by the scrollbar.
- int VirtualHitTest (wxCoord coord) const

  Returns the virtual scroll unit under the device unit given accounting for scroll position or `wxNOT_FOUND` if none (i.e.

Protected Member Functions

- virtual void OnGetUnitsSizeHint (size_t unitMin, size_t unitMax) const

  This function doesn't have to be overridden but it may be useful to do so if calculating the units' sizes is a relatively expensive operation as it gives your code a chance to calculate several of them at once and cache the result if necessary.
- virtual wxCoord EstimateTotalSize () const

  When the number of scroll units change, we try to estimate the total size of all units when the full window size is needed (i.e.
- virtual wxCoord OnGetUnitSize (size_t unit) const =0

  This function must be overridden in the derived class, and should return the size of the given unit in pixels.

21.831.2 Constructor & Destructor Documentation

wxVarScrollHelperBase::wxVarScrollHelperBase (wxWindow * *winToScroll*)

Constructor taking the target window to be scrolled by this helper class.

This will attach scroll event handlers to the target window to catch and handle scroll events appropriately.

virtual wxVarScrollHelperBase::~wxVarScrollHelperBase ( ) `[virtual]`

Virtual destructor for detaching scroll event handlers attached with this helper class.

21.831.3 Member Function Documentation

int wxVarScrollHelperBase::CalcScrolledPosition (int *coord*) const

Translates the logical coordinate given to the current device coordinate.

For example, if the window is scrolled 10 units and each scroll unit represents 10 device units (which may not be the case since this class allows for variable scroll unit sizes), a call to this function with a coordinate of 15 will return -85.

See also

CalcUnscrolledPosition()

int wxVarScrollHelperBase::CalcUnscrolledPosition (int *coord*) const

Translates the device coordinate given to the corresponding logical coordinate.

For example, if the window is scrolled 10 units and each scroll unit represents 10 device units (which may not be the case since this class allows for variable scroll unit sizes), a call to this function with a coordinate of 15 will return 115.

See also

CalcScrolledPosition()

void wxVarScrollHelperBase::EnablePhysicalScrolling (bool *scrolling* = `true`)

With physical scrolling on (when this is true), the device origin is changed properly when a wxPaintDC is prepared, children are actually moved and laid out properly, and the contents of the window (pixels) are actually moved.

When this is false, you are responsible for repainting any invalidated areas of the window yourself to account for the new scroll position.

virtual wxCoord wxVarScrollHelperBase::EstimateTotalSize ( ) const `[protected], [virtual]`

When the number of scroll units change, we try to estimate the total size of all units when the full window size is needed (i.e.

to calculate the scrollbar thumb size). This is a rather expensive operation in terms of unit access, so if the user code may estimate the average size better or faster than we do, it should override this function to implement its own logic. This function should return the best guess for the total virtual window size.

Note

Although returning a totally wrong value would still work, it risks resulting in very strange scrollbar behaviour so this function should really try to make the best guess possible.

virtual int wxVarScrollHelperBase::GetNonOrientationTargetSize ( ) const `[pure virtual]`

This function needs to be overridden in the in the derived class to return the window size with respect to the opposing orientation.

If this is a vertical scrolled window, it should return the height.

See also

GetOrientationTargetSize()

virtual wxOrientation wxVarScrollHelperBase::GetOrientation ( ) const `[pure virtual]`

This function need to be overridden to return the orientation that this helper is working with, either `wxHORIZONTAL` or `wxVERTICAL`.

virtual int wxVarScrollHelperBase::GetOrientationTargetSize ( ) const `[pure virtual]`

This function needs to be overridden in the in the derived class to return the window size with respect to the orientation this helper is working with.

If this is a vertical scrolled window, it should return the width.

See also

GetNonOrientationTargetSize()

virtual wxWindow* wxVarScrollHelperBase::GetTargetWindow ( ) const `[virtual]`

This function will return the target window this helper class is currently scrolling.

See also

SetTargetWindow()

size_t wxVarScrollHelperBase::GetVisibleBegin ( ) const

Returns the index of the first visible unit based on the scroll position.

size_t wxVarScrollHelperBase::GetVisibleEnd ( ) const

Returns the index of the last visible unit based on the scroll position.

This includes the last unit even if it is only partially visible.

bool wxVarScrollHelperBase::IsVisible (size_t *unit*) const

Returns true if the given scroll unit is currently visible (even if only partially visible) or false otherwise.

virtual wxCoord wxVarScrollHelperBase::OnGetUnitSize (size_t *unit*) const `[protected], [pure virtual]`

This function must be overridden in the derived class, and should return the size of the given unit in pixels.

virtual void wxVarScrollHelperBase::OnGetUnitsSizeHint (size_t *unitMin*, size_t *unitMax*) const `[protected], [virtual]`

This function doesn't have to be overridden but it may be useful to do so if calculating the units' sizes is a relatively expensive operation as it gives your code a chance to calculate several of them at once and cache the result if necessary.

OnGetUnitsSizeHint() is normally called just before OnGetUnitSize() but you shouldn't rely on the latter being called for all units in the interval specified here. It is also possible that OnGetUnitSize() will be called for units outside of this interval, so this is really just a hint, not a promise.

Finally, note that *unitMin* is inclusive, while *unitMax* is exclusive.

virtual void wxVarScrollHelperBase::RefreshAll ( ) [virtual]

Recalculate all parameters and repaint all units.

virtual void wxVarScrollHelperBase::SetTargetWindow (wxWindow * *target*) [virtual]

Normally the window will scroll itself, but in some rare occasions you might want it to scroll (part of) another window (e.g.

a child of it in order to scroll only a portion the area between the scrollbars like a spreadsheet where only the cell area will move).

See also

GetTargetWindow()

virtual void wxVarScrollHelperBase::UpdateScrollbar ( ) [virtual]

Update the thumb size shown by the scrollbar.

int wxVarScrollHelperBase::VirtualHitTest (wxCoord *coord*) const

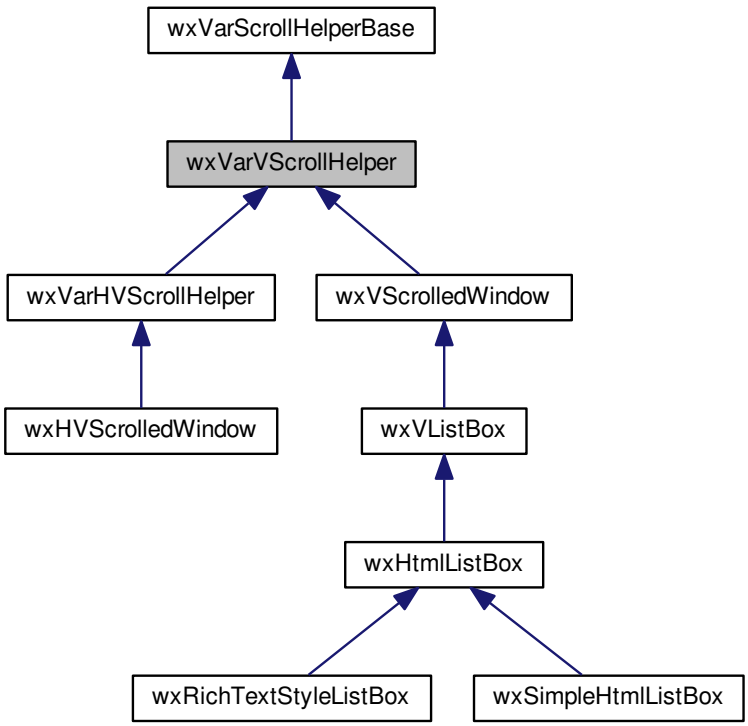
Returns the virtual scroll unit under the device unit given accounting for scroll position or `wxNOT_FOUND` if none (i.e.

if it is below the last item).

21.832 wxVarVScrollHelper Class Reference

`#include <wx/vscroll.h>`

Inheritance diagram for wxVarVScrollHelper:



21.832.1 Detailed Description

This class provides functions wrapping the wxVarScrollHelperBase class, targeted for vertical-specific scrolling.

Like wxVarScrollHelperBase, this class is mostly only useful to those classes built into wxWidgets deriving from here, and this documentation is mostly only provided for referencing the functions provided by this class. You will likely want to derive your window from wxVScrolledWindow rather than from here directly.

Library: wxCore

Category: Miscellaneous Windows

See also

wxHScrolledWindow, wxHVScrolledWindow, wxVScrolledWindow

Public Member Functions

- wxVarVScrollHelper (wxWindow *winToScroll)

  Constructor taking the target window to be scrolled by this helper class.
- size_t GetRowCount () const

  Returns the number of rows the target window contains.

- size_t GetVisibleRowsBegin () const

  Returns the index of the first visible row based on the scroll position.
- size_t GetVisibleRowsEnd () const

  Returns the index of the last visible row based on the scroll position.
- bool IsRowVisible (size_t row) const

  Returns true if the given row is currently visible (even if only partially visible) or false otherwise.
- virtual void RefreshRow (size_t row)

  Triggers a refresh for just the given row's area of the window if it's visible.
- virtual void RefreshRows (size_t from, size_t to)

  Triggers a refresh for the area between the specified range of rows given (inclusively).
- virtual bool ScrollRowPages (int pages)

  Scroll by the specified number of pages which may be positive (to scroll down) or negative (to scroll up).
- virtual bool ScrollRows (int rows)

  Scroll by the specified number of rows which may be positive (to scroll down) or negative (to scroll up).
- bool ScrollToRow (size_t row)

  Scroll to the specified row.
- void SetRowCount (size_t rowCount)

  Set the number of rows the window contains.

Protected Member Functions

- virtual void OnGetRowsHeightHint (size_t rowMin, size_t rowMax) const

  This function doesn't have to be overridden but it may be useful to do so if calculating the rows' sizes is a relatively expensive operation as it gives your code a chance to calculate several of them at once and cache the result if necessary.
- virtual wxCoord EstimateTotalHeight () const

  This class forwards calls from EstimateTotalSize() to this function so derived classes can override either just the height or the width estimation, or just estimate both differently if desired in any wxHVScrolledWindow derived class.
- virtual wxCoord OnGetRowHeight (size_t row) const =0

  This function must be overridden in the derived class, and should return the height of the given row in pixels.

21.832.2 Constructor & Destructor Documentation

wxVarVScrollHelper::wxVarVScrollHelper (wxWindow * *winToScroll*)

Constructor taking the target window to be scrolled by this helper class.

This will attach scroll event handlers to the target window to catch and handle scroll events appropriately.

21.832.3 Member Function Documentation

virtual wxCoord wxVarVScrollHelper::EstimateTotalHeight ( ) const `[protected], [virtual]`

This class forwards calls from EstimateTotalSize() to this function so derived classes can override either just the height or the width estimation, or just estimate both differently if desired in any wxHVScrolledWindow derived class.

Note

This function will not be called if EstimateTotalSize() is overridden in your derived class.

size_t wxVarVScrollHelper::GetRowCount ( ) const

Returns the number of rows the target window contains.

See also

SetRowCount()

size_t wxVarVScrollHelper::GetVisibleRowsBegin ( ) const

Returns the index of the first visible row based on the scroll position.

size_t wxVarVScrollHelper::GetVisibleRowsEnd ( ) const

Returns the index of the last visible row based on the scroll position.

This includes the last row even if it is only partially visible.

bool wxVarVScrollHelper::IsRowVisible (size_t *row*) const

Returns true if the given row is currently visible (even if only partially visible) or false otherwise.

virtual wxCoord wxVarVScrollHelper::OnGetRowHeight (size_t *row*) const `[protected], [pure virtual]`

This function must be overridden in the derived class, and should return the height of the given row in pixels.

virtual void wxVarVScrollHelper::OnGetRowsHeightHint (size_t *rowMin,* size_t *rowMax*) const `[protected], [virtual]`

This function doesn't have to be overridden but it may be useful to do so if calculating the rows' sizes is a relatively expensive operation as it gives your code a chance to calculate several of them at once and cache the result if necessary.

OnGetRowsHeightHint() is normally called just before OnGetRowHeight() but you shouldn't rely on the latter being called for all rows in the interval specified here. It is also possible that OnGetRowHeight() will be called for units outside of this interval, so this is really just a hint, not a promise.

Finally, note that *rowMin* is inclusive, while *rowMax* is exclusive.

virtual void wxVarVScrollHelper::RefreshRow (size_t *row*) `[virtual]`

Triggers a refresh for just the given row's area of the window if it's visible.

virtual void wxVarVScrollHelper::RefreshRows (size_t *from,* size_t *to*) `[virtual]`

Triggers a refresh for the area between the specified range of rows given (inclusively).

virtual bool wxVarVScrollHelper::ScrollRowPages (int *pages*) `[virtual]`

Scroll by the specified number of pages which may be positive (to scroll down) or negative (to scroll up).

virtual bool wxVarVScrollHelper::ScrollRows (int *rows*) `[virtual]`

Scroll by the specified number of rows which may be positive (to scroll down) or negative (to scroll up).

Returns

true if the window was scrolled, false otherwise (for example, if we're trying to scroll down but we are already showing the last row).

bool wxVarVScrollHelper::ScrollToRow (size_t *row*)

Scroll to the specified row.

It will become the first visible row in the window.

Returns

true if we scrolled the window, false if nothing was done.

void wxVarVScrollHelper::SetRowCount (size_t *rowCount*)

Set the number of rows the window contains.

The derived class must provide the heights for all rows with indices up to the one given here in it's OnGetRowHeight() implementation.

See also

GetRowCount()

21.833 wxVector< T > Class Template Reference

`#include <wx/vector.h>`

21.833.1 Detailed Description

template<typename T>class wxVector< T >

wxVector<T> is a template class which implements most of the `std::vector` class and can be used like it.

If wxWidgets is compiled in STL mode, wxVector will just be a typedef to `std::vector`. Just like for `std::vector`, objects stored in wxVector<T> need to be *assignable* but don't have to be *"default constructible"*.

Please refer to the STL documentation for further information.

Library: None; this class implementation is entirely header-based.

Category: Containers

See also

Container Classes, wxList<T>, wxArray<T>, wxVectorSort<T>

Public Types

- typedef size_t size_type
- typedef size_t difference_type
- typedef T value_type
- typedef value_type * pointer
- typedef value_type * iterator
- typedef const value_type * const_iterator
- typedef value_type & reference

Public Member Functions

- wxVector ()

  Constructor.
- wxVector (size_type size)

  Constructor initializing the vector with the given number of default-constructed objects.
- wxVector (size_type size, const value_type &value)

  Constructor initializing the vector with the given number of copies of the given object.
- template<class InputIterator >
  wxVector (InputIterator first, InputIterator last)

  Constructor initializing the vector with the elements in the given range.
- wxVector (const wxVector< T > &c)

  Copy constructor.
- ~wxVector ()

  Destructor.
- void assign (size_type n, const value_type &v)

  Resizes the vector to n and assigns v to all elements.
- template<class InputIterator >
  void assign (InputIterator first, InputIterator last)

  Assigns the elements in the given range to the vector.
- const value_type & at (size_type idx) const

  Returns item at position idx.
- value_type & at (size_type idx)

  Returns item at position idx.
- const value_type & back () const

  Return the last item.
- value_type & back ()

  Return the last item.
- const_iterator begin () const

  Return iterator to beginning of the vector.
- iterator begin ()

  Return iterator to beginning of the vector.
- reverse_iterator rbegin ()

  Return reverse iterator to end of the vector.
- reverse_iterator rend ()

  Return reverse iterator to beginning of the vector.
- size_type capacity () const

  Returns vector's current capacity, i.e. how much memory is allocated.
- void clear ()

  Clears the vector.
- bool empty () const

Returns true if the vector is empty.

- const_iterator end () const

  Returns iterator to the end of the vector.

- iterator end ()

  Returns iterator to the end of the vector.

- iterator erase (iterator it)

  Erase item pointed to by iterator it.

- iterator erase (iterator first, iterator last)

  Erase items in the range first to last (last is not erased).

- const value_type & front () const

  Returns the first item.

- value_type & front ()

  Returns the first item.

- iterator insert (iterator it, const value_type &v=value_type())

  Insert item v at given position it.

- wxVector & operator= (const wxVector &vb)

  Assignment operator.

- const value_type & operator[] (size_type idx) const

  Returns item at position idx.

- value_type & operator[] (size_type idx)

  Returns item at position idx.

- void pop_back ()

  Removes the last item.

- void push_back (const value_type &v)

  Adds an item to the end of the vector.

- void reserve (size_type n)

  Reserves memory for at least n items.

- size_type size () const

  Returns the size of the vector.

- void swap (wxVector &v)

  Efficiently exchanges contents of this vector with another one.

- void resize (size_type n)

  Makes the vector of size n.

- void resize (size_type n, const value_type &v)

  Makes the vector of size n.

21.833.2 Member Typedef Documentation

template<typename T > typedef const value_type* wxVector< T >::const_iterator

template<typename T > typedef size_t wxVector< T >::difference_type

template<typename T > typedef value_type* wxVector< T >::iterator

template<typename T > typedef value_type* wxVector< T >::pointer

template<typename T > typedef value_type& wxVector< T >::reference

template<typename T > typedef size_t wxVector< T >::size_type

template<typename T > typedef T wxVector< T >::value_type

21.833.3 Constructor & Destructor Documentation

template<typename T > wxVector< T >::wxVector ( )

Constructor.

template<typename T > wxVector< T >::wxVector (size_type *size*)

Constructor initializing the vector with the given number of default-constructed objects.

template<typename T > wxVector< T >::wxVector (size_type *size*, const value_type & *value*)

Constructor initializing the vector with the given number of copies of the given object.

template<typename T > template<class InputIterator > wxVector< T >::wxVector (InputIterator *first*, InputIterator *last*)

Constructor initializing the vector with the elements in the given range.

The *InputIterator* template parameter must be an input iterator type. This constructor adds all elements from *first* until, not not including, *last* to the vector.

Since

2.9.5

template<typename T > wxVector< T >::wxVector (const wxVector< T > & *c*)

Copy constructor.

template<typename T > wxVector< T >::~wxVector ( )

Destructor.

21.833.4 Member Function Documentation

template<typename T > void wxVector< T >::assign (size_type *n*, const value_type & *v*)

Resizes the vector to *n* and assigns *v* to all elements.

See also

resize()

Since

2.9.5

template<typename T > template<class InputIterator > void wxVector< T >::assign (InputIterator *first,* InputIterator *last*)

Assigns the elements in the given range to the vector.

The *InputIterator* template parameter must be an input iterator type. This method clears the vector and then adds all elements from *first* until, not not including, *last* to it.

Since

2.9.5

template<typename T > const value_type& wxVector< T >::at (size_type *idx*) const

Returns item at position *idx*.

template<typename T > value_type& wxVector< T >::at (size_type *idx*)

Returns item at position *idx*.

template<typename T > const value_type& wxVector< T >::back ( ) const

Return the last item.

template<typename T > value_type& wxVector< T >::back ( )

Return the last item.

template<typename T > const_iterator wxVector< T >::begin ( ) const

Return iterator to beginning of the vector.

template<typename T > iterator wxVector< T >::begin ( )

Return iterator to beginning of the vector.

template<typename T > size_type wxVector< T >::capacity ( ) const

Returns vector's current capacity, i.e. how much memory is allocated.

See also

reserve()

template<typename T > void wxVector< T >::clear ( )

Clears the vector.

template<typename T > bool wxVector< T >::empty ( ) const

Returns true if the vector is empty.

template<typename T > const_iterator wxVector< T >::end ( ) const

Returns iterator to the end of the vector.

template<typename T > iterator wxVector< T >::end ( )

Returns iterator to the end of the vector.

template<typename T > iterator wxVector< T >::erase (iterator *it*)

Erase item pointed to by iterator *it*.

Returns

Iterator pointing to the item immediately after the erased one.

template<typename T > iterator wxVector< T >::erase (iterator *first,* iterator *last*)

Erase items in the range *first* to *last* (*last* is not erased).

Returns

Iterator pointing to the item immediately after the erased range.

template<typename T > const value_type& wxVector< T >::front ( ) const

Returns the first item.

template<typename T > value_type& wxVector< T >::front ( )

Returns the first item.

template<typename T > iterator wxVector< T >::insert (iterator *it,* const value_type & *v* = value_type ( ))

Insert item *v* at given position *it*.

Returns

Iterator for the inserted item.

template<typename T > wxVector& wxVector< T >::operator= (const wxVector< T > & *vb*)

Assignment operator.

template<typename T > const value_type& wxVector< T >::operator[] (size_type *idx*) const

Returns item at position *idx*.

template<typename T > value_type& wxVector< T >::operator[] (size_type *idx*)

Returns item at position *idx*.

template<typename T > void wxVector< T >::pop_back ( )

Removes the last item.

template<typename T > void wxVector< T >::push_back (const value_type & *v*)

Adds an item to the end of the vector.

template<typename T > reverse_iterator wxVector< T >::rbegin ( )

Return reverse iterator to end of the vector.

template<typename T > reverse_iterator wxVector< T >::rend ( )

Return reverse iterator to beginning of the vector.

template<typename T > void wxVector< T >::reserve (size_type *n*)

Reserves memory for at least *n* items.

See also

capacity()

template<typename T > void wxVector< T >::resize (size_type *n*)

Makes the vector of size *n*.

If *n* is less than the current size(), the elements at the end of the vector are erased. If it is greater, then the vector is completed with either the copies of the given object *v* or `value_type()` objects until it becomes of size *n*.

template<typename T > void wxVector< T >::resize (size_type *n*, const value_type & *v*)

Makes the vector of size *n*.

If *n* is less than the current size(), the elements at the end of the vector are erased. If it is greater, then the vector is completed with either the copies of the given object *v* or `value_type()` objects until it becomes of size *n*.

template<typename T > size_type wxVector< T >::size ( ) const

Returns the size of the vector.

template<typename T > void wxVector< T >::swap (wxVector< T > & *v*)

Efficiently exchanges contents of this vector with another one.

After the execution of this function the contents of this vector is equal to the original contents of *v* and the contents of *v* becomes the original contents of this vector without copying the data.

Since

2.9.1

21.834 wxVersionInfo Class Reference

```
#include <wx/versioninfo.h>
```

21.834.1 Detailed Description

wxVersionInfo contains version information.

This class is used by wxWidgets to provide version information about the libraries it uses and itself, but you can also apply it in user space, to provide version information about your own libraries, or other libraries that you use.

Library: wxBase

Category: Data Structures

Since

2.9.2

Public Member Functions

- wxVersionInfo (const wxString &name=wxString(), int major=0, int minor=0, int micro=0, const wxString &description=wxString(), const wxString &copyright=wxString())
  Constructor.
- const wxString & GetName () const
  Get the name of the object (library).
- int GetMajor () const
  Get the major version number.
- int GetMinor () const
  Get the minor version number.
- int GetMicro () const
  Get the micro version, or release number.
- wxString ToString () const
  Get the string representation of this version object.
- wxString GetVersionString () const
  Get the string representation.
- bool HasDescription () const
  Return true if a description string has been specified.
- const wxString & GetDescription ()
  Get the description string.
- bool HasCopyright () const
  Returns true if a copyright string has been specified.
- const wxString & GetCopyright () const
  Get the copyright string.

21.834.2 Constructor & Destructor Documentation

wxVersionInfo::wxVersionInfo (const wxString & *name* = wxString (), int *major* = 0, int *minor* = 0, int *micro* = 0, const wxString & *description* = wxString (), const wxString & *copyright* = wxString ())

Constructor.

The version information objects need to be initialized with this constructor and are immutable once they are created.

Parameters

name	The name of the library or other entity that this object pertains to.
major	The major version component.
minor	The minor version component.
micro	The micro version component, 0 by default.
description	Free form description of this version, none by default.
copyright	Copyright string, none by default.

21.834.3 Member Function Documentation

const wxString& wxVersionInfo::GetCopyright ( ) const

Get the copyright string.

The copyright string may be empty.

Returns

The copyright string.

const wxString& wxVersionInfo::GetDescription ( )

Get the description string.

The description may be empty.

Returns

The description string, free-form.

int wxVersionInfo::GetMajor ( ) const

Get the major version number.

Returns

Major version number.

int wxVersionInfo::GetMicro ( ) const

Get the micro version, or release number.

Returns

Micro version, or release number.

int wxVersionInfo::GetMinor ( ) const

Get the minor version number.

Returns

Minor version number.

const wxString& wxVersionInfo::GetName ( ) const

Get the name of the object (library).

Returns

Name string.

wxString wxVersionInfo::GetVersionString ( ) const

Get the string representation.

The micro component of the version is ignored/not used if it is 0.

Returns

The version string in the form "name major.minor[.micro]".

bool wxVersionInfo::HasCopyright ( ) const

Returns true if a copyright string has been specified.

See also

GetCopyright()

bool wxVersionInfo::HasDescription ( ) const

Return true if a description string has been specified.

See also

GetDescription()

wxString wxVersionInfo::ToString ( ) const

Get the string representation of this version object.

This function returns the description if it is non-empty or GetVersionString() if there is no description.

See also

GetDescription(), GetVersionString()

21.835 wxVideoMode Struct Reference

`#include <wx/vidmode.h>`

21.835.1 Detailed Description

Determines the sizes and locations of displays connected to the system.

Library: wxCore

Category: Application and System configuration

Predefined objects/pointers: wxDefaultVideoMode

See also

wxClientDisplayRect(), wxDisplaySize(), wxDisplaySizeMM()

Public Member Functions

- wxVideoMode (int width=0, int height=0, int depth=0, int freq=0)

  Constructs this class using the given parameters.
- bool Matches (const wxVideoMode &other) const

  Returns true if this mode matches the other one in the sense that all non zero fields of the other mode have the same value in this one (except for refresh which is allowed to have a greater value).
- int GetWidth () const

  Returns the screen width in pixels (e.g. 640), 0 means unspecified.
- int GetHeight () const

  Returns the screen height in pixels (e.g. 480), 0 means unspecified.
- int GetDepth () const

  Returns bits per pixel (e.g. 32), 1 is monochrome and 0 means unspecified/known.
- bool IsOk () const

  Returns true if the object has been initialized.
- bool operator== (const wxVideoMode &m) const
- bool operator!= (const wxVideoMode &mode) const

Public Attributes

- int w

  The screen width in pixels (e.g. 640), 0 means unspecified.
- int h

  The screen height in pixels (e.g. 480), 0 means unspecified.
- int bpp

  Bits per pixel (e.g. 32), 1 is monochrome and 0 means unspecified/known.
- int refresh

  Refresh frequency in Hz, 0 means unspecified/unknown.

21.835.2 Constructor & Destructor Documentation

wxVideoMode::wxVideoMode (int *width* = 0, int *height* = 0, int *depth* = 0, int *freq* = 0)

Constructs this class using the given parameters.

21.835.3 Member Function Documentation

int wxVideoMode::GetDepth ( ) const

Returns bits per pixel (e.g. 32), 1 is monochrome and 0 means unspecified/known.

int wxVideoMode::GetHeight ( ) const

Returns the screen height in pixels (e.g. 480), 0 means unspecified.

int wxVideoMode::GetWidth ( ) const

Returns the screen width in pixels (e.g. 640), 0 means unspecified.

bool wxVideoMode::IsOk ( ) const

Returns true if the object has been initialized.

bool wxVideoMode::Matches (const wxVideoMode & *other*) const

Returns true if this mode matches the other one in the sense that all non zero fields of the other mode have the same value in this one (except for refresh which is allowed to have a greater value).

bool wxVideoMode::operator!= (const wxVideoMode & *mode*) const

bool wxVideoMode::operator== (const wxVideoMode & *m*) const

21.835.4 Member Data Documentation

int wxVideoMode::bpp

Bits per pixel (e.g. 32), 1 is monochrome and 0 means unspecified/known.

int wxVideoMode::h

The screen height in pixels (e.g. 480), 0 means unspecified.

int wxVideoMode::refresh

Refresh frequency in Hz, 0 means unspecified/unknown.

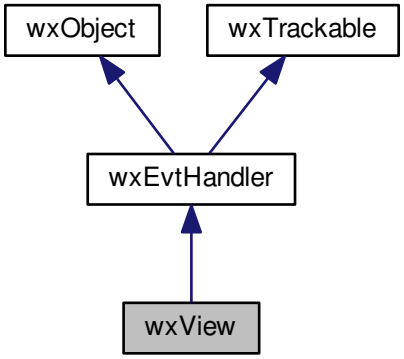
int wxVideoMode::w

The screen width in pixels (e.g. 640), 0 means unspecified.

21.836 wxView Class Reference

`#include <wx/docview.h>`

Inheritance diagram for wxView:



21.836.1 Detailed Description

The view class can be used to model the viewing and editing component of an application's file-based data.

It is part of the document/view framework supported by wxWidgets, and cooperates with the wxDocument, wxDocTemplate and wxDocManager classes.

Library: wxCore

Category: Document/View Framework

See also

wxView Overview, wxDocument, wxDocTemplate, wxDocManager

Public Member Functions

- wxView ()

  Constructor.
- virtual ~wxView ()

  Destructor.
- virtual void Activate (bool activate)

  Call this from your view frame's wxDocChildFrame::OnActivate() member to tell the framework which view is currently active.
- virtual bool Close (bool deleteWindow=true)

  Closes the view by calling OnClose().
- wxDocument * GetDocument () const

  Gets a pointer to the document associated with the view.
- wxDocManager * GetDocumentManager () const

  Returns a pointer to the document manager instance associated with this view.
- wxWindow * GetFrame () const

  Gets the frame associated with the view (if any).

- wxString GetViewName () const

  Gets the name associated with the view (passed to the wxDocTemplate constructor).
- virtual void OnActivateView (bool activate, wxView *activeView, wxView *deactiveView)

  Called when a view is activated by means of Activate().
- virtual void OnChangeFilename ()

  Called when the filename has changed.
- virtual bool OnClose (bool deleteWindow)

  Implements closing behaviour.
- virtual void OnClosingDocument ()

  Override this to clean up the view when the document is being closed.
- virtual bool OnCreate (wxDocument *doc, long flags)

  wxDocManager or wxDocument creates a wxView via a wxDocTemplate.
- virtual wxPrintout * OnCreatePrintout ()

  If the printing framework is enabled in the library, this function returns a wxPrintout object for the purposes of printing.
- virtual void OnDraw (wxDC *dc)=0

  Override this function to render the view on the given device context.
- virtual void OnUpdate (wxView *sender, wxObject *hint=0)

  Called when the view should be updated.
- virtual void SetDocument (wxDocument *doc)

  Associates the given document with the view.
- void SetFrame (wxWindow *frame)

  Sets the frame associated with this view.
- void SetViewName (const wxString &name)

  Sets the view type name.

Public Attributes

- wxDocument * m_viewDocument

  The document associated with this view.
- wxFrame * m_viewFrame

  Frame associated with the view, if any.
- wxString m_viewTypeName

  The view type name given to the wxDocTemplate constructor, copied to this variable when the view is created.

Additional Inherited Members

21.836.2 Constructor & Destructor Documentation

wxView::wxView ( )

Constructor.

Define your own default constructor to initialize application-specific data.

virtual wxView::~wxView ( ) [virtual]

Destructor.

Removes itself from the document's list of views.

21.836.3 Member Function Documentation

virtual void wxView::Activate (bool *activate*) `[virtual]`

Call this from your view frame's wxDocChildFrame::OnActivate() member to tell the framework which view is currently active.

If your windowing system doesn't call wxDocChildFrame::OnActivate(), you may need to call this function from any place where you know the view must be active, and the framework will need to get the current view.

The prepackaged view frame wxDocChildFrame calls Activate() from its wxDocChildFrame::OnActivate() member.

This function calls OnActivateView().

virtual bool wxView::Close (bool *deleteWindow* = `true`) `[virtual]`

Closes the view by calling OnClose().

If *deleteWindow* is true, this function should delete the window associated with the view.

wxDocument* wxView::GetDocument ( ) const

Gets a pointer to the document associated with the view.

wxDocManager* wxView::GetDocumentManager ( ) const

Returns a pointer to the document manager instance associated with this view.

wxWindow* wxView::GetFrame ( ) const

Gets the frame associated with the view (if any).

Note that this "frame" is not a wxFrame at all in the generic MDI implementation which uses notebook pages instead of frames and this is why this method returns a wxWindow and not a wxFrame.

wxString wxView::GetViewName ( ) const

Gets the name associated with the view (passed to the wxDocTemplate constructor).

Not currently used by the framework.

virtual void wxView::OnActivateView (bool *activate*, wxView * *activeView*, wxView * *deactiveView*) `[virtual]`

Called when a view is activated by means of Activate().

The default implementation does nothing.

virtual void wxView::OnChangeFilename ( ) `[virtual]`

Called when the filename has changed.

The default implementation constructs a suitable title and sets the title of the view frame (if any).

virtual bool wxView::OnClose (bool *deleteWindow*) `[virtual]`

Implements closing behaviour.

The default implementation calls wxDocument::Close() to close the associated document. Does not delete the view. The application may wish to do some cleaning up operations in this function, *if* a call to wxDocument::Close() succeeded. For example, if your views all share the same window, you need to disassociate the window from the view and perhaps clear the window. If *deleteWindow* is true, delete the frame associated with the view.

virtual void wxView::OnClosingDocument ( ) [virtual]

Override this to clean up the view when the document is being closed.

virtual bool wxView::OnCreate (wxDocument * *doc,* long *flags*) [virtual]

wxDocManager or wxDocument creates a wxView via a wxDocTemplate.

Just after the wxDocTemplate creates the wxView, it calls OnCreate(). The wxView can create a wxDocChildFrame (or derived class) in its wxView::OnCreate() member function. This wxDocChildFrame provides user interface elements to view and/or edit the contents of the wxDocument.

By default, simply returns true. If the function returns false, the view will be deleted.

virtual wxPrintout* wxView::OnCreatePrintout ( ) [virtual]

If the printing framework is enabled in the library, this function returns a wxPrintout object for the purposes of printing.

It should create a new object every time it is called; the framework will delete objects it creates.

By default, this function returns an instance of wxDocPrintout, which prints and previews one page by calling OnDraw().

Override to return an instance of a class other than wxDocPrintout.

virtual void wxView::OnDraw (wxDC * *dc*) [pure virtual]

Override this function to render the view on the given device context.

virtual void wxView::OnUpdate (wxView * *sender,* wxObject * *hint* = 0) [virtual]

Called when the view should be updated.

Parameters

sender	A pointer to the wxView that sent the update request, or NULL if no single view requested the update (for instance, when the document is opened).
hint	This is unused currently, but may in future contain application-specific information for making updating more efficient.

virtual void wxView::SetDocument (wxDocument * *doc*) [virtual]

Associates the given document with the view.

Normally called by the framework.

void wxView::SetFrame (wxWindow * *frame*)

Sets the frame associated with this view.

The application should call this if possible, to tell the view about the frame.

See GetFrame() for the explanation about the mismatch between the "Frame" in the method name and the type of its parameter.

void wxView::SetViewName (const wxString & *name*)

Sets the view type name.

Should only be called by the framework.

21.836.4 Member Data Documentation

wxDocument* wxView::m_viewDocument

The document associated with this view.

There may be more than one view per document, but there can never be more than one document for one view.

wxFrame* wxView::m_viewFrame

Frame associated with the view, if any.

wxString wxView::m_viewTypeName

The view type name given to the wxDocTemplate constructor, copied to this variable when the view is created.

Not currently used by the framework.

21.837 wxVisualAttributes Struct Reference

`#include <wx/window.h>`

21.837.1 Detailed Description

Struct containing all the visual attributes of a control.

Public Attributes

- wxFont font

  The font used for control label/text inside it.
- wxColour colFg

  The foreground colour.
- wxColour colBg

  The background colour.

21.837.2 Member Data Documentation

wxColour wxVisualAttributes::colBg

The background colour.

May be wxNullColour if the controls background colour is not solid.

wxColour wxVisualAttributes::colFg

The foreground colour.

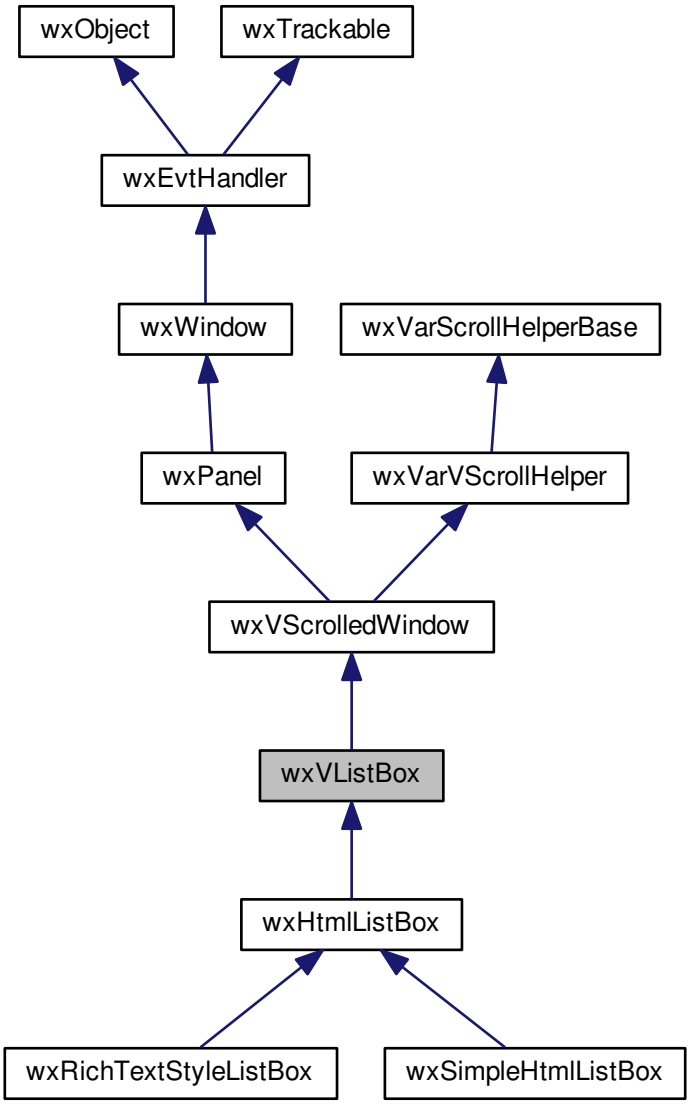
wxFont wxVisualAttributes::font

The font used for control label/text inside it.

21.838 wxVListBox Class Reference

`#include <wx/vlbox.h>`

Inheritance diagram for wxVListBox:



21.838.1 Detailed Description

wxVListBox is a wxListBox-like control with the following two main differences from a regular wxListBox: it can have an arbitrarily huge number of items because it doesn't store them itself but uses the OnDrawItem() callback to draw them (so it is a virtual listbox) and its items can have variable height as determined by OnMeasureItem() (so it is also a listbox with the lines of variable height).

Also, as a consequence of its virtual nature, it doesn't have any methods to append or insert items in it as it isn't necessary to do it: you just have to call SetItemCount() to tell the control how many items it should display. Of course, this also means that you will never use this class directly because it has pure virtual functions, but will need to derive your own class from it (for example, wxHtmlListBox).

However it emits the same events as wxListBox and the same event macros may be used with it. Since wxVListBox does not store its items itself, the events will only contain the index, not any contents such as the string of an item.

Library: wxCore

Category: Controls

See also

wxSimpleHtmlListBox, wxHtmlListBox

Public Member Functions

- wxVListBox ()

  Default constructor, you must call Create() later.
- wxVListBox (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxVListBoxNameStr)

  Normal constructor which calls Create() internally.
- virtual ~wxVListBox ()

  Destructor.
- void Clear ()

  Deletes all items from the control.
- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxVListBoxNameStr)

  Creates the control.
- bool DeselectAll ()

  Deselects all the items in the listbox.
- int GetFirstSelected (unsigned long &cookie) const

  Returns the index of the first selected item in the listbox or `wxNOT_FOUND` if no items are currently selected.
- size_t GetItemCount () const

  Get the number of items in the control.
- wxPoint GetMargins () const

  Returns the margins used by the control.
- wxRect GetItemRect (size_t item) const

  Returns the rectangle occupied by this item in physical coordinates.
- int GetNextSelected (unsigned long &cookie) const

  Returns the index of the next selected item or `wxNOT_FOUND` if there are no more.
- size_t GetSelectedCount () const

  Returns the number of the items currently selected.
- int GetSelection () const

  Get the currently selected item or `wxNOT_FOUND` if there is no selection.

- const wxColour & GetSelectionBackground () const

  Returns the background colour used for the selected cells.
- bool HasMultipleSelection () const

  Returns true if the listbox was created with `wxLB_MULTIPLE` style and so supports multiple selection or false if it is a single selection listbox.
- bool IsCurrent (size_t item) const

  Returns true if this item is the current one, false otherwise.
- bool IsSelected (size_t item) const

  Returns true if this item is selected, false otherwise.
- bool Select (size_t item, bool select=true)

  Selects or deselects the specified item which must be valid (i.e. not equal to `wxNOT_FOUND`).
- bool SelectAll ()

  Selects all the items in the listbox.
- bool SelectRange (size_t from, size_t to)

  Selects all items in the specified range which may be given in any order.
- virtual void SetItemCount (size_t count)

  Set the number of items to be shown in the control.
- void SetSelection (int selection)

  Set the selection to the specified item, if it is -1 the selection is unset.
- void SetSelectionBackground (const wxColour &col)

  Sets the colour to be used for the selected cells background.
- void Toggle (size_t item)

  Toggles the state of the specified item, i.e. selects it if it was unselected and deselects it if it was selected.

- void SetMargins (const wxPoint &pt)

  Set the margins: horizontal margin is the distance between the window border and the item contents while vertical margin is half of the distance between items.
- void SetMargins (wxCoord x, wxCoord y)

  Set the margins: horizontal margin is the distance between the window border and the item contents while vertical margin is half of the distance between items.

Protected Member Functions

- virtual void OnDrawItem (wxDC &dc, const wxRect &rect, size_t n) const =0

  The derived class must implement this function to actually draw the item with the given index on the provided DC.
- virtual void OnDrawBackground (wxDC &dc, const wxRect &rect, size_t n) const

  This method is used to draw the item's background and, maybe, a border around it.
- virtual void OnDrawSeparator (wxDC &dc, wxRect &rect, size_t n) const

  This method may be used to draw separators between the lines.
- virtual wxCoord OnMeasureItem (size_t n) const =0

  The derived class must implement this method to return the height of the specified item (in pixels).

Additional Inherited Members

21.838.2 Constructor & Destructor Documentation

wxVListBox::wxVListBox ( )

Default constructor, you must call Create() later.

wxVListBox::wxVListBox (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxVListBoxNameStr)

Normal constructor which calls Create() internally.

virtual wxVListBox::~wxVListBox ( ) [virtual]

Destructor.

21.838.3 Member Function Documentation

void wxVListBox::Clear ( )

Deletes all items from the control.

bool wxVListBox::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxVListBoxNameStr)

Creates the control.

To finish creating it you also should call SetItemCount() to let it know about the number of items it contains.

The only special style which may be used with wxVListBox is wxLB_MULTIPLE which indicates that the listbox should support multiple selection.

Returns

true on success or false if the control couldn't be created.

bool wxVListBox::DeselectAll ( )

Deselects all the items in the listbox.

This method is only valid for multi selection listboxes.

Returns

true if any items were changed, i.e. if there had been any selected items before, or false if all the items were already deselected.

See also

SelectAll(), Select()

int wxVListBox::GetFirstSelected (unsigned long & *cookie*) const

Returns the index of the first selected item in the listbox or wxNOT_FOUND if no items are currently selected.

cookie is an opaque parameter which should be passed to the subsequent calls to GetNextSelected(). It is needed in order to allow parallel iterations over the selected items.

Here is a typical example of using these functions:

```
unsigned long cookie;
int item = hlbox->GetFirstSelected(cookie);
while ( item != wxNOT_FOUND )
{
    // ... process item ...
    item = hlbox->GetNextSelected(cookie);
}
```

This method is only valid for multi selection listboxes.

size_t wxVListBox::GetItemCount ( ) const

Get the number of items in the control.

See also

SetItemCount()

wxRect wxVListBox::GetItemRect (size_t *item*) const

Returns the rectangle occupied by this item in physical coordinates.

If the item is not currently visible, returns an empty rectangle.

Since

2.9.0

wxPoint wxVListBox::GetMargins ( ) const

Returns the margins used by the control.

The `x` field of the returned point is the horizontal margin and the `y` field is the vertical one.

See also

SetMargins()

int wxVListBox::GetNextSelected (unsigned long & *cookie*) const

Returns the index of the next selected item or `wxNOT_FOUND` if there are no more.

This method is only valid for multi selection listboxes.

See also

GetFirstSelected()

size_t wxVListBox::GetSelectedCount ( ) const

Returns the number of the items currently selected.

It is valid for both single and multi selection controls. In the former case it may only return 0 or 1 however.

See also

IsSelected(), GetFirstSelected(), GetNextSelected()

int wxVListBox::GetSelection ( ) const

Get the currently selected item or `wxNOT_FOUND` if there is no selection.

const wxColour& wxVListBox::GetSelectionBackground ( ) const

Returns the background colour used for the selected cells.

By default the standard system colour is used.

See also

wxSystemSettings::GetColour(), SetSelectionBackground()

bool wxVListBox::HasMultipleSelection ( ) const

Returns true if the listbox was created with `wxLB_MULTIPLE` style and so supports multiple selection or false if it is a single selection listbox.

bool wxVListBox::IsCurrent (size_t *item*) const

Returns true if this item is the current one, false otherwise.

The current item is always the same as selected one for the single selection listbox and in this case this method is equivalent to IsSelected() but they are different for multi selection listboxes where many items may be selected but only one (at most) is current.

bool wxVListBox::IsSelected (size_t *item*) const

Returns true if this item is selected, false otherwise.

virtual void wxVListBox::OnDrawBackground (wxDC & *dc*, const wxRect & *rect*, size_t *n*) const `[protected], [virtual]`

This method is used to draw the item's background and, maybe, a border around it.

The base class version implements a reasonable default behaviour which consists in drawing the selected item with the standard background colour and drawing a border around the item if it is either selected or current.

Todo Change this function signature to non-const.

virtual void wxVListBox::OnDrawItem (wxDC & *dc*, const wxRect & *rect*, size_t *n*) const `[protected], [pure virtual]`

The derived class must implement this function to actually draw the item with the given index on the provided DC.

Parameters

dc	The device context to use for drawing.
rect	The bounding rectangle for the item being drawn (DC clipping region is set to this rectangle before calling this function).

n	The index of the item to be drawn.

Todo Change this function signature to non-const.

virtual void wxVListBox::OnDrawSeparator (wxDC & *dc*, wxRect & *rect*, size_t *n*) const `[protected], [virtual]`

This method may be used to draw separators between the lines.

The rectangle passed to it may be modified, typically to deflate it a bit before passing to OnDrawItem().

The base class version of this method doesn't do anything.

Parameters

dc	The device context to use for drawing.
rect	The bounding rectangle for the item.
n	The index of the item.

Todo Change this function signature to non-const.

virtual wxCoord wxVListBox::OnMeasureItem (size_t *n*) const `[protected], [pure virtual]`

The derived class must implement this method to return the height of the specified item (in pixels).

bool wxVListBox::Select (size_t *item*, bool *select* = `true`)

Selects or deselects the specified item which must be valid (i.e. not equal to `wxNOT_FOUND`).

Returns

true if the items selection status has changed or false otherwise.

This function is only valid for the multiple selection listboxes, use SetSelection() for the single selection ones.

bool wxVListBox::SelectAll ( )

Selects all the items in the listbox.

Returns

true if any items were changed, i.e. if there had been any unselected items before, or false if all the items were already selected.

This method is only valid for multi selection listboxes.

See also

DeselectAll(), Select()

bool wxVListBox::SelectRange (size_t *from*, size_t *to*)

Selects all items in the specified range which may be given in any order.

Returns

true if the items selection status has changed or false otherwise.

This method is only valid for multi selection listboxes.

See also

SelectAll(), Select()

virtual void wxVListBox::SetItemCount (size_t *count*) [virtual]

Set the number of items to be shown in the control.

This is just a synonym for wxVScrolledWindow::SetRowCount().

void wxVListBox::SetMargins (const wxPoint & *pt*)

Set the margins: horizontal margin is the distance between the window border and the item contents while vertical margin is half of the distance between items.

By default both margins are 0.

void wxVListBox::SetMargins (wxCoord *x*, wxCoord *y*)

Set the margins: horizontal margin is the distance between the window border and the item contents while vertical margin is half of the distance between items.

By default both margins are 0.

void wxVListBox::SetSelection (int *selection*)

Set the selection to the specified item, if it is -1 the selection is unset.

The selected item will be automatically scrolled into view if it isn't currently visible.

This method may be used both with single and multiple selection listboxes.

void wxVListBox::SetSelectionBackground (const wxColour & *col*)

Sets the colour to be used for the selected cells background.

The background of the standard cells may be changed by simply calling wxWindow::SetBackgroundColour().

Note

Using a non-default background colour may result in control having an appearance different from the similar native controls and should be avoided in general.

See also

GetSelectionBackground()

void wxVListBox::Toggle (size_t *item*)

Toggles the state of the specified *item*, i.e. selects it if it was unselected and deselects it if it was selected.

This method is only valid for multi selection listboxes.

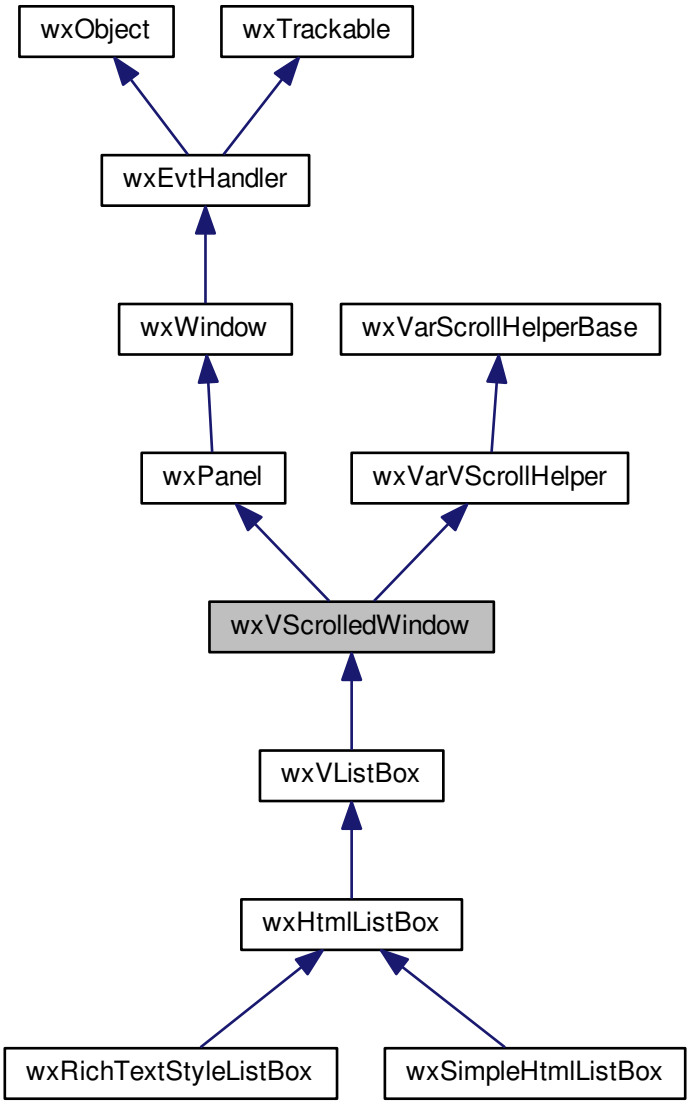
See also

Select()

21.839 wxVScrolledWindow Class Reference

`#include <wx/vscroll.h>`

Inheritance diagram for wxVScrolledWindow:



21.839.1 Detailed Description

In the name of this class, "V" may stand for "variable" because it can be used for scrolling rows of variable heights; "virtual", because it is not necessary to know the heights of all rows in advance – only those which are shown on the screen need to be measured; or even "vertical", because this class only supports scrolling vertically.

In any case, this is a generalization of wxScrolled which can be only used when all rows have the same heights. It lacks some other wxScrolled features however, notably it can't scroll specific pixel sizes of the window or its exact client area size.

To use this class, you need to derive from it and implement the OnGetRowHeight() pure virtual method. You also must call SetRowCount() to let the base class know how many rows it should display, but from that moment on the scrolling is handled entirely by wxVScrolledWindow. You only need to draw the visible part of contents in your `OnPaint()` method as usual. You should use GetVisibleRowsBegin() and GetVisibleRowsEnd() to select the lines to display. Note that the device context origin is not shifted so the first visible row always appears at the point (0, 0) in physical as well as logical coordinates.

21.839.2 wxWidgets 2.8 Compatibility Functions

The following functions provide backwards compatibility for applications originally built using wxVScrolledWindow in 2.6 or 2.8. Originally, wxVScrolledWindow referred to scrolling "lines". We now use "units" in wxVarScrollHelperBase to avoid implying any orientation (since the functions are used for both horizontal and vertical scrolling in derived classes). And in the new wxVScrolledWindow and wxHScrolledWindow classes, we refer to them as "rows" and "columns", respectively. This is to help clear some confusion in not only those classes, but also in wxHVScrolledWindow where functions are inherited from both.

You are encouraged to update any existing code using these function to use the new replacements mentioned below, and avoid using these functions for any new code as they are deprecated.

`size_t GetFirstVisibleLine() const`	Deprecated for GetVisibleRowsBegin().
`size_t GetLastVisibleLine() const`	Deprecated for GetVisibleRowsEnd(). This function originally had a slight design flaw in that it was possible to return `(size_t)-1` (ie: a large positive number) if the scroll position was 0 and the first line wasn't completely visible.
`size_t GetLineCount() const`	Deprecated for GetRowCount().
`int HitTest(wxCoord x, wxCoord y) const` `int HitTest(const wxPoint& pt) const`	Deprecated for VirtualHitTest().
`virtual wxCoord OnGetLineHeight(size_t line) const`	Deprecated for OnGetRowHeight().
`virtual void OnGetLinesHint(size_t lineMin, size_t lineMax) const`	Deprecated for OnGetRowsHeightHint().
`virtual void RefreshLine(size_t line)`	Deprecated for RefreshRow().
`virtual void RefreshLines(size_t from, size_t to)`	Deprecated for RefreshRows().
`virtual bool ScrollLines(int lines)`	Deprecated for ScrollRows().
`virtual bool ScrollPages(int pages)`	Deprecated for ScrollRowPages().
`bool ScrollToLine(size_t line)`	Deprecated for ScrollToRow().
`void SetLineCount(size_t count)`	Deprecated for SetRowCount().

Library: wxCore

Category: Miscellaneous Windows

See also

wxHScrolledWindow, wxHVScrolledWindow

Public Member Functions

- wxVScrolledWindow ()

  Default constructor, you must call Create() later.

- wxVScrolledWindow (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  This is the normal constructor, no need to call Create() after using this constructor.

- bool Create (wxWindow *parent, wxWindowID id=wxID_ANY, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  Same as the non-default constructor, but returns a status code: true if ok, false if the window couldn't be created.

Additional Inherited Members

21.839.3 Constructor & Destructor Documentation

wxVScrolledWindow::wxVScrolledWindow ( )

Default constructor, you must call Create() later.

wxVScrolledWindow::wxVScrolledWindow (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxPanelNameStr`)

This is the normal constructor, no need to call Create() after using this constructor.

Note

`wxVSCROLL` is always automatically added to the style, there is no need to specify it explicitly.

Parameters

parent	The parent window, must not be NULL.
id	The identifier of this window, wxID_ANY by default.
pos	The initial window position.
size	The initial window size.
style	The window style. There are no special style bits defined for this class.
name	The name for this window; usually not used.

21.839.4 Member Function Documentation

bool wxVScrolledWindow::Create (wxWindow * *parent*, wxWindowID *id* = wxID_ANY, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxPanelNameStr`)

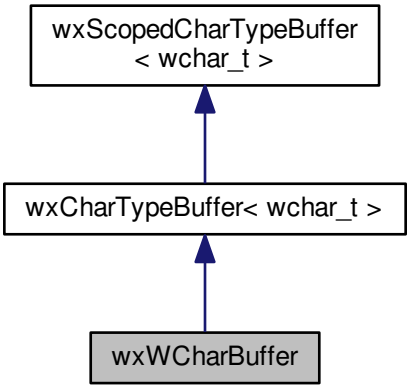
Same as the non-default constructor, but returns a status code: true if ok, false if the window couldn't be created.

Just as with the constructor, the `wxVSCROLL` style is always used, there is no need to specify it explicitly.

21.840 wxWCharBuffer Class Reference

`#include <wx/buffer.h>`

Inheritance diagram for wxWCharBuffer:



21.840.1 Detailed Description

This is a specialization of wxCharTypeBuffer<T> for `wchar_t` type.

Library: None; this class implementation is entirely header-based.

Category: Data Structures

Public Types

- typedef wxCharTypeBuffer< wchar_t > wxCharTypeBufferBase
- typedef wxScopedCharTypeBuffer < wchar_t > wxScopedCharTypeBufferBase

Public Member Functions

- wxWCharBuffer (const wxCharTypeBufferBase &buf)
- wxWCharBuffer (const wxScopedCharTypeBufferBase &buf)
- wxWCharBuffer (const CharType *str=NULL)
- wxWCharBuffer (size_t len)
- wxWCharBuffer (const wxCStrData &cstr)

Additional Inherited Members

21.840.2 Member Typedef Documentation

typedef wxCharTypeBuffer<wchar_t> wxWCharBuffer::wxCharTypeBufferBase

typedef wxScopedCharTypeBuffer<wchar_t> wxWCharBuffer::wxScopedCharTypeBufferBase

21.840.3 Constructor & Destructor Documentation

wxWCharBuffer::wxWCharBuffer (const wxCharTypeBufferBase & *buf*)

wxWCharBuffer::wxWCharBuffer (const wxScopedCharTypeBufferBase & *buf*)

wxWCharBuffer::wxWCharBuffer (const CharType * *str* = NULL)

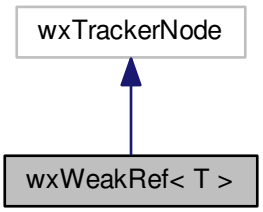
wxWCharBuffer::wxWCharBuffer (size_t *len*)

wxWCharBuffer::wxWCharBuffer (const wxCStrData & *cstr*)

21.841 wxWeakRef< T > Class Template Reference

`#include <wx/weakref.h>`

Inheritance diagram for wxWeakRef< T >:



21.841.1 Detailed Description

template<typename T>class wxWeakRef< T >

wxWeakRef<T> is a template class for weak references to wxWidgets objects, such as wxEvtHandler, wxWindow and wxObject.

A weak reference behaves much like an ordinary pointer, but when the object pointed is destroyed, the weak reference is automatically reset to a NULL pointer.

wxWeakRef<T> can be used whenever one must keep a pointer to an object that one does not directly own, and that may be destroyed before the object holding the reference.

wxWeakRef<T> is a small object and the mechanism behind it is fast (**O(1)**). So the overall cost of using it is small.

Example:

```
wxWindow *wnd = new wxWindow( parent, wxID_ANY, "wxWindow" );
wxWeakRef<wxWindow> wr = wnd;
wxWindowRef wr2 = wnd;        // Same as above, but using a typedef
// Do things with window
wnd->Show( true );
// Weak ref is used like an ordinary pointer
```

```
wr->Show( false );
wnd->Destroy();
// Now the weak ref has been reset, so we don't risk accessing
// a dangling pointer:
wxASSERT( wr==NULL );
```

wxWeakRef<T> works for any objects that are derived from wxTrackable. By default, wxEvtHandler and wxWindow derive from wxTrackable. However, wxObject does not, so types like wxFont and wxColour are not trackable. The example below shows how to create a wxObject derived class that is trackable:

```
class wxMyTrackableObject : public wxObject, public wxTrackable
{
    // ... other members here
};
```

The following types of weak references are predefined:

```
typedef wxWeakRef<wxEvtHandler>  wxEvtHandlerRef;
typedef wxWeakRef<wxWindow>      wxWindowRef;
```

Template Parameters

T	The type to which the smart pointer points to.

Library: None; this class implementation is entirely header-based.

Category: Smart Pointers

See also

wxSharedPtr<T>, wxScopedPtr<T>

Public Types

- typedef T element_type

  Type of the element stored by this reference.

Public Member Functions

- wxWeakRef (T *pobj=NULL)

  Constructor.
- wxWeakRef (const wxWeakRef< T > &wr)

  Copy constructor.
- virtual ~wxWeakRef ()

  Destructor.
- virtual void OnObjectDestroy ()

  Called when the tracked object is destroyed.
- void Release ()

  Release currently tracked object and rests object reference.
- T * get () const

  Returns pointer to the tracked object or NULL.
- T * operator= (wxWeakRef< T > &wr)

  Release currently tracked object and start tracking the same object as the wxWeakRef wr.
- T * operator* () const

  Implicit conversion to T.*

- T & operator* () const

  Returns a reference to the tracked object.
- T * operator-> ()

  Smart pointer member access.
- T * operator= (T *pobj)

  Releases the currently tracked object and starts tracking pobj.

21.841.2 Member Typedef Documentation

template<typename T > typedef T wxWeakRef< T >::element_type

Type of the element stored by this reference.

21.841.3 Constructor & Destructor Documentation

template<typename T > wxWeakRef< T >::wxWeakRef (T * *pobj* = NULL)

Constructor.

The weak reference is initialized to *pobj*.

template<typename T > wxWeakRef< T >::wxWeakRef (const wxWeakRef< T > & *wr*)

Copy constructor.

template<typename T > virtual wxWeakRef< T >::~wxWeakRef ( ) [virtual]

Destructor.

21.841.4 Member Function Documentation

template<typename T > T* wxWeakRef< T >::get ( ) const

Returns pointer to the tracked object or NULL.

template<typename T > virtual void wxWeakRef< T >::OnObjectDestroy ( ) [virtual]

Called when the tracked object is destroyed.

Be default sets internal pointer to NULL. You need to call this method if you override it.

template<typename T > T* wxWeakRef< T >::operator* ( ) const

Implicit conversion to T*.

Returns pointer to the tracked object or NULL.

template<typename T > T& wxWeakRef< T >::operator* ( ) const

Returns a reference to the tracked object.

If the internal pointer is NULL this method will cause an assert in debug mode.

template<typename T > T* wxWeakRef< T >::operator->( )

Smart pointer member access.

Returns a pointer to the tracked object. If the internal pointer is NULL this method will cause an assert in debug mode.

template<typename T > T* wxWeakRef< T >::operator=(wxWeakRef< T > & *wr*)

Release currently tracked object and start tracking the same object as the wxWeakRef *wr*.

template<typename T > T* wxWeakRef< T >::operator=(T * *pobj*)

Releases the currently tracked object and starts tracking *pobj*.

A weak reference may be reset by passing NULL as *pobj*.

template<typename T > void wxWeakRef< T >::Release( )

Release currently tracked object and rests object reference.

21.842 wxWeakRefDynamic< T > Class Template Reference

`#include <wx/weakref.h>`

21.842.1 Detailed Description

template<typename T>class wxWeakRefDynamic< T >

wxWeakRefDynamic<T> is a template class for weak references that is used in the same way as wxWeakRef<T>.

The only difference is that wxWeakRefDynamic defaults to using `dynamic_cast` for establishing the object reference (while wxWeakRef defaults to `static_cast`).

So, wxWeakRef will detect a type mismatch during compile time and will have a little better run-time performance. The role of wxWeakRefDynamic is to handle objects which derived type one does not know.

Note

> wxWeakRef<T> selects an implementation based on the static type of T. If T does not have wxTrackable statically, it defaults to a mixed- mode operation, where it uses `dynamic_cast` as the last measure (if available from the compiler and enabled when building wxWidgets).

For general cases, wxWeakRef<T> is the better choice.

For API documentation, see: wxWeakRef<T>.

Template Parameters

T	The type to which the smart pointer points to.

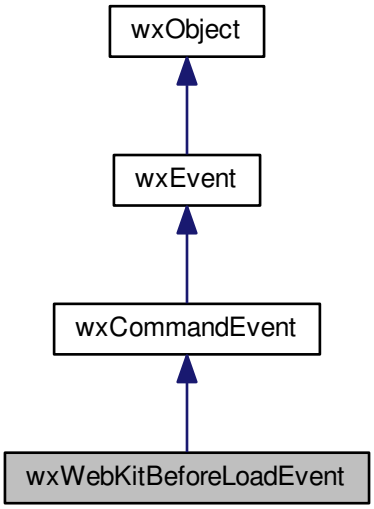
Library: None; this class implementation is entirely header-based.

Category: Smart Pointers

21.843 wxWebKitBeforeLoadEvent Class Reference

`#include <wx/html/webkit.h>`

Inheritance diagram for wxWebKitBeforeLoadEvent:



Public Member Functions

- bool IsCancelled ()
- void Cancel (bool cancel=true)
- wxString GetURL ()
- void SetURL (const wxString &url)
- void SetNavigationType (int navType)
- int GetNavigationType ()
- wxWebKitBeforeLoadEvent (wxWindow *win=0)

Additional Inherited Members

21.843.1 Constructor & Destructor Documentation

wxWebKitBeforeLoadEvent::wxWebKitBeforeLoadEvent (wxWindow * *win* = `0`)

21.843.2 Member Function Documentation

void wxWebKitBeforeLoadEvent::Cancel (bool *cancel* = `true`)

int wxWebKitBeforeLoadEvent::GetNavigationType ( )

wxString wxWebKitBeforeLoadEvent::GetURL ( )

bool wxWebKitBeforeLoadEvent::IsCancelled ( )

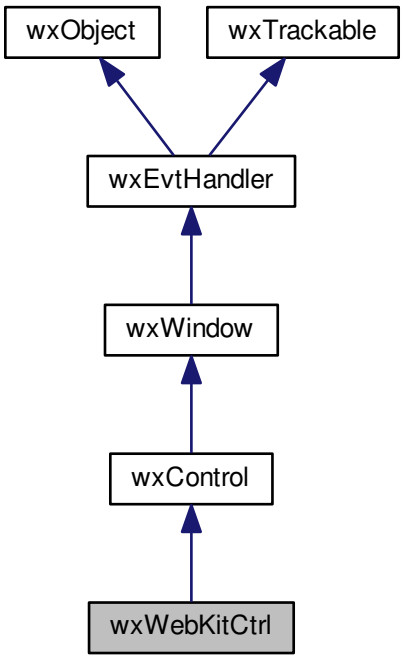
void wxWebKitBeforeLoadEvent::SetNavigationType (int *navType*)

void wxWebKitBeforeLoadEvent::SetURL (const wxString & *url*)

21.844 wxWebKitCtrl Class Reference

`#include <wx/html/webkit.h>`

Inheritance diagram for wxWebKitCtrl:



21.844.1 Detailed Description

This control is a native wrapper around the Safari web browsing engine.

This wrapper differs from the one in wxWebView in that this version supports functionality specific to WebKit, such as having RunScript return a value, which is a very critical feature in many web embedding scenarios.

This class is only available on OSX.

Public Member Functions

- wxWebKitCtrl ()
- wxWebKitCtrl (wxWindow *parent, wxWindowID winid, const wxString &strURL, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxWebKitCtrlNameStr)
- bool Create (wxWindow *parent, wxWindowID winid, const wxString &strURL, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxValidator &validator=wxDefaultValidator, const wxString &name=wxWebKitCtrlNameStr)

- virtual ~wxWebKitCtrl ()
- void LoadURL (const wxString &url)
- bool CanGoBack ()
- bool CanGoForward ()
- bool GoBack ()
- bool GoForward ()
- void Reload ()
- void Stop ()
- bool CanGetPageSource ()
- wxString GetPageSource ()
- void SetPageSource (const wxString &source, const wxString &baseUrl=wxEmptyString)
- wxString GetPageURL ()
- void SetPageTitle (const wxString &title)
- wxString GetPageTitle ()
- void SetTitle (const wxString &title)
- wxString GetTitle ()
- wxString GetSelection ()
- bool CanIncreaseTextSize ()
- void IncreaseTextSize ()
- bool CanDecreaseTextSize ()
- void DecreaseTextSize ()
- void Print (bool showPrompt=false)
- void MakeEditable (bool enable=true)
- bool IsEditable ()
- wxString RunScript (const wxString &javascript)
- void SetScrollPos (int pos)
- int GetScrollPos ()

Additional Inherited Members

21.844.2 Constructor & Destructor Documentation

wxWebKitCtrl::wxWebKitCtrl ( )

wxWebKitCtrl::wxWebKitCtrl (wxWindow * *parent*, wxWindowID *winid*, const wxString & *strURL*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxWebKitCtrlNameStr)

virtual wxWebKitCtrl::~wxWebKitCtrl ( ) [virtual]

21.844.3 Member Function Documentation

bool wxWebKitCtrl::CanDecreaseTextSize ( )

bool wxWebKitCtrl::CanGetPageSource ( )

bool wxWebKitCtrl::CanGoBack ( )

bool wxWebKitCtrl::CanGoForward ( )

bool wxWebKitCtrl::CanIncreaseTextSize ( )

bool wxWebKitCtrl::Create (wxWindow * *parent*, wxWindowID *winid*, const wxString & *strURL*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxValidator & *validator* = wxDefaultValidator, const wxString & *name* = wxWebKitCtrlNameStr)

void wxWebKitCtrl::DecreaseTextSize ( )

wxString wxWebKitCtrl::GetPageSource ( )

wxString wxWebKitCtrl::GetPageTitle ( )

wxString wxWebKitCtrl::GetPageURL ( )

int wxWebKitCtrl::GetScrollPos ( )

wxString wxWebKitCtrl::GetSelection ( )

wxString wxWebKitCtrl::GetTitle ( )

bool wxWebKitCtrl::GoBack ( )

bool wxWebKitCtrl::GoForward ( )

void wxWebKitCtrl::IncreaseTextSize ( )

bool wxWebKitCtrl::IsEditable ( )

void wxWebKitCtrl::LoadURL (const wxString & *url*)

void wxWebKitCtrl::MakeEditable (bool *enable* = `true`)

void wxWebKitCtrl::Print (bool *showPrompt* = `false`)

void wxWebKitCtrl::Reload ( )

wxString wxWebKitCtrl::RunScript (const wxString & *javascript*)

void wxWebKitCtrl::SetPageSource (const wxString & *source,* const wxString & *baseUrl* = wxEmptyString)

void wxWebKitCtrl::SetPageTitle (const wxString & *title*)

void wxWebKitCtrl::SetScrollPos (int *pos*)

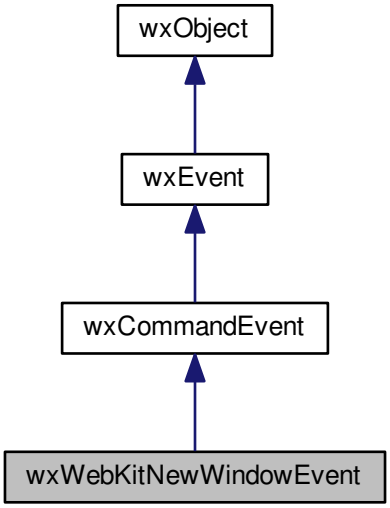
void wxWebKitCtrl::SetTitle (const wxString & *title*)

void wxWebKitCtrl::Stop ( )

21.845 wxWebKitNewWindowEvent Class Reference

`#include <wx/html/webkit.h>`

Inheritance diagram for wxWebKitNewWindowEvent:



Public Member Functions

- wxString GetURL () const
- void SetURL (const wxString &url)
- wxString GetTargetName () const
- void SetTargetName (const wxString &name)
- wxWebKitNewWindowEvent (wxWindow *win=0)

Additional Inherited Members

21.845.1 Constructor & Destructor Documentation

wxWebKitNewWindowEvent::wxWebKitNewWindowEvent (wxWindow * *win* = 0)

21.845.2 Member Function Documentation

wxString wxWebKitNewWindowEvent::GetTargetName ( ) const

wxString wxWebKitNewWindowEvent::GetURL ( ) const

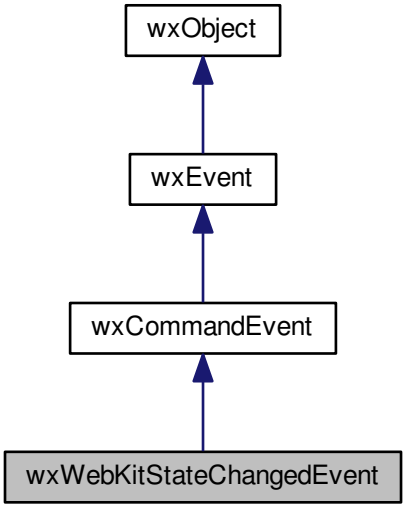
void wxWebKitNewWindowEvent::SetTargetName (const wxString & *name*)

void wxWebKitNewWindowEvent::SetURL (const wxString & *url*)

21.846 wxWebKitStateChangedEvent Class Reference

`#include <wx/html/webkit.h>`

Inheritance diagram for wxWebKitStateChangedEvent:



Public Member Functions

- int GetState ()
- void SetState (const int state)
- wxString GetURL ()
- void SetURL (const wxString &url)
- wxWebKitStateChangedEvent (wxWindow *win=0)

Additional Inherited Members

21.846.1 Constructor & Destructor Documentation

wxWebKitStateChangedEvent::wxWebKitStateChangedEvent (wxWindow * *win* = 0)

21.846.2 Member Function Documentation

int wxWebKitStateChangedEvent::GetState ( )

wxString wxWebKitStateChangedEvent::GetURL ( )

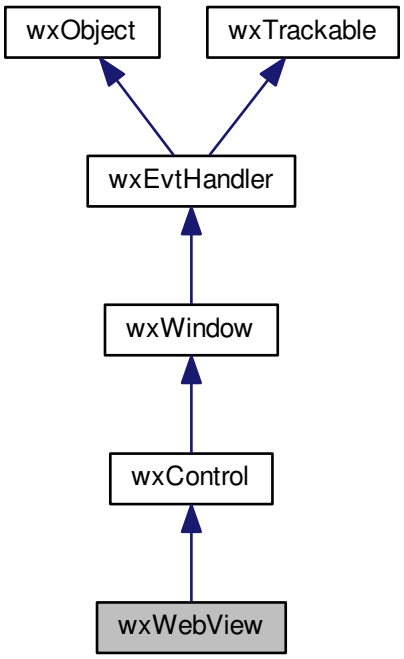
void wxWebKitStateChangedEvent::SetState (const int *state*)

void wxWebKitStateChangedEvent::SetURL (const wxString & *url*)

21.847 wxWebView Class Reference

`#include <wx/webview.h>`

Inheritance diagram for wxWebView:



21.847.1 Detailed Description

This control may be used to render web (HTML / CSS / javascript) documents.

It is designed to allow the creation of multiple backends for each port, although currently just one is available. It differs from wxHtmlWindow in that each backend is actually a full rendering engine, Trident on MSW and Webkit on OSX and GTK. This allows the correct viewing complex pages with javascript and css.

21.847.2 Backend Descriptions

wxWEBVIEW_BACKEND_IE (MSW)

The IE backend uses Microsoft's Trident rendering engine, specifically the version used by the locally installed copy of Internet Explorer. As such it is only available for the MSW port. By default recent versions of the `WebBrowser` control, which this backend uses, emulate Internet Explorer 7. This can be changed with a registry setting, see `this` article for more information. This backend has full support for custom schemes and virtual file systems.

wxWEBVIEW_WEBKIT (GTK)

Under GTK the WebKit backend uses `WebKitGTK+`. The current minimum version required is 1.3.1 which ships by default with Ubuntu Natty and Debian Wheezy and has the package name libwebkitgtk-dev. Custom schemes and virtual files systems are supported under this backend, however embedded resources such as images and stylesheets are currently loaded using the data:// scheme.

wxWEBVIEW_WEBKIT (OSX)

The OSX WebKit backend uses Apple's `WebView` class. This backend has full support for custom schemes and virtual file systems.

21.847.3 Asynchronous Notifications

Many of the methods in wxWebView are asynchronous, i.e. they return immediately and perform their work in the background. This includes functions such as LoadURL() and Reload(). To receive notification of the progress and completion of these functions you need to handle the events that are provided. Specifically `wxEVT_WEBVIEW_LOADED` notifies when the page or a sub-frame has finished loading and `wxEVT_WEBVIEW_ERROR` notifies that an error has occurred.

21.847.4 Virtual File Systems and Custom Schemes

wxWebView supports the registering of custom scheme handlers, for example `file` or `http`. To do this create a new class which inherits from wxWebViewHandler, where wxWebHandler::GetFile() returns a pointer to a wxFSFile which represents the given url. You can then register your handler with RegisterHandler() it will be called for all pages and resources.

wxWebViewFSHandler is provided to access the virtual file system encapsulated by wxFileSystem. The wxMemoryFSHandler documentation gives an example of how this may be used.

wxWebViewArchiveHandler is provided to allow the navigation of pages inside a zip archive. It supports paths of the form: `scheme:///C`:/example/docs.zip;protocol=zip/main.htm

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxWebViewEvent& event)

Event macros for events emitted by this class:

- EVT_WEBVIEW_NAVIGATING(id, func): Process a `wxEVT_WEBVIEW_NAVIGATING` event, generated before trying to get a resource. This event may be vetoed to prevent navigating to this resource. Note that if the displayed HTML document has several frames, one such event will be generated per frame.
- EVT_WEBVIEW_NAVIGATED(id, func): Process a `wxEVT_WEBVIEW_NAVIGATED` event generated after it was confirmed that a resource would be requested. This event may not be vetoed. Note that if the displayed HTML document has several frames, one such event will be generated per frame.
- EVT_WEBVIEW_LOADED(id, func): Process a `wxEVT_WEBVIEW_LOADED` event generated when the document is fully loaded and displayed. Note that if the displayed HTML document has several frames, one such event will be generated per frame.
- EVT_WEBVIEW_ERROR(id, func): Process a `wxEVT_WEBVIEW_ERROR` event generated when a navigation error occurs. The integer associated with this event will be a wxWebNavigationError item. The string associated with this event may contain a backend-specific more precise error message/code.
- EVT_WEBVIEW_NEWWINDOW(id, func): Process a `wxEVT_WEBVIEW_NEWWINDOW` event, generated when a new window is created. You must handle this event if you want anything to happen, for example to load the page in a new window or tab.
- EVT_WEBVIEW_TITLE_CHANGED(id, func): Process a `wxEVT_WEBVIEW_TITLE_CHANGED` event, generated when the page title changes. Use GetString to get the title.

Since

2.9.3

Library: wxWebView

Category: Controls, WebView

See also

wxWebViewHandler, wxWebViewEvent

Public Member Functions

- virtual bool Create (wxWindow *parent, wxWindowID id, const wxString &url=wxWebViewDefaultURLStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxWebViewNameStr)=0

  Creation function for two-step creation.
- virtual wxString GetCurrentTitle () const =0

  Get the title of the current web page, or its URL/path if title is not available.
- virtual wxString GetCurrentURL () const =0

  Get the URL of the currently displayed document.
- virtual void * GetNativeBackend () const =0

  Return the pointer to the native backend used by this control.
- virtual wxString GetPageSource () const =0

  Get the HTML source code of the currently displayed document.
- virtual wxString GetPageText () const =0

  Get the text of the current page.
- virtual bool IsBusy () const =0

  Returns whether the web control is currently busy (e.g. loading a page).
- virtual bool IsEditable () const =0

  Returns whether the web control is currently editable.
- virtual void LoadURL (const wxString &url)=0

  Load a web page from a URL.
- virtual void Print ()=0

  Opens a print dialog so that the user may print the currently displayed page.
- virtual void RegisterHandler (wxSharedPtr< wxWebViewHandler > handler)=0

  Registers a custom scheme handler.
- virtual void Reload (wxWebViewReloadFlags flags=wxWEBVIEW_RELOAD_DEFAULT)=0

  Reload the currently displayed URL.
- virtual void RunScript (const wxString &javascript)=0

  Runs the given javascript code.
- virtual void SetEditable (bool enable=true)=0

  Set the editable property of the web control.
- virtual void SetPage (const wxString &html, const wxString &baseUrl)=0

  Set the displayed page source to the contents of the given string.
- virtual void SetPage (wxInputStream &html, wxString baseUrl)

  Set the displayed page source to the contents of the given stream.
- virtual void Stop ()=0

  Stop the current page loading process, if any.

Clipboard

- virtual bool CanCopy () const =0
  Returns true if the current selection can be copied.
- virtual bool CanCut () const =0
  Returns true if the current selection can be cut.
- virtual bool CanPaste () const =0
  Returns true if data can be pasted.
- virtual void Copy ()=0
  Copies the current selection.
- virtual void Cut ()=0
  Cuts the current selection.
- virtual void Paste ()=0
  Pastes the current data.

Context Menu

- virtual void EnableContextMenu (bool enable=true)
  Enable or disable the right click context menu.
- virtual bool IsContextMenuEnabled () const
  Returns true if a context menu will be shown on right click.

History

- virtual bool CanGoBack () const =0
  Returns true if it is possible to navigate backward in the history of visited pages.
- virtual bool CanGoForward () const =0
  Returns true if it is possible to navigate forward in the history of visited pages.
- virtual void ClearHistory ()=0
  Clear the history, this will also remove the visible page.
- virtual void EnableHistory (bool enable=true)=0
  Enable or disable the history.
- virtual wxVector< wxSharedPtr < wxWebViewHistoryItem > > GetBackwardHistory ()=0
  Returns a list of items in the back history.
- virtual wxVector< wxSharedPtr < wxWebViewHistoryItem > > GetForwardHistory ()=0
  Returns a list of items in the forward history.
- virtual void GoBack ()=0
  Navigate back in the history of visited pages.
- virtual void GoForward ()=0
  Navigate forward in the history of visited pages.
- virtual void LoadHistoryItem (wxSharedPtr< wxWebViewHistoryItem > item)=0
  Loads a history item.

Selection

- virtual void ClearSelection ()=0
  Clears the current selection.
- virtual void DeleteSelection ()=0
  Deletes the current selection.
- virtual wxString GetSelectedSource () const =0
  Returns the currently selected source, if any.
- virtual wxString GetSelectedText () const =0
  Returns the currently selected text, if any.
- virtual bool HasSelection () const =0
  Returns true if there is a current selection.
- virtual void SelectAll ()=0
  Selects the entire page.

Undo / Redo

- virtual bool CanRedo () const =0
  Returns true if there is an action to redo.
- virtual bool CanUndo () const =0
  Returns true if there is an action to undo.
- virtual void Redo ()=0
  Redos the last action.
- virtual void Undo ()=0
  Undos the last action.

Finding

- virtual long Find (const wxString &text, wxWebViewFindFlags flags=wxWEBVIEW_FIND_DEFAULT)=0
  Finds a phrase on the current page and if found, the control will scroll the phrase into view and select it.

Zoom

- virtual bool CanSetZoomType (wxWebViewZoomType type) const =0
  Retrieve whether the current HTML engine supports a zoom type.
- virtual wxWebViewZoom GetZoom () const =0
  Get the zoom factor of the page.
- virtual wxWebViewZoomType GetZoomType () const =0
  Get how the zoom factor is currently interpreted.
- virtual void SetZoom (wxWebViewZoom zoom)=0
  Set the zoom factor of the page.
- virtual void SetZoomType (wxWebViewZoomType zoomType)=0
  Set how to interpret the zoom factor.

Static Public Member Functions

- static wxWebView * New (const wxString &backend=wxWebViewBackendDefault)
  Factory function to create a new wxWebView with two-step creation, wxWebView::Create should be called on the returned object.
- static wxWebView * New (wxWindow *parent, wxWindowID id, const wxString &url=wxWebViewDefaultURLStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, const wxString &backend=wxWebViewBackendDefault, long style=0, const wxString &name=wxWebViewNameStr)
  Factory function to create a new wxWebView using a wxWebViewFactory.
- static void RegisterFactory (const wxString &backend, wxSharedPtr< wxWebViewFactory > factory)
  Allows the registering of new backend for wxWebView.

Additional Inherited Members

21.847.5 Member Function Documentation

virtual bool wxWebView::CanCopy ( ) const `[pure virtual]`

Returns true if the current selection can be copied.

Note

This always returns `true` on the OSX WebKit backend.

virtual bool wxWebView::CanCut ( ) const `[pure virtual]`

Returns true if the current selection can be cut.

Note

This always returns `true` on the OSX WebKit backend.

virtual bool wxWebView::CanGoBack ( ) const `[pure virtual]`

Returns true if it is possible to navigate backward in the history of visited pages.

virtual bool wxWebView::CanGoForward ( ) const `[pure virtual]`

Returns true if it is possible to navigate forward in the history of visited pages.

virtual bool wxWebView::CanPaste ( ) const `[pure virtual]`

Returns true if data can be pasted.

Note

This always returns `true` on the OSX WebKit backend.

virtual bool wxWebView::CanRedo ( ) const `[pure virtual]`

Returns true if there is an action to redo.

virtual bool wxWebView::CanSetZoomType (wxWebViewZoomType *type*) const `[pure virtual]`

Retrieve whether the current HTML engine supports a zoom type.

Parameters

type	The zoom type to test.

Returns

Whether this type of zoom is supported by this HTML engine (and thus can be set through SetZoomType()).

virtual bool wxWebView::CanUndo ( ) const `[pure virtual]`

Returns true if there is an action to undo.

virtual void wxWebView::ClearHistory ( ) `[pure virtual]`

Clear the history, this will also remove the visible page.

virtual void wxWebView::ClearSelection ( ) `[pure virtual]`

Clears the current selection.

virtual void wxWebView::Copy ( ) [pure virtual]

Copies the current selection.

virtual bool wxWebView::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *url* = wxWebViewDefaultURLStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxWebViewNameStr) [pure virtual]

Creation function for two-step creation.

virtual void wxWebView::Cut ( ) [pure virtual]

Cuts the current selection.

virtual void wxWebView::DeleteSelection ( ) [pure virtual]

Deletes the current selection.

Note that for wxWEBVIEW_BACKEND_WEBKIT the selection must be editable, either through SetEditable or the correct HTML attribute.

virtual void wxWebView::EnableContextMenu (bool *enable* = true) [virtual]

Enable or disable the right click context menu.

By default the standard context menu is enabled, this method can be used to disable it or re-enable it later.

Since

2.9.5

virtual void wxWebView::EnableHistory (bool *enable* = true) [pure virtual]

Enable or disable the history.

This will also clear the history.

virtual long wxWebView::Find (const wxString & *text*, wxWebViewFindFlags *flags* = wxWEBVIEW_FIND_DEFAULT) [pure virtual]

Finds a phrase on the current page and if found, the control will scroll the phrase into view and select it.

Parameters

text	The phrase to search for.
flags	The flags for the search.

Returns

If search phrase was not found in combination with the flags then wxNOT_FOUND is returned. If called for the first time with search phrase then the total number of results will be returned. Then for every time its called with the same search phrase it will return the number of the current match.

Note

This function will restart the search if the flags `wxWEBVIEW_FIND_ENTIRE_WORD` or `wxWEBVIEW_↩ FIND_MATCH_CASE` are changed, since this will require a new search. To reset the search, for example resetting the highlights call the function with an empty search phrase. This always returns `wxNOT_FOUND` on the OSX WebKit backend.

Since

2.9.5

virtual wxVector<wxSharedPtr<wxWebViewHistoryItem> > wxWebView::GetBackwardHistory ( ) [pure virtual]

Returns a list of items in the back history.

The first item in the vector is the first page that was loaded by the control.

virtual wxString wxWebView::GetCurrentTitle ( ) const [pure virtual]

Get the title of the current web page, or its URL/path if title is not available.

virtual wxString wxWebView::GetCurrentURL ( ) const [pure virtual]

Get the URL of the currently displayed document.

virtual wxVector<wxSharedPtr<wxWebViewHistoryItem> > wxWebView::GetForwardHistory ( ) [pure virtual]

Returns a list of items in the forward history.

The first item in the vector is the next item in the history with respect to the currently loaded page.

virtual void* wxWebView::GetNativeBackend ( ) const [pure virtual]

Return the pointer to the native backend used by this control.

This method can be used to retrieve the pointer to the native rendering engine used by this control. The return value needs to be down-casted to the appropriate type depending on the platform: under Windows, it's a pointer to IWebBrowser2 interface, under OS X it's a WebView pointer and under GTK it's a WebKitWebView.

For example, you could set the WebKit options using this method:

```
#include <webkit/webkit.h>

#ifdef __WXGTK__
   WebKitWebView*
    wv = static_cast<WebKitWebView*>(m_window->GetNativeBackend());

   WebKitWebSettings* settings = webkit_web_view_get_settings(wv);
   g_object_set(G_OBJECT(settings),
                "enable-frame-flattening", TRUE,
                NULL);
#endif
```

Since

2.9.5

virtual wxString wxWebView::GetPageSource ( ) const `[pure virtual]`

Get the HTML source code of the currently displayed document.

Returns

The HTML source code, or an empty string if no page is currently shown.

virtual wxString wxWebView::GetPageText ( ) const `[pure virtual]`

Get the text of the current page.

virtual wxString wxWebView::GetSelectedSource ( ) const `[pure virtual]`

Returns the currently selected source, if any.

virtual wxString wxWebView::GetSelectedText ( ) const `[pure virtual]`

Returns the currently selected text, if any.

virtual wxWebViewZoom wxWebView::GetZoom ( ) const `[pure virtual]`

Get the zoom factor of the page.

Returns

The current level of zoom.

virtual wxWebViewZoomType wxWebView::GetZoomType ( ) const `[pure virtual]`

Get how the zoom factor is currently interpreted.

Returns

How the zoom factor is currently interpreted by the HTML engine.

virtual void wxWebView::GoBack ( ) `[pure virtual]`

Navigate back in the history of visited pages.

Only valid if CanGoBack() returns true.

virtual void wxWebView::GoForward ( ) `[pure virtual]`

Navigate forward in the history of visited pages.

Only valid if CanGoForward() returns true.

virtual bool wxWebView::HasSelection ( ) const `[pure virtual]`

Returns true if there is a current selection.

virtual bool wxWebView::IsBusy ( ) const `[pure virtual]`

Returns whether the web control is currently busy (e.g. loading a page).

virtual bool wxWebView::IsContextMenuEnabled ( ) const `[virtual]`

Returns true if a context menu will be shown on right click.

Since

2.9.5

virtual bool wxWebView::IsEditable ( ) const `[pure virtual]`

Returns whether the web control is currently editable.

virtual void wxWebView::LoadHistoryItem (wxSharedPtr< wxWebViewHistoryItem > *item*) `[pure virtual]`

Loads a history item.

virtual void wxWebView::LoadURL (const wxString & *url*) `[pure virtual]`

Load a web page from a URL.

Parameters

url	The URL of the page to be loaded.

Note

Web engines generally report errors asynchronously, so if you wish to know whether loading the URL was successful, register to receive navigation error events.

static wxWebView* wxWebView::New (const wxString & *backend* = `wxWebViewBackendDefault`) `[static]`

Factory function to create a new wxWebView with two-step creation, wxWebView::Create should be called on the returned object.

Parameters

backend	The backend web rendering engine to use. `wxWebViewBackendDefault`, `wxWebViewBackendIE` and `wxWebViewBackendWebKit` are predefined where appropriate.

Returns

The created wxWebView

Since

2.9.5

static wxWebView* wxWebView::New (wxWindow * *parent*, wxWindowID *id*, const wxString & *url* = wxWebViewDefaultURLStr, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, const wxString & *backend* = wxWebViewBackendDefault, long *style* = 0, const wxString & *name* = wxWebViewNameStr) [static]

Factory function to create a new wxWebView using a wxWebViewFactory.

Parameters

parent	Parent window for the control
id	ID of this control
url	Initial URL to load
pos	Position of the control
size	Size of the control
backend	The backend web rendering engine to use. `wxWebViewBackendDefault`, `wxWebViewBackendIE` and `wxWebViewBackendWebKit` are predefined where appropriate.
style	Window style. For generic window styles, please see wxWindow.
name	Window name.

Returns

The created wxWebView, or `NULL` if the requested backend is not available

Since

2.9.5

virtual void wxWebView::Paste ( ) [pure virtual]

Pastes the current data.

virtual void wxWebView::Print ( ) [pure virtual]

Opens a print dialog so that the user may print the currently displayed page.

virtual void wxWebView::Redo ( ) [pure virtual]

Redos the last action.

static void wxWebView::RegisterFactory (const wxString & *backend*, wxSharedPtr< wxWebViewFactory > *factory*) [static]

Allows the registering of new backend for wxWebView.

backend can be used as an argument to New().

Parameters

backend	The name for the new backend to be registered under
factory	A shared pointer to the factory which creates the appropriate backend.

Since

2.9.5

virtual void wxWebView::RegisterHandler (wxSharedPtr< wxWebViewHandler > *handler*) [pure virtual]

Registers a custom scheme handler.

Parameters

handler	A shared pointer to a wxWebHandler.

virtual void wxWebView::Reload (wxWebViewReloadFlags *flags* = wxWEBVIEW_RELOAD_DEFAULT) [pure virtual]

Reload the currently displayed URL.

Parameters

flags	A bit array that may optionally contain reload options.

virtual void wxWebView::RunScript (const wxString & *javascript*) [pure virtual]

Runs the given javascript code.

Note

When using wxWEBVIEW_BACKEND_IE you must wait for the current page to finish loading before calling RunScript().

virtual void wxWebView::SelectAll ( ) [pure virtual]

Selects the entire page.

virtual void wxWebView::SetEditable (bool *enable* = true) [pure virtual]

Set the editable property of the web control.

Enabling allows the user to edit the page even if the `contenteditable` attribute is not set. The exact capabilities vary with the backend being used.

virtual void wxWebView::SetPage (const wxString & *html*, const wxString & *baseUrl*) [pure virtual]

Set the displayed page source to the contents of the given string.

Parameters

html	The string that contains the HTML data to display.
baseUrl	URL assigned to the HTML data, to be used to resolve relative paths, for instance.

Note

When using `wxWEBVIEW_BACKEND_IE` you must wait for the current page to finish loading before calling SetPage(). The baseURL parameter is not used in this backend.

virtual void wxWebView::SetPage (wxInputStream & *html*, wxString *baseUrl*) [virtual]

Set the displayed page source to the contents of the given stream.

Parameters

html	The stream to read HTML data from.
baseUrl	URL assigned to the HTML data, to be used to resolve relative paths, for instance.

virtual void wxWebView::SetZoom (wxWebViewZoom *zoom*) [pure virtual]

Set the zoom factor of the page.

Parameters

zoom	How much to zoom (scale) the HTML document.

virtual void wxWebView::SetZoomType (wxWebViewZoomType *zoomType*) [pure virtual]

Set how to interpret the zoom factor.

Parameters

zoomType	How the zoom factor should be interpreted by the HTML engine.

Note

invoke CanSetZoomType() first, some HTML renderers may not support all zoom types.

virtual void wxWebView::Stop ( ) [pure virtual]

Stop the current page loading process, if any.

May trigger an error event of type `wxWEBVIEW_NAV_ERR_USER_CANCELLED`. TODO: make `wxWEBVIEW_NAV_ERR_USER_CANCELLED` errors uniform across ports.

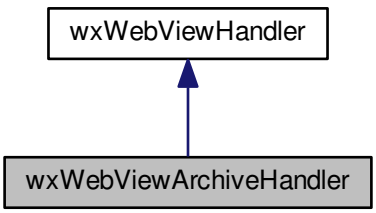
virtual void wxWebView::Undo ( ) [pure virtual]

Undos the last action.

21.848 wxWebViewArchiveHandler Class Reference

`#include <wx/webviewarchivehandler.h>`

Inheritance diagram for wxWebViewArchiveHandler:



21.848.1 Detailed Description

A custom handler for the file scheme which also supports loading from archives.

The syntax for wxWebViewArchiveHandler differs from virtual file systems in the rest of wxWidgets by using a syntax such as `scheme:///C:/example/docs.zip;protocol=zip/main.htm` Currently the only supported protocol is `zip`.

Since

2.9.3

Library: wxWebView

Category: WebView

See also

wxWebView, wxWebViewHandler

Public Member Functions

- wxWebViewArchiveHandler (const wxString &scheme)

  Constructor.

- virtual wxFSFile * GetFile (const wxString &uri)

21.848.2 Constructor & Destructor Documentation

wxWebViewArchiveHandler::wxWebViewArchiveHandler (const wxString & *scheme*)

Constructor.

21.848.3 Member Function Documentation

virtual wxFSFile* wxWebViewArchiveHandler::GetFile (const wxString & *uri*) `[virtual]`

Returns

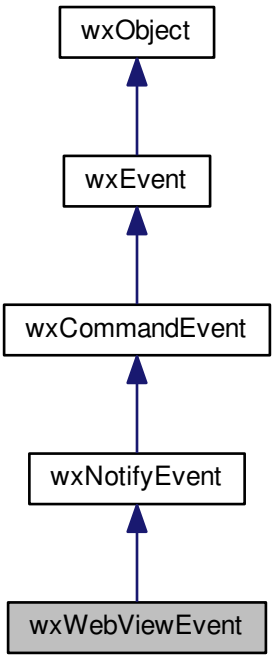
A pointer to the file represented by `uri`.

Implements wxWebViewHandler.

21.849 wxWebViewEvent Class Reference

```
#include <wx/webview.h>
```

Inheritance diagram for wxWebViewEvent:



21.849.1 Detailed Description

A navigation event holds information about events associated with wxWebView objects.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxWebViewEvent& event)

Event macros for events emitted by this class:

- EVT_WEBVIEW_NAVIGATING(id, func): Process a `wxEVT_WEBVIEW_NAVIGATING` event, generated before trying to get a resource. This event may be vetoed to prevent navigating to this resource. Note that if the displayed HTML document has several frames, one such event will be generated per frame.
- EVT_WEBVIEW_NAVIGATED(id, func): Process a `wxEVT_WEBVIEW_NAVIGATED` event generated after it was confirmed that a resource would be requested. This event may not be vetoed. Note that if the displayed HTML document has several frames, one such event will be generated per frame.
- EVT_WEBVIEW_LOADED(id, func): Process a `wxEVT_WEBVIEW_LOADED` event generated when the document is fully loaded and displayed. Note that if the displayed HTML document has several frames, one such event will be generated per frame.
- EVT_WEBVIEW_ERROR(id, func): Process a `wxEVT_WEBVIEW_ERROR` event generated when a navigation error occurs. The integer associated with this event will be a wxWebViewNavigationError item. The string associated with this event may contain a backend-specific more precise error message/code.

- EVT_WEBVIEW_NEWWINDOW(id, func): Process a `wxEVT_WEBVIEW_NEWWINDOW` event, generated when a new window is created. You must handle this event if you want anything to happen, for example to load the page in a new window or tab.
- EVT_WEBVIEW_TITLE_CHANGED(id, func): Process a `wxEVT_WEBVIEW_TITLE_CHANGED` event, generated when the page title changes. Use GetString to get the title.

Since

2.9.3

Library: wxWebView

Category: Events, WebView

See also

wxWebView

Public Member Functions

- wxWebViewEvent ()
- wxWebViewEvent (wxEventType type, int id, const wxString href, const wxString target)
- const wxString & GetTarget () const

  Get the name of the target frame which the url of this event has been or will be loaded into.
- const wxString & GetURL () const

  Get the URL being visited.

Additional Inherited Members

21.849.2 Constructor & Destructor Documentation

wxWebViewEvent::wxWebViewEvent ( )

wxWebViewEvent::wxWebViewEvent (wxEventType *type,* int *id,* const wxString *href,* const wxString *target*)

21.849.3 Member Function Documentation

const wxString& wxWebViewEvent::GetTarget ( ) const

Get the name of the target frame which the url of this event has been or will be loaded into.

This may return an empty string if the frame is not available.

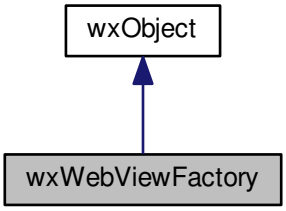
const wxString& wxWebViewEvent::GetURL ( ) const

Get the URL being visited.

21.850 wxWebViewFactory Class Reference

```
#include <wx/webview.h>
```

Inheritance diagram for wxWebViewFactory:



21.850.1 Detailed Description

An abstract factory class for creating wxWebView backends.

Each implementation of wxWebView should have its own factory.

Since

2.9.5

Library: wxWebView

Category: WebView

See also

wxWebView

Public Member Functions

- virtual wxWebView * Create ()=0

  Function to create a new wxWebView with two-step creation, wxWebView::Create should be called on the returned object.

- virtual wxWebView * Create (wxWindow *parent, wxWindowID id, const wxString &url=wxWebViewDefaultURLStr, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxWebViewNameStr)=0

  Function to create a new wxWebView with parameters.

Additional Inherited Members

21.850.2 Member Function Documentation

virtual wxWebView* wxWebViewFactory::Create ( ) [pure virtual]

Function to create a new wxWebView with two-step creation, wxWebView::Create should be called on the returned object.

Returns

the created wxWebView

virtual wxWebView* wxWebViewFactory::Create (wxWindow * *parent*, wxWindowID *id*, const wxString & *url* = `wxWebViewDefaultURLStr`, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = `0`, const wxString & *name* = `wxWebViewNameStr`) `[pure virtual]`

Function to create a new wxWebView with parameters.

Parameters

parent	Parent window for the control
id	ID of this control
url	Initial URL to load
pos	Position of the control
size	Size of the control
style	Window style. For generic window styles, please see wxWindow.
name	Window name.

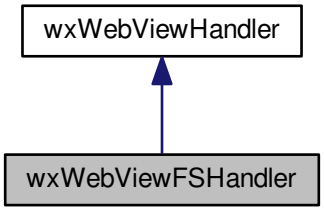
Returns

the created wxWebView

21.851 wxWebViewFSHandler Class Reference

`#include <wx/webviewfshandler.h>`

Inheritance diagram for wxWebViewFSHandler:



21.851.1 Detailed Description

A wxWebView file system handler to support standard wxFileSystem protocols of the form `example:page.htm` The handler allows wxWebView to use wxFileSystem in a similar fashion to its use with wxHtml.

The wxMemoryFSHandler documentation gives an example of how it may be used.

Since

2.9.5

Library: wxWebView

Category: WebView

See also

wxWebView, wxWebViewHandler, wxWebViewArchiveHandler

Public Member Functions

- wxWebViewFSHandler (const wxString &scheme)

  Constructor.
- virtual wxFSFile * GetFile (const wxString &uri)

21.851.2 Constructor & Destructor Documentation

wxWebViewFSHandler::wxWebViewFSHandler (const wxString & *scheme*)

Constructor.

21.851.3 Member Function Documentation

virtual wxFSFile* wxWebViewFSHandler::GetFile (const wxString & *uri*) `[virtual]`

Returns

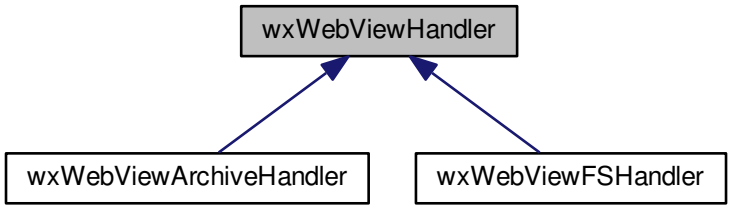
A pointer to the file represented by `uri`.

Implements wxWebViewHandler.

21.852 wxWebViewHandler Class Reference

`#include <wx/webview.h>`

Inheritance diagram for wxWebViewHandler:



21.852.1 Detailed Description

The base class for handling custom schemes in wxWebView, for example to allow virtual file system support.

Since

2.9.3

Library: wxWebView

Category: WebView

See also

wxWebView

Public Member Functions

- wxWebViewHandler (const wxString &scheme)

  Constructor.
- virtual wxFSFile * GetFile (const wxString &uri)=0
- virtual wxString GetName () const

21.852.2 Constructor & Destructor Documentation

wxWebViewHandler::wxWebViewHandler (const wxString & *scheme*)

Constructor.

Takes the name of the scheme that will be handled by this class for example `file` or `zip`.

21.852.3 Member Function Documentation

virtual wxFSFile* wxWebViewHandler::GetFile (const wxString & *uri*) `[pure virtual]`

Returns

A pointer to the file represented by `uri`.

Implemented in wxWebViewArchiveHandler, and wxWebViewFSHandler.

virtual wxString wxWebViewHandler::GetName ( ) const `[virtual]`

Returns

The name of the scheme, as passed to the constructor.

21.853 wxWebViewHistoryItem Class Reference

`#include <wx/webview.h>`

21.853.1 Detailed Description

A simple class that contains the URL and title of an element of the history of a wxWebView.

Since

2.9.3

Library: wxWebView

Category: WebView

See also

wxWebView

Public Member Functions

- wxWebViewHistoryItem (const wxString &url, const wxString &title)

  Construtor.

- wxString GetUrl ()
- wxString GetTitle ()

21.853.2 Constructor & Destructor Documentation

wxWebViewHistoryItem::wxWebViewHistoryItem (const wxString & *url,* const wxString & *title*)

Construtor.

21.853.3 Member Function Documentation

wxString wxWebViewHistoryItem::GetTitle ( )

Returns

The title of the page.

wxString wxWebViewHistoryItem::GetUrl ( )

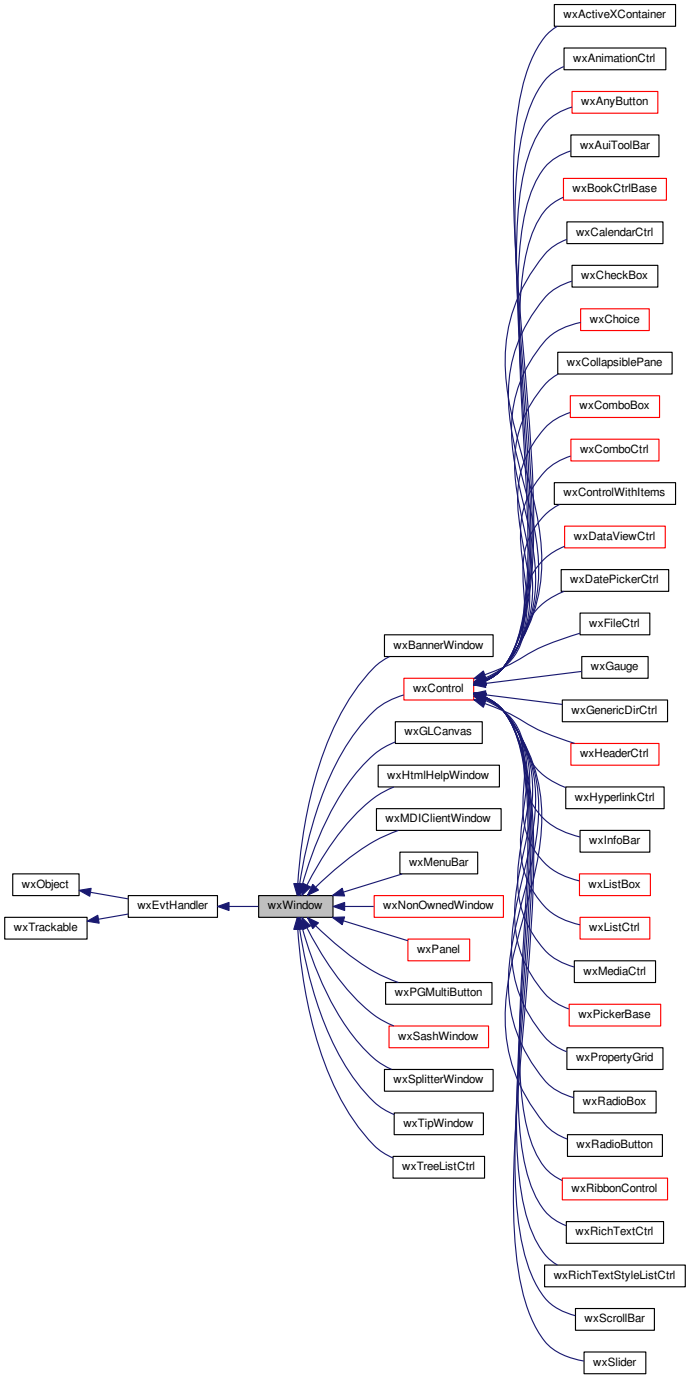
Returns

The url of the page.

21.854 wxWindow Class Reference

`#include <wx/window.h>`

Inheritance diagram for wxWindow:



21.854.1 Detailed Description

wxWindow is the base class for all windows and represents any visible object on screen.

All controls, top level windows and so on are windows. Sizers and device contexts are not, however, as they don't appear on screen themselves.

Please note that all children of the window will be deleted automatically by the destructor before the window itself is deleted which means that you don't have to worry about deleting them manually. Please see the window deletion overview for more information.

Also note that in this, and many others, wxWidgets classes some `GetXXX()` methods may be overloaded (as, for example, wxWindow::GetSize or wxWindow::GetClientSize). In this case, the overloads are non-virtual because having multiple virtual functions with the same name results in a virtual function name hiding at the derived class level (in English, this means that the derived class has to override all overloaded variants if it overrides any of them). To allow overriding them in the derived class, wxWidgets uses a unique protected virtual `DoGetXXX()` method and all `GetXXX()` ones are forwarded to it, so overriding the former changes the behaviour of the latter.

Styles

This class supports the following styles:

- wxBORDER_DEFAULT: The window class will decide the kind of border to show, if any.
- wxBORDER_SIMPLE: Displays a thin border around the window. wxSIMPLE_BORDER is the old name for this style.
- wxBORDER_SUNKEN: Displays a sunken border. wxSUNKEN_BORDER is the old name for this style.
- wxBORDER_RAISED: Displays a raised border. wxRAISED_BORDER is the old name for this style.
- wxBORDER_STATIC: Displays a border suitable for a static control. wxSTATIC_BORDER is the old name for this style. Windows only.
- wxBORDER_THEME: Displays a native border suitable for a control, on the current platform. On Windows XP or Vista, this will be a themed border; on most other platforms a sunken border will be used. For more information for themed borders on Windows, please see Themed borders on Windows.
- wxBORDER_NONE: Displays no border, overriding the default border style for the window. wxNO_BORDER is the old name for this style.
- wxBORDER_DOUBLE: This style is obsolete and should not be used.
- wxTRANSPARENT_WINDOW: The window is transparent, that is, it will not receive paint events. Windows only.
- wxTAB_TRAVERSAL: Use this to enable tab traversal for non-dialog windows.
- wxWANTS_CHARS: Use this to indicate that the window wants to get all char/key events for all keys - even for keys like TAB or ENTER which are usually used for dialog navigation and which wouldn't be generated without this style. If you need to use this style in order to get the arrows or etc., but would still like to have normal keyboard navigation take place, you should call Navigate in response to the key events for Tab and Shift-Tab.
- wxNO_FULL_REPAINT_ON_RESIZE: On Windows, this style used to disable repainting the window completely when its size is changed. Since this behaviour is now the default, the style is now obsolete and no longer has an effect.
- wxVSCROLL: Use this style to enable a vertical scrollbar. Notice that this style cannot be used with native controls which don't support scrollbars nor with top-level windows in most ports.
- wxHSCROLL: Use this style to enable a horizontal scrollbar. The same limitations as for wxVSCROLL apply to this style.

- wxALWAYS_SHOW_SB: If a window has scrollbars, disable them instead of hiding them when they are not needed (i.e. when the size of the window is big enough to not require the scrollbars to navigate it). This style is currently implemented for wxMSW, wxGTK and wxUniversal and does nothing on the other platforms.
- wxCLIP_CHILDREN: Use this style to eliminate flicker caused by the background being repainted, then children being painted over them. Windows only.
- wxFULL_REPAINT_ON_RESIZE: Use this style to force a complete redraw of the window whenever it is resized instead of redrawing just the part of the window affected by resizing. Note that this was the behaviour by default before 2.5.1 release and that if you experience redraw problems with code which previously used to work you may want to try this. Currently this style applies on GTK+ 2 and Windows only, and full repainting is always done on other platforms.

Extra Styles

This class supports the following extra styles:

- wxWS_EX_VALIDATE_RECURSIVELY: By default, wxWindow::Validate(), wxWindow::TransferDataTo() and wxWindow::TransferDataFromWindow() only work on direct children of the window (compatible behaviour). Set this flag to make them recursively descend into all subwindows.
- wxWS_EX_BLOCK_EVENTS: wxCommandEvents and the objects of the derived classes are forwarded to the parent window and so on recursively by default. Using this flag for the given window allows to block this propagation at this window, i.e. prevent the events from being propagated further upwards. Dialogs have this flag on by default for the reasons explained in the Events and Event Handling.
- wxWS_EX_TRANSIENT: Don't use this window as an implicit parent for the other windows: this must be used with transient windows as otherwise there is the risk of creating a dialog/frame with this window as a parent, which would lead to a crash if the parent were destroyed before the child.
- wxWS_EX_CONTEXTHELP: Under Windows, puts a query button on the caption. When pressed, Windows will go into a context-sensitive help mode and wxWidgets will send a `wxEVT_HELP` event if the user clicked on an application window. This style cannot be used (because of the underlying native behaviour) together with `wxMAXIMIZE_BOX` or `wxMINIMIZE_BOX`, so these two styles are automatically turned off if this one is used.
- wxWS_EX_PROCESS_IDLE: This window should always process idle events, even if the mode set by wxIdleEvent::SetMode is `wxIDLE_PROCESS_SPECIFIED`.
- wxWS_EX_PROCESS_UI_UPDATES: This window should always process UI update events, even if the mode set by wxUpdateUIEvent::SetMode is `wxUPDATE_UI_PROCESS_SPECIFIED`.

Events emitted by this class

Event macros for events emitted by this class:

- EVT_ACTIVATE(id, func): Process a `wxEVT_ACTIVATE` event. See wxActivateEvent.
- EVT_CHILD_FOCUS(func): Process a `wxEVT_CHILD_FOCUS` event. See wxChildFocusEvent.
- EVT_CONTEXT_MENU(func): A right click (or other context menu command depending on platform) has been detected. See wxContextMenuEvent.
- EVT_HELP(id, func): Process a `wxEVT_HELP` event. See wxHelpEvent.
- EVT_HELP_RANGE(id1, id2, func): Process a `wxEVT_HELP` event for a range of ids. See wxHelpEvent.
- EVT_DROP_FILES(func): Process a `wxEVT_DROP_FILES` event. See wxDropFilesEvent.
- EVT_ERASE_BACKGROUND(func): Process a `wxEVT_ERASE_BACKGROUND` event. See wxEraseEvent.

- EVT_SET_FOCUS(func): Process a `wxEVT_SET_FOCUS` event. See wxFocusEvent.
- EVT_KILL_FOCUS(func): Process a `wxEVT_KILL_FOCUS` event. See wxFocusEvent.
- EVT_IDLE(func): Process a `wxEVT_IDLE` event. See wxIdleEvent.
- EVT_JOY_*(func): Processes joystick events. See wxJoystickEvent.
- EVT_KEY_DOWN(func): Process a `wxEVT_KEY_DOWN` event (any key has been pressed). See wxKeyEvent.
- EVT_KEY_UP(func): Process a `wxEVT_KEY_UP` event (any key has been released). See wxKeyEvent.
- EVT_CHAR(func): Process a `wxEVT_CHAR` event. See wxKeyEvent.
- EVT_CHAR_HOOK(func): Process a `wxEVT_CHAR_HOOK` event. See wxKeyEvent.
- EVT_MOUSE_CAPTURE_LOST(func): Process a `wxEVT_MOUSE_CAPTURE_LOST` event. See wxMouseCaptureLostEvent.
- EVT_MOUSE_CAPTURE_CHANGED(func): Process a `wxEVT_MOUSE_CAPTURE_CHANGED` event. See wxMouseCaptureChangedEvent.
- EVT_MOUSE_*(func): See wxMouseEvent.
- EVT_PAINT(func): Process a `wxEVT_PAINT` event. See wxPaintEvent.
- EVT_POWER_*(func): The system power state changed. See wxPowerEvent.
- EVT_SCROLLWIN_*(func): Process scroll events. See wxScrollWinEvent.
- EVT_SET_CURSOR(func): Process a `wxEVT_SET_CURSOR` event. See wxSetCursorEvent.
- EVT_SIZE(func): Process a `wxEVT_SIZE` event. See wxSizeEvent.
- EVT_SYS_COLOUR_CHANGED(func): Process a `wxEVT_SYS_COLOUR_CHANGED` event. See wxSysColourChangedEvent.

Library: wxCore

Category: Miscellaneous Windows

See also

Events and Event Handling, Window Sizing Overview

Classes

- class ChildrenRepositioningGuard

  Helper for ensuring EndRepositioningChildren() is called correctly.

Public Member Functions

- wxWindow ()

  Default constructor.
- wxWindow (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

  Constructs a window, which can be a child of a frame, dialog or any other non-control window.
- virtual ~wxWindow ()

  Destructor.

- bool Create (wxWindow *parent, wxWindowID id, const wxPoint &pos=wxDefaultPosition, const wxSize &size=wxDefaultSize, long style=0, const wxString &name=wxPanelNameStr)

Focus functions

See also the static function FindFocus().

- virtual bool AcceptsFocus () const
  This method may be overridden in the derived classes to return false to indicate that this control doesn't accept input at all (i.e. behaves like e.g. wxStaticText) and so doesn't need focus.
- virtual bool AcceptsFocusFromKeyboard () const
  This method may be overridden in the derived classes to return false to indicate that while this control can, in principle, have focus if the user clicks it with the mouse, it shouldn't be included in the TAB traversal chain when using the keyboard.
- virtual bool AcceptsFocusRecursively () const
  Overridden to indicate whether this window or one of its children accepts focus.
- bool IsFocusable () const
  Can this window itself have focus?
- bool CanAcceptFocus () const
  Can this window have focus right now?
- bool CanAcceptFocusFromKeyboard () const
  Can this window be assigned focus from keyboard right now?
- virtual bool HasFocus () const
  Returns true if the window (or in case of composite controls, its main child window) has focus.
- virtual void SetCanFocus (bool canFocus)
  This method is only implemented by ports which have support for native TAB traversal (such as GTK+ 2.0).
- virtual void SetFocus ()
  This sets the window to receive keyboard input.
- virtual void SetFocusFromKbd ()
  This function is called by wxWidgets keyboard navigation code when the user gives the focus to this window from keyboard (e.g. using `TAB` key).

Child management functions

- virtual void AddChild (wxWindow *child)
  Adds a child window.
- bool DestroyChildren ()
  Destroys all children of a window.
- wxWindow * FindWindow (long id) const
  Find a child of this window, by id.
- wxWindow * FindWindow (const wxString &name) const
  Find a child of this window, by name.
- wxWindowList & GetChildren ()
  Returns a reference to the list of the window's children.
- const wxWindowList & GetChildren () const
  Returns a const reference to the list of the window's children.
- virtual void RemoveChild (wxWindow *child)
  Removes a child window.

Sibling and parent management functions

- wxWindow * GetGrandParent () const
  Returns the grandparent of a window, or NULL if there isn't one.
- wxWindow * GetNextSibling () const
  Returns the next window after this one among the parent's children or NULL if this window is the last child.
- wxWindow * GetParent () const
  Returns the parent of the window, or NULL if there is no parent.
- wxWindow * GetPrevSibling () const
  Returns the previous window before this one among the parent's children or NULL if this window is the first child.
- bool IsDescendant (wxWindowBase *win) const

Check if the specified window is a descendant of this one.

- virtual bool Reparent (wxWindow *newParent)

  Reparents the window, i.e. the window will be removed from its current parent window (e.g.

Scrolling and scrollbars functions

Note that these methods don't work with native controls which don't use wxWidgets scrolling framework (i.e. don't derive from wxScrolledWindow).

- virtual void AlwaysShowScrollbars (bool hflag=true, bool vflag=true)

  Call this function to force one or both scrollbars to be always shown, even if the window is big enough to show its entire contents without scrolling.
- virtual int GetScrollPos (int orientation) const

  Returns the built-in scrollbar position.
- virtual int GetScrollRange (int orientation) const

  Returns the built-in scrollbar range.
- virtual int GetScrollThumb (int orientation) const

  Returns the built-in scrollbar thumb size.
- bool CanScroll (int orient) const

  Returns true if this window can have a scroll bar in this orientation.
- bool HasScrollbar (int orient) const

  Returns true if this window currently has a scroll bar for this orientation.
- virtual bool IsScrollbarAlwaysShown (int orient) const

  Return whether a scrollbar is always shown.
- virtual bool ScrollLines (int lines)

  Scrolls the window by the given number of lines down (if lines is positive) or up.
- virtual bool ScrollPages (int pages)

  Scrolls the window by the given number of pages down (if pages is positive) or up.
- virtual void ScrollWindow (int dx, int dy, const wxRect *rect=NULL)

  Physically scrolls the pixels in the window and move child windows accordingly.
- bool LineUp ()

  Same as ScrollLines (-1).
- bool LineDown ()

  Same as ScrollLines (1).
- bool PageUp ()

  Same as ScrollPages (-1).
- bool PageDown ()

  Same as ScrollPages (1).
- virtual void SetScrollPos (int orientation, int pos, bool refresh=true)

  Sets the position of one of the built-in scrollbars.
- virtual void SetScrollbar (int orientation, int position, int thumbSize, int range, bool refresh=true)

  Sets the scrollbar properties of a built-in scrollbar.

Sizing functions

See also the protected functions DoGetBestSize() and DoGetBestClientSize().

- bool BeginRepositioningChildren ()

  Prepare for changing positions of multiple child windows.
- void EndRepositioningChildren ()

  Fix child window positions after setting all of them at once.
- void CacheBestSize (const wxSize &size) const

  Sets the cached best size value.
- virtual wxSize ClientToWindowSize (const wxSize &size) const

  Converts client area size size to corresponding window size.
- virtual wxSize WindowToClientSize (const wxSize &size) const

  Converts window size size to corresponding client area size In other words, the returned value is what would GetClientSize() return if this window had given window size.
- virtual void Fit ()

  Sizes the window so that it fits around its subwindows.

- virtual void FitInside ()

  Similar to Fit(), but sizes the interior (virtual) size of a window.
- wxSize GetBestSize () const

  This functions returns the best acceptable minimal size for the window.
- int GetBestHeight (int width) const

  Returns the best height needed by this window if it had the given width.
- int GetBestWidth (int height) const

  Returns the best width needed by this window if it had the given height.
- void GetClientSize (int ∗width, int ∗height) const

  Returns the size of the window 'client area' in pixels.
- wxSize GetClientSize () const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- virtual wxSize GetEffectiveMinSize () const

  Merges the window's best size into the min size and returns the result.
- virtual wxSize GetMaxClientSize () const

  Returns the maximum size of window's client area.
- virtual wxSize GetMaxSize () const

  Returns the maximum size of the window.
- virtual wxSize GetMinClientSize () const

  Returns the minimum size of window's client area, an indication to the sizer layout mechanism that this is the minimum required size of its client area.
- virtual wxSize GetMinSize () const

  Returns the minimum size of the window, an indication to the sizer layout mechanism that this is the minimum required size.
- int GetMinWidth () const

  Returns the horizontal component of window minimal size.
- int GetMinHeight () const

  Returns the vertical component of window minimal size.
- int GetMaxWidth () const

  Returns the horizontal component of window maximal size.
- int GetMaxHeight () const

  Returns the vertical component of window maximal size.
- void GetSize (int ∗width, int ∗height) const

  Returns the size of the entire window in pixels, including title bar, border, scrollbars, etc.
- wxSize GetSize () const

  See the GetSize(int∗,int∗) overload for more info.
- wxSize GetVirtualSize () const

  This gets the virtual size of the window in pixels.
- void GetVirtualSize (int ∗width, int ∗height) const

  Like the other GetVirtualSize() overload but uses pointers instead.
- virtual wxSize GetBestVirtualSize () const

  Return the largest of ClientSize and BestSize (as determined by a sizer, interior children, or other means)
- virtual double GetContentScaleFactor () const

  Returns the magnification of the backing store of this window, eg 2.0 for a window on a retina screen.
- virtual wxSize GetWindowBorderSize () const

  Returns the size of the left/right and top/bottom borders of this window in x and y components of the result respectively.
- virtual bool InformFirstDirection (int direction, int size, int availableOtherDir)

  wxSizer and friends use this to give a chance to a component to recalc its min size once one of the final size components is known.
- void InvalidateBestSize ()

  Resets the cached best size value so it will be recalculated the next time it is needed.
- void PostSizeEvent ()

  Posts a size event to the window.
- void PostSizeEventToParent ()

  Posts a size event to the parent of this window.
- virtual void SendSizeEvent (int flags=0)

  This function sends a dummy size event to the window allowing it to re-layout its children positions.

- void SendSizeEventToParent (int flags=0)

  Safe wrapper for GetParent()->SendSizeEvent().
- void SetClientSize (int width, int height)

  This sets the size of the window client area in pixels.
- void SetClientSize (const wxSize &size)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetClientSize (const wxRect &rect)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetContainingSizer (wxSizer *sizer)

  This normally does not need to be called by user code.
- void SetInitialSize (const wxSize &size=wxDefaultSize)

  A smart SetSize that will fill in default size components with the window's best size values.
- virtual void SetMaxClientSize (const wxSize &size)

  Sets the maximum client size of the window, to indicate to the sizer layout mechanism that this is the maximum possible size of its client area.
- virtual void SetMaxSize (const wxSize &size)

  Sets the maximum size of the window, to indicate to the sizer layout mechanism that this is the maximum possible size.
- virtual void SetMinClientSize (const wxSize &size)

  Sets the minimum client size of the window, to indicate to the sizer layout mechanism that this is the minimum required size of window's client area.
- virtual void SetMinSize (const wxSize &size)

  Sets the minimum size of the window, to indicate to the sizer layout mechanism that this is the minimum required size.
- void SetSize (int x, int y, int width, int height, int sizeFlags=wxSIZE_AUTO)

  Sets the size of the window in pixels.
- void SetSize (const wxRect &rect)

  Sets the size of the window in pixels.
- void SetSize (const wxSize &size)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetSize (int width, int height)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- virtual void SetSizeHints (const wxSize &minSize, const wxSize &maxSize=wxDefaultSize, const wxSize &incSize=wxDefaultSize)

  Use of this function for windows which are not toplevel windows (such as wxDialog or wxFrame) is discouraged.
- virtual void SetSizeHints (int minW, int minH, int maxW=-1, int maxH=-1, int incW=-1, int incH=-1)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void SetVirtualSize (int width, int height)

  Sets the virtual size of the window in pixels.
- void SetVirtualSize (const wxSize &size)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Positioning functions

- void Center (int dir=wxBOTH)

  A synonym for Centre().
- void CenterOnParent (int dir=wxBOTH)

  A synonym for CentreOnParent().
- void Centre (int direction=wxBOTH)

  Centres the window.
- void CentreOnParent (int direction=wxBOTH)

  Centres the window on its parent.
- void GetPosition (int *x, int *y) const

This gets the position of the window in pixels, relative to the parent window for the child windows or relative to the display origin for the top level windows.

- wxPoint GetPosition () const

  This gets the position of the window in pixels, relative to the parent window for the child windows or relative to the display origin for the top level windows.
- wxRect GetRect () const

  Returns the position and size of the window as a wxRect object.
- void GetScreenPosition (int *x, int *y) const

  Returns the window position in screen coordinates, whether the window is a child window or a top level one.
- wxPoint GetScreenPosition () const

  Returns the window position in screen coordinates, whether the window is a child window or a top level one.
- wxRect GetScreenRect () const

  Returns the position and size of the window on the screen as a wxRect object.
- virtual wxPoint GetClientAreaOrigin () const

  Get the origin of the client area of the window relative to the window top left corner (the client area may be shifted because of the borders, scrollbars, other decorations...)
- wxRect GetClientRect () const

  Get the client rectangle in window (i.e. client) coordinates.
- void Move (int x, int y, int flags=wxSIZE_USE_EXISTING)

  Moves the window to the given position.
- void Move (const wxPoint &pt, int flags=wxSIZE_USE_EXISTING)

  Moves the window to the given position.
- void SetPosition (const wxPoint &pt)

  A synonym for Centre().

Coordinate conversion functions

- void ClientToScreen (int *x, int *y) const

  Converts to screen coordinates from coordinates relative to this window.
- wxPoint ClientToScreen (const wxPoint &pt) const

  Converts to screen coordinates from coordinates relative to this window.
- wxPoint ConvertDialogToPixels (const wxPoint &pt) const

  Converts a point or size from dialog units to pixels.
- wxSize ConvertDialogToPixels (const wxSize &sz) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxPoint ConvertPixelsToDialog (const wxPoint &pt) const

  Converts a point or size from pixels to dialog units.
- wxSize ConvertPixelsToDialog (const wxSize &sz) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void ScreenToClient (int *x, int *y) const

  Converts from screen to client window coordinates.
- wxPoint ScreenToClient (const wxPoint &pt) const

  Converts from screen to client window coordinates.

Drawing-related functions

- virtual void ClearBackground ()

  Clears the window by filling it with the current background colour.
- void Freeze ()

  Freezes the window or, in other words, prevents any updates from taking place on screen, the window is not redrawn at all.
- void Thaw ()

  Re-enables window updating after a previous call to Freeze().
- bool IsFrozen () const

  Returns true if the window is currently frozen by a call to Freeze().
- wxColour GetBackgroundColour () const

  Returns the background colour of the window.

- virtual wxBackgroundStyle GetBackgroundStyle () const

  Returns the background style of the window.
- virtual int GetCharHeight () const

  Returns the character height for this window.
- virtual int GetCharWidth () const

  Returns the average character width for this window.
- virtual wxVisualAttributes GetDefaultAttributes () const

  Currently this is the same as calling wxWindow::GetClassDefaultAttributes(wxWindow::GetWindowVariant()).
- wxFont GetFont () const

  Returns the font for this window.
- wxColour GetForegroundColour () const

  Returns the foreground colour of the window.
- void GetTextExtent (const wxString &string, int ∗w, int ∗h, int ∗descent=NULL, int ∗externalLeading=NULL, const wxFont ∗font=NULL) const

  Gets the dimensions of the string as it would be drawn on the window with the currently selected font.
- wxSize GetTextExtent (const wxString &string) const

  Gets the dimensions of the string as it would be drawn on the window with the currently selected font.
- const wxRegion & GetUpdateRegion () const

  Returns the region specifying which parts of the window have been damaged.
- wxRect GetUpdateClientRect () const

  Get the update rectangle bounding box in client coords.
- virtual bool HasTransparentBackground ()

  Returns true if this window background is transparent (as, for example, for wxStaticText) and should show the parent window background.
- virtual void Refresh (bool eraseBackground=true, const wxRect ∗rect=NULL)

  Causes this window, and all of its children recursively (except under wxGTK1 where this is not implemented), to be repainted.
- void RefreshRect (const wxRect &rect, bool eraseBackground=true)

  Redraws the contents of the given rectangle: only the area inside it will be repainted.
- virtual void Update ()

  Calling this method immediately repaints the invalidated area of the window and all of its children recursively (this normally only happens when the flow of control returns to the event loop).
- virtual bool SetBackgroundColour (const wxColour &colour)

  Sets the background colour of the window.
- virtual bool SetBackgroundStyle (wxBackgroundStyle style)

  Sets the background style of the window.
- virtual bool IsTransparentBackgroundSupported (wxString ∗reason=NULL) const

  Checks whether using transparent background might work.
- virtual bool SetFont (const wxFont &font)

  Sets the font for this window.
- virtual bool SetForegroundColour (const wxColour &colour)

  Sets the foreground colour of the window.
- void SetOwnBackgroundColour (const wxColour &colour)

  Sets the background colour of the window but prevents it from being inherited by the children of this window.
- bool InheritsBackgroundColour () const

  Return true if this window inherits the background colour from its parent.
- bool UseBgCol () const

  Return true if a background colour has been set for this window.
- void SetOwnFont (const wxFont &font)

  Sets the font of the window but prevents it from being inherited by the children of this window.
- void SetOwnForegroundColour (const wxColour &colour)

  Sets the foreground colour of the window but prevents it from being inherited by the children of this window.
- void SetPalette (const wxPalette &pal)
- virtual bool ShouldInheritColours () const

  Return true from here to allow the colours of this window to be changed by InheritAttributes().
- virtual void SetThemeEnabled (bool enable)

  This function tells a window if it should use the system's "theme" code to draw the windows' background instead of its own background drawing code.
- virtual bool GetThemeEnabled () const

Clears the window by filling it with the current background colour.

- virtual bool CanSetTransparent ()

  Returns true if the system supports transparent windows and calling SetTransparent() may succeed.
- virtual bool SetTransparent (wxByte alpha)

  Set the transparency of the window.

Event-handling functions

wxWindow allows you to build a (sort of) stack of event handlers which can be used to override the window's own event handling.

- wxEvtHandler * GetEventHandler () const

  Returns the event handler for this window.
- bool HandleAsNavigationKey (const wxKeyEvent &event)

  This function will generate the appropriate call to Navigate() if the key event is one normally used for keyboard navigation and return true in this case.
- bool HandleWindowEvent (wxEvent &event) const

  Shorthand for:
- bool ProcessWindowEvent (wxEvent &event)

  Convenient wrapper for ProcessEvent().
- bool ProcessWindowEventLocally (wxEvent &event)

  Wrapper for wxEvtHandler::ProcessEventLocally().
- wxEvtHandler * PopEventHandler (bool deleteHandler=false)

  Removes and returns the top-most event handler on the event handler stack.
- void PushEventHandler (wxEvtHandler *handler)

  Pushes this event handler onto the event stack for the window.
- bool RemoveEventHandler (wxEvtHandler *handler)

  Find the given handler in the windows event handler stack and removes (but does not delete) it from the stack.
- void SetEventHandler (wxEvtHandler *handler)

  Sets the event handler for this window.
- virtual void SetNextHandler (wxEvtHandler *handler)

  wxWindows cannot be used to form event handler chains; this function thus will assert when called.
- virtual void SetPreviousHandler (wxEvtHandler *handler)

  wxWindows cannot be used to form event handler chains; this function thus will assert when called.

Window styles functions

- long GetExtraStyle () const

  Returns the extra style bits for the window.
- virtual long GetWindowStyleFlag () const

  Gets the window style that was passed to the constructor or Create() method.
- long GetWindowStyle () const

  See GetWindowStyleFlag() for more info.
- bool HasExtraStyle (int exFlag) const

  Returns true if the window has the given exFlag bit set in its extra styles.
- bool HasFlag (int flag) const

  Returns true if the window has the given flag bit set.
- virtual void SetExtraStyle (long exStyle)

  Sets the extra style bits for the window.
- virtual void SetWindowStyleFlag (long style)

  Sets the style of the window.
- void SetWindowStyle (long style)

  See SetWindowStyleFlag() for more info.
- bool ToggleWindowStyle (int flag)

  Turns the given flag on if it's currently turned off and vice versa.

Tab order functions

- void MoveAfterInTabOrder (wxWindow *win)

Moves this window in the tab navigation order after the specified win.

- void MoveBeforeInTabOrder (wxWindow *win)

  Same as MoveAfterInTabOrder() except that it inserts this window just before win instead of putting it right after it.
- bool Navigate (int flags=wxNavigationKeyEvent::IsForward)

  Performs a keyboard navigation action starting from this window.
- bool NavigateIn (int flags=wxNavigationKeyEvent::IsForward)

  Performs a keyboard navigation action inside this window.

Z order functions

- virtual void Lower ()

  Lowers the window to the bottom of the window hierarchy (Z-order).
- virtual void Raise ()

  Raises the window to the top of the window hierarchy (Z-order).

Window status functions

- bool Hide ()

  Equivalent to calling wxWindow::Show(false).
- virtual bool HideWithEffect (wxShowEffect effect, unsigned int timeout=0)

  This function hides a window, like Hide(), but using a special visual effect if possible.
- bool IsEnabled () const

  Returns true if the window is enabled, i.e. if it accepts user input, false otherwise.
- bool IsExposed (int x, int y) const

  Returns true if the given point or rectangle area has been exposed since the last repaint.
- bool IsExposed (wxPoint &pt) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool IsExposed (int x, int y, int w, int h) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool IsExposed (wxRect &rect) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- virtual bool IsShown () const

  Returns true if the window is shown, false if it has been hidden.
- virtual bool IsShownOnScreen () const

  Returns true if the window is physically visible on the screen, i.e. it is shown and all its parents up to the toplevel window are shown as well.
- bool Disable ()

  Disables the window.
- virtual bool Enable (bool enable=true)

  Enable or disable the window for user input.
- virtual bool Show (bool show=true)

  Shows or hides the window.
- virtual bool ShowWithEffect (wxShowEffect effect, unsigned int timeout=0)

  This function shows a window, like Show(), but using a special visual effect if possible.

Context-sensitive help functions

- wxString GetHelpText () const

  Gets the help text to be used as context-sensitive help for this window.
- void SetHelpText (const wxString &helpText)

  Sets the help text to be used as context-sensitive help for this window.
- virtual wxString GetHelpTextAtPoint (const wxPoint &point, wxHelpEvent::Origin origin) const

  Gets the help text to be used as context-sensitive help for this window.
- wxToolTip * GetToolTip () const

  Get the associated tooltip or NULL if none.
- wxString GetToolTipText () const

Get the text of the associated tooltip or empty string if none.

- void SetToolTip (const wxString &tipString)

  Attach a tooltip to the window.
- void SetToolTip (wxToolTip *tip)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- void UnsetToolTip ()

  Unset any existing tooltip.

Popup/context menu functions

- int GetPopupMenuSelectionFromUser (wxMenu &menu, const wxPoint &pos=wxDefaultPosition)

  This function shows a popup menu at the given position in this window and returns the selected id.
- int GetPopupMenuSelectionFromUser (wxMenu &menu, int x, int y)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool PopupMenu (wxMenu *menu, const wxPoint &pos=wxDefaultPosition)

  Pops up the given menu at the specified coordinates, relative to this window, and returns control when the user has dismissed the menu.
- bool PopupMenu (wxMenu *menu, int x, int y)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- virtual wxValidator * GetValidator ()

  Validator functions.
- virtual void SetValidator (const wxValidator &validator)

  Deletes the current validator (if any) and sets the window validator, having called wxValidator::Clone to create a new validator of this type.
- virtual bool TransferDataFromWindow ()

  Transfers values from child controls to data areas specified by their validators.
- virtual bool TransferDataToWindow ()

  Transfers values to child controls from data areas specified by their validators.
- virtual bool Validate ()

  Validates the current values of the child controls using their validators.

wxWindow properties functions

- wxWindowID GetId () const

  Returns the identifier of the window.
- virtual wxString GetLabel () const

  Generic way of getting a label from any window, for identification purposes.
- virtual wxLayoutDirection GetLayoutDirection () const

  Returns the layout direction for this window, Note that `wxLayout_Default` is returned if layout direction is not supported.
- virtual wxCoord AdjustForLayoutDirection (wxCoord x, wxCoord width, wxCoord widthTotal) const

  Mirror coordinates for RTL layout if this window uses it and if the mirroring is not done automatically like Win32.
- virtual wxString GetName () const

  Returns the window's name.
- wxWindowVariant GetWindowVariant () const

  Returns the value previously passed to SetWindowVariant().
- void SetId (wxWindowID winid)

  Sets the identifier of the window.
- virtual void SetLabel (const wxString &label)

  Sets the window's label.
- virtual void SetLayoutDirection (wxLayoutDirection dir)

  Sets the layout direction for this window.
- virtual void SetName (const wxString &name)

Sets the window's name.

- void SetWindowVariant (wxWindowVariant variant)

  Chooses a different variant of the window display to use.
- wxAcceleratorTable * GetAcceleratorTable ()

  Gets the accelerator table for this window.
- wxAccessible * GetAccessible ()

  Returns the accessible object for this window, if any.
- virtual void SetAcceleratorTable (const wxAcceleratorTable &accel)

  Sets the accelerator table for this window.
- void SetAccessible (wxAccessible *accessible)

  Sets the accessible for this window.

Window deletion functions

- bool Close (bool force=false)

  This function simply generates a wxCloseEvent whose handler usually tries to close the window.
- virtual bool Destroy ()

  Destroys the window safely.
- bool IsBeingDeleted () const

  Returns true if this window is in process of being destroyed.

Drag and drop functions

- virtual wxDropTarget * GetDropTarget () const

  Returns the associated drop target, which may be NULL.
- virtual void SetDropTarget (wxDropTarget *target)

  Associates a drop target with this window.
- virtual void DragAcceptFiles (bool accept)

  Enables or disables eligibility for drop file events (OnDropFiles).

Constraints, sizers and window layout functions

- wxSizer * GetContainingSizer () const

  Returns the sizer of which this window is a member, if any, otherwise NULL.
- wxSizer * GetSizer () const

  Returns the sizer associated with the window by a previous call to SetSizer(), or NULL.
- void SetSizer (wxSizer *sizer, bool deleteOld=true)

  Sets the window to have the given layout sizer.
- void SetSizerAndFit (wxSizer *sizer, bool deleteOld=true)

  This method calls SetSizer() and then wxSizer::SetSizeHints which sets the initial window size to the size needed to accommodate all sizer elements and sets the size hints which, if this window is a top level one, prevent the user from resizing it to be less than this minimal size.
- wxLayoutConstraints * GetConstraints () const

  Returns a pointer to the window's layout constraints, or NULL if there are none.
- void SetConstraints (wxLayoutConstraints *constraints)

  Sets the window to have the given layout constraints.
- virtual bool Layout ()

  Invokes the constraint-based layout algorithm or the sizer-based algorithm for this window.
- void SetAutoLayout (bool autoLayout)

  Determines whether the Layout() function will be called automatically when the window is resized.
- bool GetAutoLayout () const

  Returns the sizer of which this window is a member, if any, otherwise NULL.

Mouse functions

- void CaptureMouse ()

  Directs all mouse input to this window.
- wxCaret * GetCaret () const

  Returns the caret() associated with the window.

- const wxCursor & GetCursor () const

  Return the cursor associated with this window.
- virtual bool HasCapture () const

  Returns true if this window has the current mouse capture.
- void ReleaseMouse ()

  Releases mouse input captured with CaptureMouse().
- void SetCaret (wxCaret *caret)

  Sets the caret() associated with the window.
- virtual bool SetCursor (const wxCursor &cursor)

  Sets the window's cursor.
- virtual void WarpPointer (int x, int y)

  Moves the pointer to the given position on the window.

Miscellaneous functions

- wxHitTest HitTest (wxCoord x, wxCoord y) const

  Return where the given point lies, exactly.
- wxHitTest HitTest (const wxPoint &pt) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- wxBorder GetBorder (long flags) const

  Get the window border style from the given flags: this is different from simply doing flags & wxBORDER_MASK because it uses GetDefaultBorder() to translate wxBORDER_DEFAULT to something reasonable.
- wxBorder GetBorder () const

  Get border for the flags of this window.
- virtual void DoUpdateWindowUI (wxUpdateUIEvent &event)

  Does the window-specific updating after processing the update event.
- virtual WXWidget GetHandle () const

  Returns the platform-specific handle of the physical window.
- virtual bool HasMultiplePages () const

  This method should be overridden to return true if this window has multiple pages.
- virtual void InheritAttributes ()

  This function is (or should be, in case of custom controls) called during window creation to intelligently set up the window visual attributes, that is the font and the foreground and background colours.
- virtual void InitDialog ()

  Sends an `wxEVT_INIT_DIALOG` event, whose handler usually transfers data to the dialog via validators.
- virtual bool IsDoubleBuffered () const

  Returns true if the window contents is double-buffered by the system, i.e. if any drawing done on the window is really done on a temporary backing surface and transferred to the screen all at once later.
- void SetDoubleBuffered (bool on)

  Turn on or off double buffering of the window if the system supports it.
- virtual bool IsRetained () const

  Returns true if the window is retained, false otherwise.
- bool IsThisEnabled () const

  Returns true if this window is intrinsically enabled, false otherwise, i.e. if Enable() Enable(false) had been called.
- virtual bool IsTopLevel () const

  Returns true if the given window is a top-level one.
- virtual void OnInternalIdle ()

  This virtual function is normally only used internally, but sometimes an application may need it to implement functionality that should not be disabled by an application defining an OnIdle handler in a derived class.
- virtual bool SendIdleEvents (wxIdleEvent &event)

  Send idle event to window and all subwindows.
- virtual bool RegisterHotKey (int hotkeyId, int modifiers, int virtualKeyCode)

  Registers a system wide hotkey.
- virtual bool UnregisterHotKey (int hotkeyId)

  Unregisters a system wide hotkey.
- virtual void UpdateWindowUI (long flags=wxUPDATE_UI_NONE)

  This function sends one or more wxUpdateUIEvent to the window.

Static Public Member Functions

Miscellaneous static functions

- static wxVisualAttributes GetClassDefaultAttributes (wxWindowVariant variant=wxWINDOW_VARIANT_NORMAL)

  Returns the default font and colours which are used by the control.
- static wxWindow * FindFocus ()

  Finds the window or control which currently has the keyboard focus.
- static wxWindow * FindWindowById (long id, const wxWindow *parent=0)

  Find the first window with the given id.
- static wxWindow * FindWindowByLabel (const wxString &label, const wxWindow *parent=0)

  Find a window by its label.
- static wxWindow * FindWindowByName (const wxString &name, const wxWindow *parent=0)

  Find a window by its name (as given in a window constructor or Create() function call).
- static wxWindow * GetCapture ()

  Returns the currently captured window.
- static wxWindowID NewControlId (int count=1)

  Create a new ID or range of IDs that are not currently in use.
- static void UnreserveControlId (wxWindowID id, int count=1)

  Unreserve an ID or range of IDs that was reserved by NewControlId().

Protected Member Functions

- virtual void DoCentre (int direction)

  Centres the window.
- virtual wxSize DoGetBestSize () const

  Implementation of GetBestSize() that can be overridden.
- virtual wxSize DoGetBestClientSize () const

  Override this method to return the best size for a custom control.
- virtual int DoGetBestClientHeight (int width) const

  Override this method to implement height-for-width best size calculation.
- virtual int DoGetBestClientWidth (int height) const

  Override this method to implement width-for-height best size calculation.
- virtual void SetInitialBestSize (const wxSize &size)

  Sets the initial window size if none is given (i.e. at least one of the components of the size passed to ctor/Create() is wxDefaultCoord).
- void SendDestroyEvent ()

  Generate wxWindowDestroyEvent for this window.
- virtual bool ProcessEvent (wxEvent &event)

  This function is public in wxEvtHandler but protected in wxWindow because for wxWindows you should always call ProcessEvent() on the pointer returned by GetEventHandler() and not on the wxWindow object itself.
- bool SafelyProcessEvent (wxEvent &event)

  See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.
- virtual void QueueEvent (wxEvent *event)

  See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.
- virtual void AddPendingEvent (const wxEvent &event)

  See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.
- void ProcessPendingEvents ()

  See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.
- bool ProcessThreadEvent (const wxEvent &event)

  See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.

Additional Inherited Members

21.854.2 Constructor & Destructor Documentation

wxWindow::wxWindow ( )

Default constructor.

wxWindow::wxWindow (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxPanelNameStr)

Constructs a window, which can be a child of a frame, dialog or any other non-control window.

Parameters

parent	Pointer to a parent window.
id	Window identifier. If wxID_ANY, will automatically create an identifier.
pos	Window position. wxDefaultPosition indicates that wxWidgets should generate a default position for the window. If using the wxWindow class directly, supply an actual position.
size	Window size. wxDefaultSize indicates that wxWidgets should generate a default size for the window. If no suitable size can be found, the window will be sized to 20x20 pixels so that the window is visible but obviously not correctly sized.
style	Window style. For generic window styles, please see wxWindow.
name	Window name.

virtual wxWindow::~wxWindow ( ) [virtual]

Destructor.

Deletes all sub-windows, then deletes itself. Instead of using the **delete** operator explicitly, you should normally use Destroy() so that wxWidgets can delete a window only when it is safe to do so, in idle time.

See also

Window Deletion Overview, Destroy(), wxCloseEvent

21.854.3 Member Function Documentation

virtual bool wxWindow::AcceptsFocus ( ) const [virtual]

This method may be overridden in the derived classes to return false to indicate that this control doesn't accept input at all (i.e. behaves like e.g. wxStaticText) and so doesn't need focus.

See also

AcceptsFocusFromKeyboard()

Reimplemented in wxPanel.

virtual bool wxWindow::AcceptsFocusFromKeyboard ( ) const [virtual]

This method may be overridden in the derived classes to return false to indicate that while this control can, in principle, have focus if the user clicks it with the mouse, it shouldn't be included in the TAB traversal chain when using the keyboard.

virtual bool wxWindow::AcceptsFocusRecursively ( ) const `[virtual]`

Overridden to indicate whether this window or one of its children accepts focus.

Usually it's the same as AcceptsFocus() but is overridden for container windows.

virtual void wxWindow::AddChild (wxWindow * *child*) `[virtual]`

Adds a child window.

This is called automatically by window creation functions so should not be required by the application programmer. Notice that this function is mostly internal to wxWidgets and shouldn't be called by the user code.

Parameters

child	Child window to add.

virtual void wxWindow::AddPendingEvent (const wxEvent & *event*) `[protected], [virtual]`

See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.

Reimplemented from wxEvtHandler.

virtual wxCoord wxWindow::AdjustForLayoutDirection (wxCoord *x*, wxCoord *width*, wxCoord *widthTotal*) const `[virtual]`

Mirror coordinates for RTL layout if this window uses it and if the mirroring is not done automatically like Win32.

virtual void wxWindow::AlwaysShowScrollbars (bool *hflag* = `true`, bool *vflag* = `true`) `[virtual]`

Call this function to force one or both scrollbars to be always shown, even if the window is big enough to show its entire contents without scrolling.

Since

2.9.0

Parameters

hflag	Whether the horizontal scroll bar should always be visible.
vflag	Whether the vertical scroll bar should always be visible.

Remarks

This function is currently only implemented under Mac/Carbon.

bool wxWindow::BeginRepositioningChildren ( )

Prepare for changing positions of multiple child windows.

This method should be called before changing positions of multiple child windows to reduce flicker and, in MSW case, even avoid display corruption in some cases. It is used internally by wxWidgets and called automatically when the window size changes but it can also be useful to call it from outside of the library if a repositioning involving multiple children is done without changing the window size.

If this method returns true, then EndRepositioningChildren() must be called after setting all children positions. Use ChildrenRepositioningGuard class to ensure that this requirement is satisfied.

Since

2.9.5

void wxWindow::CacheBestSize (const wxSize & *size*) const

Sets the cached best size value.

See also

GetBestSize()

bool wxWindow::CanAcceptFocus ( ) const

Can this window have focus right now?

If this method returns true, it means that calling SetFocus() will put focus either to this window or one of its children, if you need to know whether this window accepts focus itself, use IsFocusable()

bool wxWindow::CanAcceptFocusFromKeyboard ( ) const

Can this window be assigned focus from keyboard right now?

bool wxWindow::CanScroll (int *orient*) const

Returns true if this window can have a scroll bar in this orientation.

Parameters

orient	Orientation to check, either wxHORIZONTAL or wxVERTICAL.

Since

2.9.1

virtual bool wxWindow::CanSetTransparent ( ) [virtual]

Returns true if the system supports transparent windows and calling SetTransparent() may succeed.

If this function returns false, transparent windows are definitely not supported by the current system.

Reimplemented in wxTopLevelWindow.

void wxWindow::CaptureMouse ( )

Directs all mouse input to this window.

Call ReleaseMouse() to release the capture.

Note that wxWidgets maintains the stack of windows having captured the mouse and when the mouse is released the capture returns to the window which had had captured it previously and it is only really released if there were no previous window. In particular, this means that you must release the mouse as many times as you capture it, unless the window receives the wxMouseCaptureLostEvent event.

Any application which captures the mouse in the beginning of some operation must handle wxMouseCaptureLost↩Event and cancel this operation when it receives the event. The event handler must not recapture mouse.

See also

ReleaseMouse(), wxMouseCaptureLostEvent

void wxWindow::Center (int *dir* = **wxBOTH**)

A synonym for Centre().

void wxWindow::CenterOnParent (int *dir* = **wxBOTH**)

A synonym for CentreOnParent().

void wxWindow::Centre (int *direction* = **wxBOTH**)

Centres the window.

Parameters

direction	Specifies the direction for the centring. May be wxHORIZONTAL, wxVERTICAL or wxBOTH. It may also include the wxCENTRE_ON_SCREEN flag if you want to centre the window on the entire screen and not on its parent window.

Remarks

If the window is a top level one (i.e. doesn't have a parent), it will be centred relative to the screen anyhow.

See also

Center()

void wxWindow::CentreOnParent (int *direction* = **wxBOTH**)

Centres the window on its parent.

This is a more readable synonym for Centre().

Parameters

direction	Specifies the direction for the centring. May be wxHORIZONTAL, wxVERTICAL or wxBOTH.

Remarks

This methods provides for a way to centre top level windows over their parents instead of the entire screen. If there is no parent or if the window is not a top level window, then behaviour is the same as Centre().

See also

wxTopLevelWindow::CentreOnScreen

virtual void wxWindow::ClearBackground ( ) [virtual]

Clears the window by filling it with the current background colour.

Does not cause an erase background event to be generated.

Notice that this uses wxClientDC to draw on the window and the results of doing it while also drawing on wxPaintDC for this window are undefined. Hence this method shouldn't be used from EVT_PAINT handlers, just use wxDC::Clear() on the wxPaintDC you already use there instead.

void wxWindow::ClientToScreen (int * *x*, int * *y*) const

Converts to screen coordinates from coordinates relative to this window.

Parameters

x	A pointer to a integer value for the x coordinate. Pass the client coordinate in, and a screen coordinate will be passed out.
y	A pointer to a integer value for the y coordinate. Pass the client coordinate in, and a screen coordinate will be passed out.

wxPerl Note: In wxPerl this method returns a 2-element list instead of modifying its parameters.

wxPoint wxWindow::ClientToScreen (const wxPoint & *pt*) const

Converts to screen coordinates from coordinates relative to this window.

Parameters

pt	The client position for the second form of the function.

virtual wxSize wxWindow::ClientToWindowSize (const wxSize & *size*) const [virtual]

Converts client area size *size* to corresponding window size.

In other words, the returned value is what would GetSize() return if this window had client area of given size. Components with wxDefaultCoord value are left unchanged. Note that the conversion is not always exact, it assumes that non-client area doesn't change and so doesn't take into account things like menu bar (un)wrapping or (dis)appearance of the scrollbars.

Since

2.8.8

See also

WindowToClientSize()

bool wxWindow::Close (bool *force* = false)

This function simply generates a wxCloseEvent whose handler usually tries to close the window.

It doesn't close the window itself, however.

Parameters

force	false if the window's close handler should be able to veto the destruction of this window, true if it cannot.

Returns

true if the event was handled and not vetoed, false otherwise.

Remarks

Close calls the close handler for the window, providing an opportunity for the window to choose whether to destroy the window. Usually it is only used with the top level windows (wxFrame and wxDialog classes) as the others are not supposed to have any special OnClose() logic. The close handler should check whether the window is being deleted forcibly, using wxCloseEvent::CanVeto, in which case it should destroy the window using wxWindow::Destroy. Note that calling Close does not guarantee that the window will be destroyed; but it provides a way to simulate a manual close of a window, which may or may not be implemented by destroying the window. The default implementation of wxDialog::OnCloseWindow does not necessarily delete the dialog, since it will simply simulate an wxID_CANCEL event which is handled by the appropriate button event handler and may do anything at all. To guarantee that the window will be destroyed, call wxWindow::Destroy instead

See also

Window Deletion Overview, Destroy(), wxCloseEvent

wxPoint wxWindow::ConvertDialogToPixels (const wxPoint & *pt*) const

Converts a point or size from dialog units to pixels.

For the x dimension, the dialog units are multiplied by the average character width and then divided by 4. For the y dimension, the dialog units are multiplied by the average character height and then divided by 8.

Remarks

Dialog units are used for maintaining a dialog's proportions even if the font changes. You can also use these functions programmatically. A convenience macro is defined:

```
#define wxDLG_UNIT(parent, pt) parent->ConvertDialogToPixels(pt)
```

See also

ConvertPixelsToDialog()

wxSize wxWindow::ConvertDialogToPixels (const wxSize & *sz*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxPoint wxWindow::ConvertPixelsToDialog (const wxPoint & *pt*) const

Converts a point or size from pixels to dialog units.

For the x dimension, the pixels are multiplied by 4 and then divided by the average character width. For the y dimension, the pixels are multiplied by 8 and then divided by the average character height.

Remarks

Dialog units are used for maintaining a dialog's proportions even if the font changes.

See also

ConvertDialogToPixels()

wxSize wxWindow::ConvertPixelsToDialog (const wxSize & *sz*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxWindow::Create (wxWindow * *parent*, wxWindowID *id*, const wxPoint & *pos* = wxDefaultPosition, const wxSize & *size* = wxDefaultSize, long *style* = 0, const wxString & *name* = wxPanelNameStr)

virtual bool wxWindow::Destroy ( ) [virtual]

Destroys the window safely.

Use this function instead of the delete operator, since different window classes can be destroyed differently. Frames and dialogs are not destroyed immediately when this function is called – they are added to a list of windows to be deleted on idle time, when all the window's events have been processed. This prevents problems with events being sent to non-existent windows.

Returns

true if the window has either been successfully deleted, or it has been added to the list of windows pending real deletion.

bool wxWindow::DestroyChildren ( )

Destroys all children of a window.

Called automatically by the destructor.

bool wxWindow::Disable ( )

Disables the window.

Same as Enable() Enable(false).

Returns

Returns true if the window has been disabled, false if it had been already disabled before the call to this function.

virtual void wxWindow::DoCentre (int *direction*) [protected], [virtual]

Centres the window.

Parameters

direction	Specifies the direction for the centring. May be wxHORIZONTAL, wxVERTICAL or wxBOTH. It may also include the wxCENTRE_ON_SCREEN flag.

Remarks

This function is not meant to be called directly by user code, but via Centre, Center, CentreOnParent, or CenterOnParent. This function can be overridden to fine-tune centring behaviour.

virtual int wxWindow::DoGetBestClientHeight (int *width*) const [protected], [virtual]

Override this method to implement height-for-width best size calculation.

Return the height needed to fully display the control contents if its width is fixed to the given value. Custom classes implementing wrapping should override this method and return the height corresponding to the number of lines needed to lay out the control contents at this width.

Currently this method is not used by wxWidgets yet, however it is planned that it will be used by the new sizer classes implementing height-for-width layout strategy in the future.

Notice that implementing this method or even implementing both it and DoGetBestClientWidth() doesn't replace overriding DoGetBestClientSize(), i.e. you still need to implement the latter as well in order to provide the best size when neither width nor height are constrained.

By default returns wxDefaultCoord meaning that the vertical component of DoGetBestClientSize() return value should be used.

Since

2.9.4

virtual wxSize wxWindow::DoGetBestClientSize ( ) const `[protected], [virtual]`

Override this method to return the best size for a custom control.

A typical implementation of this method should compute the minimal size needed to fully display the control contents taking into account the current font size.

The default implementation simply returns wxDefaultSize and GetBestSize() returns an arbitrary hardcoded size for the window, so you must override it when implementing a custom window class.

Notice that the best size returned by this function is cached internally, so if anything that results in the best size changing (e.g. change to the control contents) happens, you need to call InvalidateBestSize() to notify wxWidgets about it.

See also

Window Sizing Overview

Since

2.9.0

virtual int wxWindow::DoGetBestClientWidth (int *height*) const `[protected], [virtual]`

Override this method to implement width-for-height best size calculation.

This method is exactly the same as DoGetBestClientHeight() except that it determines the width assuming the height is fixed instead of vice versa.

Since

2.9.4

virtual wxSize wxWindow::DoGetBestSize ( ) const `[protected], [virtual]`

Implementation of GetBestSize() that can be overridden.

Notice that it is usually more convenient to override DoGetBestClientSize() rather than this method itself as you need to explicitly account for the window borders size if you do the latter.

The default implementation of this function is designed for use in container windows, such as wxPanel, and works something like this:

1. If the window has a sizer then it is used to calculate the best size.
2. Otherwise if the window has layout constraints then those are used to calculate the best size.
3. Otherwise if the window has children then the best size is set to be large enough to show all the children.
4. Otherwise if there are no children then the window's minimal size will be used as its best size.
5. Otherwise if there is no minimal size set, then the current size is used for the best size.

See also

Window Sizing Overview

Reimplemented in wxRichTextCtrl.

virtual void wxWindow::DoUpdateWindowUI (wxUpdateUIEvent & *event*) [virtual]

Does the window-specific updating after processing the update event.

This function is called by UpdateWindowUI() in order to check return values in the wxUpdateUIEvent and act appropriately. For example, to allow frame and dialog title updating, wxWidgets implements this function as follows:

```
// do the window-specific processing after processing the update event
void wxTopLevelWindowBase::DoUpdateWindowUI(wxUpdateUIEvent& event)
{
    if ( event.GetSetEnabled() )
        Enable(event.GetEnabled());

    if ( event.GetSetText() )
    {
        if ( event.GetText() != GetTitle() )
            SetTitle(event.GetText());
    }
}
```

virtual void wxWindow::DragAcceptFiles (bool *accept*) [virtual]

Enables or disables eligibility for drop file events (OnDropFiles).

Parameters

accept	If true, the window is eligible for drop file events. If false, the window will not accept drop file events.

Remarks

Windows only until version 2.8.9, available on all platforms since 2.8.10. Cannot be used together with SetDropTarget() on non-Windows platforms.

See also

SetDropTarget()

virtual bool wxWindow::Enable (bool *enable* = true) [virtual]

Enable or disable the window for user input.

Note that when a parent window is disabled, all of its children are disabled as well and they are reenabled again when the parent is.

Parameters

enable	If true, enables the window for input. If false, disables the window.

Returns

Returns true if the window has been enabled or disabled, false if nothing was done, i.e. if the window had already been in the specified state.

See also

IsEnabled(), Disable(), wxRadioBox::Enable

void wxWindow::EndRepositioningChildren ( )

Fix child window positions after setting all of them at once.

This method must be called if and only if the previous call to BeginRepositioningChildren() returned true.

Since

2.9.5

static wxWindow* wxWindow::FindFocus ( ) [static]

Finds the window or control which currently has the keyboard focus.

Remarks

Note that this is a static function, so it can be called without needing a wxWindow pointer.

See also

SetFocus(), HasFocus()

wxWindow* wxWindow::FindWindow (long *id*) const

Find a child of this window, by *id*.

May return *this* if it matches itself.

Notice that only real children, not top level windows using this window as parent, are searched by this function.

wxWindow* wxWindow::FindWindow (const wxString & *name*) const

Find a child of this window, by name.

May return *this* if it matches itself.

Notice that only real children, not top level windows using this window as parent, are searched by this function.

static wxWindow* wxWindow::FindWindowById (long *id,* const wxWindow * *parent* = 0) [static]

Find the first window with the given *id*.

If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

See also

FindWindow()

Returns

Window with the given *id* or NULL if not found.

static wxWindow* wxWindow::FindWindowByLabel (const wxString & *label,* const wxWindow * *parent* = 0) [static]

Find a window by its label.

Depending on the type of window, the label may be a window title or panel item label. If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy. The search is recursive in both cases.

See also

FindWindow()

Returns

Window with the given *label* or NULL if not found.

static wxWindow* wxWindow::FindWindowByName (const wxString & *name,* const wxWindow * *parent* = 0) [static]

Find a window by its name (as given in a window constructor or Create() function call).

If *parent* is NULL, the search will start from all top-level frames and dialog boxes; if non-NULL, the search will be limited to the given window hierarchy.

The search is recursive in both cases. If no window with such name is found, FindWindowByLabel() is called.

See also

FindWindow()

Returns

Window with the given *name* or NULL if not found.

virtual void wxWindow::Fit ( ) [virtual]

Sizes the window so that it fits around its subwindows.

This function won't do anything if there are no subwindows and will only really work correctly if sizers are used for the subwindows layout.

Also, if the window has exactly one subwindow it is better (faster and the result is more precise as Fit() adds some margin to account for fuzziness of its calculations) to call:

```
window->SetClientSize(child->GetSize());
```

instead of calling Fit().

See also

Window Sizing Overview

virtual void wxWindow::FitInside ( ) [virtual]

Similar to Fit(), but sizes the interior (virtual) size of a window.

Mainly useful with scrolled windows to reset scrollbars after sizing changes that do not trigger a size event, and/or scrolled windows without an interior sizer. This function similarly won't do anything if there are no subwindows.

void wxWindow::Freeze ( )

Freezes the window or, in other words, prevents any updates from taking place on screen, the window is not redrawn at all.

Thaw() must be called to reenable window redrawing. Calls to these two functions may be nested but to ensure that the window is properly repainted again, you must thaw it exactly as many times as you froze it.

If the window has any children, they are recursively frozen too.

This method is useful for visual appearance optimization (for example, it is a good idea to use it before doing many large text insertions in a row into a wxTextCtrl under wxGTK) but is not implemented on all platforms nor for all controls so it is mostly just a hint to wxWidgets and not a mandatory directive.

See also

wxWindowUpdateLocker, Thaw(), IsFrozen()

wxAcceleratorTable* wxWindow::GetAcceleratorTable ( )

Gets the accelerator table for this window.

See wxAcceleratorTable.

wxAccessible* wxWindow::GetAccessible ( )

Returns the accessible object for this window, if any.

See also wxAccessible.

bool wxWindow::GetAutoLayout ( ) const

Returns the sizer of which this window is a member, if any, otherwise NULL.

wxColour wxWindow::GetBackgroundColour ( ) const

Returns the background colour of the window.

See also

SetBackgroundColour(), SetForegroundColour(), GetForegroundColour()

virtual wxBackgroundStyle wxWindow::GetBackgroundStyle ( ) const `[virtual]`

Returns the background style of the window.

See also

SetBackgroundColour(), GetForegroundColour(), SetBackgroundStyle(), SetTransparent()

int wxWindow::GetBestHeight (int *width*) const

Returns the best height needed by this window if it had the given width.

See also

DoGetBestClientHeight()

Since

2.9.4

wxSize wxWindow::GetBestSize ( ) const

This functions returns the best acceptable minimal size for the window.

For example, for a static control, it will be the minimal size such that the control label is not truncated. For windows containing subwindows (typically wxPanel), the size returned by this function will be the same as the size the window would have had after calling Fit().

Override virtual DoGetBestSize() or, better, because it's usually more convenient, DoGetBestClientSize() when writing your own custom window class to change the value returned by this public non-virtual method.

Notice that the best size respects the minimal and maximal size explicitly set for the window, if any. So even if some window believes that it needs 200 pixels horizontally, calling SetMaxSize() with a width of 100 would ensure that GetBestSize() returns the width of at most 100 pixels.

See also

CacheBestSize(), Window Sizing Overview

virtual wxSize wxWindow::GetBestVirtualSize ( ) const `[virtual]`

Return the largest of ClientSize and BestSize (as determined by a sizer, interior children, or other means)

int wxWindow::GetBestWidth (int *height*) const

Returns the best width needed by this window if it had the given height.

See also

DoGetBestClientWidth()

Since

2.9.4

wxBorder wxWindow::GetBorder (long *flags*) const

Get the window border style from the given flags: this is different from simply doing flags & wxBORDER_MASK because it uses GetDefaultBorder() to translate wxBORDER_DEFAULT to something reasonable.

wxBorder wxWindow::GetBorder ( ) const

Get border for the flags of this window.

static wxWindow* wxWindow::GetCapture ( ) [static]

Returns the currently captured window.

See also

HasCapture(), CaptureMouse(), ReleaseMouse(), wxMouseCaptureLostEvent, wxMouseCaptureChanged↩ Event

wxCaret* wxWindow::GetCaret ( ) const

Returns the caret() associated with the window.

virtual int wxWindow::GetCharHeight ( ) const [virtual]

Returns the character height for this window.

virtual int wxWindow::GetCharWidth ( ) const [virtual]

Returns the average character width for this window.

wxWindowList& wxWindow::GetChildren ( )

Returns a reference to the list of the window's children.

`wxWindowList` is a type-safe wxList-like class whose elements are of type `wxWindow*`.

const wxWindowList& wxWindow::GetChildren ( ) const

Returns a const reference to the list of the window's children.

`wxWindowList` is a type-safe wxList-like class whose elements are of type `wxWindow*`.

static wxVisualAttributes wxWindow::GetClassDefaultAttributes (wxWindowVariant *variant* = wxWINDOW_VARIANT_NORMAL) [static]

Returns the default font and colours which are used by the control.

This is useful if you want to use the same font or colour in your own control as in a standard control – which is a much better idea than hard coding specific colours or fonts which might look completely out of place on the users system, especially if it uses themes.

The *variant* parameter is only relevant under Mac currently and is ignore under other platforms. Under Mac, it will change the size of the returned font. See SetWindowVariant() for more about this.

This static method is "overridden" in many derived classes and so calling, for example, wxButton::GetClassDefault↩ Attributes() will typically return the values appropriate for a button which will be normally different from those returned by, say, wxListCtrl::GetClassDefaultAttributes().

The `wxVisualAttributes` structure has at least the fields `font`, `colFg` and `colBg`. All of them may be invalid if it was not possible to determine the default control appearance or, especially for the background colour, if the field doesn't make sense as is the case for `colBg` for the controls with themed background.

See also

InheritAttributes()

virtual wxPoint wxWindow::GetClientAreaOrigin ( ) const [virtual]

Get the origin of the client area of the window relative to the window top left corner (the client area may be shifted because of the borders, scrollbars, other decorations...)

Reimplemented in wxFrame.

wxRect wxWindow::GetClientRect ( ) const

Get the client rectangle in window (i.e. client) coordinates.

void wxWindow::GetClientSize (int * *width*, int * *height*) const

Returns the size of the window 'client area' in pixels.

The client area is the area which may be drawn on by the programmer, excluding title bar, border, scrollbars, etc. Note that if this window is a top-level one and it is currently minimized, the return size is empty (both width and height are 0).

wxPerl Note: In wxPerl this method takes no parameters and returns a 2-element list (width, height).

See also

GetSize(), GetVirtualSize()

wxSize wxWindow::GetClientSize ( ) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

wxLayoutConstraints* wxWindow::GetConstraints ( ) const

Returns a pointer to the window's layout constraints, or NULL if there are none.

wxSizer* wxWindow::GetContainingSizer ( ) const

Returns the sizer of which this window is a member, if any, otherwise NULL.

virtual double wxWindow::GetContentScaleFactor ( ) const [virtual]

Returns the magnification of the backing store of this window, eg 2.0 for a window on a retina screen.

Since

2.9.5

const wxCursor& wxWindow::GetCursor ( ) const

Return the cursor associated with this window.

See also

SetCursor()

virtual wxVisualAttributes wxWindow::GetDefaultAttributes ( ) const [virtual]

Currently this is the same as calling wxWindow::GetClassDefaultAttributes(wxWindow::GetWindowVariant()).

One advantage of using this function compared to the static version is that the call is automatically dispatched to the correct class (as usual with virtual functions) and you don't have to specify the class name explicitly.

The other one is that in the future this function could return different results, for example it might return a different font for an "Ok" button than for a generic button if the users GUI is configured to show such buttons in bold font. Of course, the down side is that it is impossible to call this function without actually having an object to apply it to whereas the static version can be used without having to create an object first.

virtual wxDropTarget* wxWindow::GetDropTarget ( ) const [virtual]

Returns the associated drop target, which may be NULL.

See also

SetDropTarget(), Drag and Drop Overview

virtual wxSize wxWindow::GetEffectiveMinSize ( ) const [virtual]

Merges the window's best size into the min size and returns the result.

This is the value used by sizers to determine the appropriate amount of space to allocate for the widget.

This is the method called by a wxSizer when it queries the size of a window or control.

See also

GetBestSize(), SetInitialSize(), Window Sizing Overview

wxEvtHandler* wxWindow::GetEventHandler ( ) const

Returns the event handler for this window.

By default, the window is its own event handler.

See also

SetEventHandler(), PushEventHandler(), PopEventHandler(), wxEvtHandler::ProcessEvent, wxEvtHandler

long wxWindow::GetExtraStyle ( ) const

Returns the extra style bits for the window.

wxFont wxWindow::GetFont ( ) const

Returns the font for this window.

See also

SetFont()

wxColour wxWindow::GetForegroundColour ( ) const

Returns the foreground colour of the window.

Remarks

The meaning of foreground colour varies according to the window class; it may be the text colour or other colour, or it may not be used at all.

See also

SetForegroundColour(), SetBackgroundColour(), GetBackgroundColour()

wxWindow* wxWindow::GetGrandParent ( ) const

Returns the grandparent of a window, or NULL if there isn't one.

virtual WXWidget wxWindow::GetHandle ( ) const `[virtual]`

Returns the platform-specific handle of the physical window.

Cast it to an appropriate handle, such as **HWND** for Windows, **Widget** for Motif or **GtkWidget** for GTK.

wxPerl Note: This method will return an integer in wxPerl.

wxString wxWindow::GetHelpText ( ) const

Gets the help text to be used as context-sensitive help for this window.

Note that the text is actually stored by the current wxHelpProvider implementation, and not in the window object itself.

See also

SetHelpText(), GetHelpTextAtPoint(), wxHelpProvider

virtual wxString wxWindow::GetHelpTextAtPoint (const wxPoint & *point*, wxHelpEvent::Origin *origin*) const `[virtual]`

Gets the help text to be used as context-sensitive help for this window.

This method should be overridden if the help message depends on the position inside the window, otherwise GetHelpText() can be used.

Parameters

point	Coordinates of the mouse at the moment of help event emission.
origin	Help event origin, see also wxHelpEvent::GetOrigin.

wxWindowID wxWindow::GetId ( ) const

Returns the identifier of the window.

Remarks

Each window has an integer identifier. If the application has not provided one (or the default wxID_ANY) a unique identifier with a negative value will be generated.

See also

SetId(), Window IDs

virtual wxString wxWindow::GetLabel ( ) const [virtual]

Generic way of getting a label from any window, for identification purposes.

Remarks

The interpretation of this function differs from class to class. For frames and dialogs, the value returned is the title. For buttons or static text controls, it is the button text. This function can be useful for meta-programs (such as testing tools or special-needs access programs) which need to identify windows by name.

Reimplemented in wxButton, wxControl, and wxCommandLinkButton.

virtual wxLayoutDirection wxWindow::GetLayoutDirection ( ) const [virtual]

Returns the layout direction for this window, Note that `wxLayout_Default` is returned if layout direction is not supported.

virtual wxSize wxWindow::GetMaxClientSize ( ) const [virtual]

Returns the maximum size of window's client area.

This is an indication to the sizer layout mechanism that this is the maximum possible size as well as the upper bound on window's size settable using SetClientSize().

See also

GetMaxSize(), Window Sizing Overview

int wxWindow::GetMaxHeight ( ) const

Returns the vertical component of window maximal size.

The returned value is wxDefaultCoord if the maximal width was not set.

See also

GetMaxSize()

virtual wxSize wxWindow::GetMaxSize ( ) const [virtual]

Returns the maximum size of the window.

This is an indication to the sizer layout mechanism that this is the maximum possible size as well as the upper bound on window's size settable using SetSize().

See also

GetMaxClientSize(), Window Sizing Overview

int wxWindow::GetMaxWidth ( ) const

Returns the horizontal component of window maximal size.

The returned value is wxDefaultCoord if the maximal width was not set.

See also

GetMaxSize()

virtual wxSize wxWindow::GetMinClientSize ( ) const `[virtual]`

Returns the minimum size of window's client area, an indication to the sizer layout mechanism that this is the minimum required size of its client area.

It normally just returns the value set by SetMinClientSize(), but it can be overridden to do the calculation on demand.

See also

GetMinSize(), Window Sizing Overview

int wxWindow::GetMinHeight ( ) const

Returns the vertical component of window minimal size.

The returned value is wxDefaultCoord if the minimal height was not set.

See also

GetMinSize()

virtual wxSize wxWindow::GetMinSize ( ) const `[virtual]`

Returns the minimum size of the window, an indication to the sizer layout mechanism that this is the minimum required size.

This method normally just returns the value set by SetMinSize(), but it can be overridden to do the calculation on demand.

See also

GetMinClientSize(), Window Sizing Overview

int wxWindow::GetMinWidth ( ) const

Returns the horizontal component of window minimal size.

The returned value is wxDefaultCoord if the minimal width was not set.

See also

GetMinSize()

virtual wxString wxWindow::GetName ( ) const `[virtual]`

Returns the window's name.

Remarks

This name is not guaranteed to be unique; it is up to the programmer to supply an appropriate name in the window constructor or via SetName().

See also

SetName()

wxWindow* wxWindow::GetNextSibling ( ) const

Returns the next window after this one among the parent's children or NULL if this window is the last child.

Since

2.8.8

See also

GetPrevSibling()

wxWindow* wxWindow::GetParent ( ) const

Returns the parent of the window, or NULL if there is no parent.

int wxWindow::GetPopupMenuSelectionFromUser (wxMenu & *menu,* const wxPoint & *pos* = wxDefaultPosition)

This function shows a popup menu at the given position in this window and returns the selected id.

It can be more convenient than the general purpose PopupMenu() function for simple menus proposing a choice in a list of strings to the user.

Notice that to avoid unexpected conflicts between the (usually consecutive range of) ids used by the menu passed to this function and the existing EVT_UPDATE_UI() handlers, this function temporarily disables UI updates for the window, so you need to manually disable (or toggle or ...) any items which should be disabled in the menu before showing it.

The parameter *menu* is the menu to show. The parameter *pos* (or the parameters *x* and *y*) is the position at which to show the menu in client coordinates. It is recommended to not explicitly specify coordinates when calling this method in response to mouse click, because some of the ports (namely, wxGTK) can do a better job of positioning the menu in that case.

Returns

The selected menu item id or `wxID_NONE` if none selected or an error occurred.

Since

2.9.0

int wxWindow::GetPopupMenuSelectionFromUser (wxMenu & *menu,* int *x,* int *y*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxWindow::GetPosition (int * *x*, int * *y*) const

This gets the position of the window in pixels, relative to the parent window for the child windows or relative to the display origin for the top level windows.

Parameters

x	Receives the x position of the window if non-NULL.
y	Receives the y position of the window if non-NULL.

wxPerl Note: In wxPerl this method is implemented as GetPositionXY() returning a 2-element list (x, y).

See also

GetScreenPosition()

wxPoint wxWindow::GetPosition ( ) const

This gets the position of the window in pixels, relative to the parent window for the child windows or relative to the display origin for the top level windows.

See also

GetScreenPosition()

wxWindow* wxWindow::GetPrevSibling ( ) const

Returns the previous window before this one among the parent's children or NULL if this window is the first child.

Since

2.8.8

See also

GetNextSibling()

wxRect wxWindow::GetRect ( ) const

Returns the position and size of the window as a wxRect object.

See also

GetScreenRect()

void wxWindow::GetScreenPosition (int* *x*, int* *y*) const

Returns the window position in screen coordinates, whether the window is a child window or a top level one.

Parameters

x	Receives the x position of the window on the screen if non-NULL.
y	Receives the y position of the window on the screen if non-NULL.

See also

GetPosition()

wxPoint wxWindow::GetScreenPosition ( ) const

Returns the window position in screen coordinates, whether the window is a child window or a top level one.

See also

GetPosition()

wxRect wxWindow::GetScreenRect ( ) const

Returns the position and size of the window on the screen as a wxRect object.

See also

GetRect()

virtual int wxWindow::GetScrollPos (int *orientation*) const `[virtual]`

Returns the built-in scrollbar position.

See also

SetScrollbar()

virtual int wxWindow::GetScrollRange (int *orientation*) const `[virtual]`

Returns the built-in scrollbar range.

See also

SetScrollbar()

virtual int wxWindow::GetScrollThumb (int *orientation*) const `[virtual]`

Returns the built-in scrollbar thumb size.

See also

SetScrollbar()

void wxWindow::GetSize (int * *width,* int * *height*) const

Returns the size of the entire window in pixels, including title bar, border, scrollbars, etc.

Note that if this window is a top-level one and it is currently minimized, the returned size is the restored window size, not the size of the window icon.

Parameters

width	Receives the window width.

height	Receives the window height.

wxPerl Note: In wxPerl this method is implemented as GetSizeWH() returning a 2-element list (width, height).

See also

GetClientSize(), GetVirtualSize(), Window Sizing Overview

wxSize wxWindow::GetSize ( ) const

See the GetSize(int*,int*) overload for more info.

wxSizer* wxWindow::GetSizer ( ) const

Returns the sizer associated with the window by a previous call to SetSizer(), or NULL.

void wxWindow::GetTextExtent (const wxString & *string*, int * *w*, int * *h*, int * *descent* = NULL, int * *externalLeading* = NULL, const wxFont * *font* = NULL) const

Gets the dimensions of the string as it would be drawn on the window with the currently selected font.

The text extent is returned in the *w* and *h* pointers.

Parameters

string	String whose extent is to be measured.
w	Return value for width.
h	Return value for height.
descent	Return value for descent (optional).
externalLeading	Return value for external leading (optional).
font	Font to use instead of the current window font (optional).

wxPerl Note: In wxPerl this method takes only the *string* and optionally *font* parameters, and returns a 4-element list (x, y, descent, externalLeading).

wxSize wxWindow::GetTextExtent (const wxString & *string*) const

Gets the dimensions of the string as it would be drawn on the window with the currently selected font.

virtual bool wxWindow::GetThemeEnabled ( ) const [virtual]

Clears the window by filling it with the current background colour.

Does not cause an erase background event to be generated.

Notice that this uses wxClientDC to draw on the window and the results of doing it while also drawing on wxPaintDC for this window are undefined. Hence this method shouldn't be used from EVT_PAINT handlers, just use wxDC::Clear() on the wxPaintDC you already use there instead.

wxToolTip* wxWindow::GetToolTip ( ) const

Get the associated tooltip or NULL if none.

wxString wxWindow::GetToolTipText ( ) const

Get the text of the associated tooltip or empty string if none.

wxRect wxWindow::GetUpdateClientRect ( ) const

Get the update rectangle bounding box in client coords.

const wxRegion& wxWindow::GetUpdateRegion ( ) const

Returns the region specifying which parts of the window have been damaged.

Should only be called within an wxPaintEvent handler.

See also

wxRegion, wxRegionIterator

virtual wxValidator* wxWindow::GetValidator ( ) [virtual]

Validator functions.

Returns a pointer to the current validator for the window, or NULL if there is none.

wxSize wxWindow::GetVirtualSize ( ) const

This gets the virtual size of the window in pixels.

By default it returns the client size of the window, but after a call to SetVirtualSize() it will return the size set with that method.

See also

Window Sizing Overview

void wxWindow::GetVirtualSize (int * *width*, int * *height*) const

Like the other GetVirtualSize() overload but uses pointers instead.

Parameters

width	Receives the window virtual width.
height	Receives the window virtual height.

virtual wxSize wxWindow::GetWindowBorderSize ( ) const [virtual]

Returns the size of the left/right and top/bottom borders of this window in x and y components of the result respectively.

long wxWindow::GetWindowStyle ( ) const

See GetWindowStyleFlag() for more info.

virtual long wxWindow::GetWindowStyleFlag ( ) const [virtual]

Gets the window style that was passed to the constructor or Create() method.

GetWindowStyle() is another name for the same function.

Reimplemented in wxAuiToolBar.

wxWindowVariant wxWindow::GetWindowVariant ( ) const

Returns the value previously passed to SetWindowVariant().

bool wxWindow::HandleAsNavigationKey (const wxKeyEvent & *event*)

This function will generate the appropriate call to Navigate() if the key event is one normally used for keyboard navigation and return true in this case.

Returns

Returns true if the key pressed was for navigation and was handled, false otherwise.

See also

Navigate()

bool wxWindow::HandleWindowEvent (wxEvent & *event*) const

Shorthand for:

```
GetEventHandler()->SafelyProcessEvent(event);
```

See also

ProcessWindowEvent()

virtual bool wxWindow::HasCapture ( ) const [virtual]

Returns true if this window has the current mouse capture.

See also

CaptureMouse(), ReleaseMouse(), wxMouseCaptureLostEvent, wxMouseCaptureChangedEvent

bool wxWindow::HasExtraStyle (int *exFlag*) const

Returns true if the window has the given *exFlag* bit set in its extra styles.

See also

SetExtraStyle()

bool wxWindow::HasFlag (int *flag*) const

Returns true if the window has the given *flag* bit set.

virtual bool wxWindow::HasFocus ( ) const `[virtual]`

Returns true if the window (or in case of composite controls, its main child window) has focus.

Since

2.9.0

See also

FindFocus()

virtual bool wxWindow::HasMultiplePages ( ) const `[virtual]`

This method should be overridden to return true if this window has multiple pages.

All standard class with multiple pages such as wxNotebook, wxListbook and wxTreebook already override it to return true and user-defined classes with similar behaviour should also do so, to allow the library to handle such windows appropriately.

bool wxWindow::HasScrollbar (int *orient*) const

Returns true if this window currently has a scroll bar for this orientation.

This method may return false even when CanScroll() for the same orientation returns true, but if CanScroll() returns false, i.e. scrolling in this direction is not enabled at all, HasScrollbar() always returns false as well.

Parameters

orient	Orientation to check, either wxHORIZONTAL or wxVERTICAL.

virtual bool wxWindow::HasTransparentBackground ( ) `[virtual]`

Returns true if this window background is transparent (as, for example, for wxStaticText) and should show the parent window background.

This method is mostly used internally by the library itself and you normally shouldn't have to call it. You may, however, have to override it in your wxWindow-derived class to ensure that background is painted correctly.

bool wxWindow::Hide ( )

Equivalent to calling wxWindow::Show(false).

virtual bool wxWindow::HideWithEffect (wxShowEffect *effect*, unsigned int *timeout* = 0) `[virtual]`

This function hides a window, like Hide(), but using a special visual effect if possible.

The parameters of this function are the same as for ShowWithEffect(), please see their description there.

Since

2.9.0

wxHitTest wxWindow::HitTest (wxCoord *x*, wxCoord *y*) const

Return where the given point lies, exactly.

This method is used to test whether the point lies inside the client window area or on one of its scrollbars.

The point coordinates are specified in client window coordinates.

wxHitTest wxWindow::HitTest (const wxPoint & *pt*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

virtual bool wxWindow::InformFirstDirection (int *direction*, int *size*, int *availableOtherDir*) [virtual]

wxSizer and friends use this to give a chance to a component to recalc its min size once one of the final size components is known.

Override this function when that is useful (such as for wxStaticText which can stretch over several lines). Parameter availableOtherDir tells the item how much more space there is available in the opposite direction (-1 if unknown).

virtual void wxWindow::InheritAttributes ( ) [virtual]

This function is (or should be, in case of custom controls) called during window creation to intelligently set up the window visual attributes, that is the font and the foreground and background colours.

By "intelligently" the following is meant: by default, all windows use their own GetClassDefaultAttributes() default attributes. However if some of the parents attributes are explicitly (that is, using SetFont() and not wxWindow::SetOwnFont) changed and if the corresponding attribute hadn't been explicitly set for this window itself, then this window takes the same value as used by the parent. In addition, if the window overrides ShouldInheritColours() to return false, the colours will not be changed no matter what and only the font might.

This rather complicated logic is necessary in order to accommodate the different usage scenarios. The most common one is when all default attributes are used and in this case, nothing should be inherited as in modern GUIs different controls use different fonts (and colours) than their siblings so they can't inherit the same value from the parent. However it was also deemed desirable to allow to simply change the attributes of all children at once by just changing the font or colour of their common parent, hence in this case we do inherit the parents attributes.

bool wxWindow::InheritsBackgroundColour ( ) const

Return true if this window inherits the background colour from its parent.

See also

SetOwnBackgroundColour(), InheritAttributes()

virtual void wxWindow::InitDialog ( ) [virtual]

Sends an `wxEVT_INIT_DIALOG` event, whose handler usually transfers data to the dialog via validators.

Reimplemented in wxPanel.

void wxWindow::InvalidateBestSize ( )

Resets the cached best size value so it will be recalculated the next time it is needed.

See also

CacheBestSize()

bool wxWindow::IsBeingDeleted ( ) const

Returns true if this window is in process of being destroyed.

Top level windows are not deleted immediately but are rather scheduled for later destruction to give them time to process any pending messages; see Destroy() description.

This function returns true if this window, or one of its parent windows, is scheduled for destruction and can be useful to avoid manipulating it as it's usually useless to do something with a window which is on the point of disappearing anyhow.

bool wxWindow::IsDescendant (wxWindowBase * *win*) const

Check if the specified window is a descendant of this one.

Returns true if the window is a descendant (i.e. a child or grand-child or grand-grand-child or ...) of this one.

Notice that a window can never be a descendant of another one if they are in different top level windows, i.e. a child of a wxDialog is not considered to be a descendant of dialogs parent wxFrame.

Parameters

win	Any window, possible NULL (false is always returned then).

Since

2.9.4

virtual bool wxWindow::IsDoubleBuffered ( ) const `[virtual]`

Returns true if the window contents is double-buffered by the system, i.e. if any drawing done on the window is really done on a temporary backing surface and transferred to the screen all at once later.

See also

wxBufferedDC

bool wxWindow::IsEnabled ( ) const

Returns true if the window is enabled, i.e. if it accepts user input, false otherwise.

Notice that this method can return false even if this window itself hadn't been explicitly disabled when one of its parent windows is disabled. To get the intrinsic status of this window, use IsThisEnabled()

See also

Enable()

bool wxWindow::IsExposed (int *x*, int *y*) const

Returns true if the given point or rectangle area has been exposed since the last repaint.

Call this in an paint event handler to optimize redrawing by only redrawing those areas, which have been exposed.

bool wxWindow::IsExposed (wxPoint & *pt*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxWindow::IsExposed (int *x,* int *y,* int *w,* int *h*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxWindow::IsExposed (wxRect & *rect*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

bool wxWindow::IsFocusable ( ) const

Can this window itself have focus?

bool wxWindow::IsFrozen ( ) const

Returns true if the window is currently frozen by a call to Freeze().

See also

Freeze(), Thaw()

virtual bool wxWindow::IsRetained ( ) const `[virtual]`

Returns true if the window is retained, false otherwise.

Remarks

Retained windows are only available on X platforms.

virtual bool wxWindow::IsScrollbarAlwaysShown (int *orient*) const `[virtual]`

Return whether a scrollbar is always shown.

Parameters

orient	Orientation to check, either wxHORIZONTAL or wxVERTICAL.

See also

AlwaysShowScrollbars()

virtual bool wxWindow::IsShown ( ) const `[virtual]`

Returns true if the window is shown, false if it has been hidden.

See also

IsShownOnScreen()

virtual bool wxWindow::IsShownOnScreen ( ) const `[virtual]`

Returns true if the window is physically visible on the screen, i.e. it is shown and all its parents up to the toplevel window are shown as well.

See also

IsShown()

bool wxWindow::IsThisEnabled ( ) const

Returns true if this window is intrinsically enabled, false otherwise, i.e. if Enable() Enable(false) had been called.

This method is mostly used for wxWidgets itself, user code should normally use IsEnabled() instead.

virtual bool wxWindow::IsTopLevel ( ) const `[virtual]`

Returns true if the given window is a top-level one.

Currently all frames and dialogs are considered to be top-level windows (even if they have a parent window).

virtual bool wxWindow::IsTransparentBackgroundSupported (wxString * *reason* = NULL) const `[virtual]`

Checks whether using transparent background might work.

If this function returns false, calling SetBackgroundStyle() with wxBG_STYLE_TRANSPARENT is not going to work. If it returns true, setting transparent style should normally succeed.

Notice that this function would typically be called on the parent of a window you want to set transparent background style for as the window for which this method is called must be fully created.

Parameters

reason	If not NULL, a reason message is provided if transparency is not supported.

Returns

true if background transparency is supported.

Since

2.9.4

virtual bool wxWindow::Layout ( ) `[virtual]`

Invokes the constraint-based layout algorithm or the sizer-based algorithm for this window.

This function does not get called automatically when the window is resized because lots of windows deriving from wxWindow does not need this functionality. If you want to have Layout() called automatically, you should derive from wxPanel (see wxPanel::Layout).

See also

Window Sizing Overview

Reimplemented in wxTopLevelWindow, and wxPanel.

bool wxWindow::LineDown ( )

Same as ScrollLines (1).

bool wxWindow::LineUp ( )

Same as ScrollLines (-1).

virtual void wxWindow::Lower ( ) [virtual]

Lowers the window to the bottom of the window hierarchy (Z-order).

Remarks

This function only works for wxTopLevelWindow-derived classes.

See also

Raise()

void wxWindow::Move (int *x*, int *y*, int *flags* = wxSIZE_USE_EXISTING)

Moves the window to the given position.

Parameters

x	Required x position.
y	Required y position.
flags	See SetSize() for more info about this parameter.

Remarks

Implementations of SetSize can also implicitly implement the Move() function, which is defined in the base wxWindow class as the call:

```
SetSize(x, y, wxDefaultCoord, wxDefaultCoord,
      wxSIZE_USE_EXISTING);
```

See also

SetSize()

void wxWindow::Move (const wxPoint & *pt*, int *flags* = wxSIZE_USE_EXISTING)

Moves the window to the given position.

Parameters

pt	wxPoint object representing the position.
flags	See SetSize() for more info about this parameter.

Remarks

Implementations of SetSize() can also implicitly implement the Move() function, which is defined in the base wxWindow class as the call:

```
SetSize(x, y, wxDefaultCoord, wxDefaultCoord,
      wxSIZE_USE_EXISTING);
```

See also

SetSize()

void wxWindow::MoveAfterInTabOrder (wxWindow * *win*)

Moves this window in the tab navigation order after the specified *win*.

This means that when the user presses `TAB` key on that other window, the focus switches to this window.

Default tab order is the same as creation order, this function and MoveBeforeInTabOrder() allow to change it after creating all the windows.

Parameters

win	A sibling of this window which should precede it in tab order, must not be NULL

void wxWindow::MoveBeforeInTabOrder (wxWindow * *win*)

Same as MoveAfterInTabOrder() except that it inserts this window just before *win* instead of putting it right after it.

bool wxWindow::Navigate (int *flags* = wxNavigationKeyEvent::IsForward)

Performs a keyboard navigation action starting from this window.

This method is equivalent to calling NavigateIn() method on the parent window.

Parameters

flags	A combination of wxNavigationKeyEvent::IsForward and wxNavigationKeyEvent::WinChange.

Returns

Returns true if the focus was moved to another window or false if nothing changed.

Remarks

You may wish to call this from a text control custom keypress handler to do the default navigation behaviour for the tab key, since the standard default behaviour for a multiline text control with the wxTE_PROCESS_TAB style is to insert a tab and not navigate to the next control. See also wxNavigationKeyEvent and HandleAsNavigationKey.

bool wxWindow::NavigateIn (int *flags* = wxNavigationKeyEvent::IsForward)

Performs a keyboard navigation action inside this window.

See Navigate() for more information.

static wxWindowID wxWindow::NewControlId (int *count* = `1`) `[static]`

Create a new ID or range of IDs that are not currently in use.

The IDs will be reserved until assigned to a wxWindow ID or unreserved with UnreserveControlId().

See Window IDs for more information.

Parameters

count	The number of sequential IDs to reserve.

Returns

Returns the ID or the first ID of the range (i.e. the most negative), or wxID_NONE if the specified number of identifiers couldn't be allocated.

See also

UnreserveControlId(), wxIdManager, Window IDs

virtual void wxWindow::OnInternalIdle ( ) [virtual]

This virtual function is normally only used internally, but sometimes an application may need it to implement functionality that should not be disabled by an application defining an OnIdle handler in a derived class.

This function may be used to do delayed painting, for example, and most implementations call UpdateWindowUI() in order to send update events to the window in idle time.

bool wxWindow::PageDown ( )

Same as ScrollPages (1).

bool wxWindow::PageUp ( )

Same as ScrollPages (-1).

wxEvtHandler* wxWindow::PopEventHandler (bool *deleteHandler* = false)

Removes and returns the top-most event handler on the event handler stack.

E.g. in the case of: when calling `W->PopEventHandler()`, the event handler `A` will be removed and `B` will be the first handler of the stack.

Note that it's an error to call this function when no event handlers were pushed on this window (i.e. when the window itself is its only event handler).

Parameters

deleteHandler	If this is true, the handler will be deleted after it is removed (and the returned value will be NULL).

See also

How Events are Processed

bool wxWindow::PopupMenu (wxMenu * *menu,* const wxPoint & *pos* = wxDefaultPosition)

Pops up the given menu at the specified coordinates, relative to this window, and returns control when the user has dismissed the menu.

If a menu item is selected, the corresponding menu event is generated and will be processed as usual. If coordinates are not specified, the current mouse cursor position is used.

menu is the menu to pop up.

The position where the menu will appear can be specified either as a wxPoint *pos* or by two integers (*x* and *y*).

Remarks

Just before the menu is popped up, wxMenu::UpdateUI is called to ensure that the menu items are in the correct state. The menu does not get deleted by the window. It is recommended to not explicitly specify coordinates when calling PopupMenu in response to mouse click, because some of the ports (namely, wxG↩ TK) can do a better job of positioning the menu in that case.

See also

wxMenu

bool wxWindow::PopupMenu (wxMenu * *menu,* int *x,* int *y*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxWindow::PostSizeEvent ( )

Posts a size event to the window.

This is the same as SendSizeEvent() with `wxSEND_EVENT_POST` argument.

void wxWindow::PostSizeEventToParent ( )

Posts a size event to the parent of this window.

This is the same as SendSizeEventToParent() with `wxSEND_EVENT_POST` argument.

virtual bool wxWindow::ProcessEvent (wxEvent & *event*) [protected], [virtual]

This function is public in wxEvtHandler but protected in wxWindow because for wxWindows you should always call ProcessEvent() on the pointer returned by GetEventHandler() and not on the wxWindow object itself.

For convenience, a ProcessWindowEvent() method is provided as a synonym for

```
GetEventHandler()->ProcessEvent()
```

Note that it's still possible to call these functions directly on the wxWindow object (e.g. casting it to wxEvtHandler) but doing that will create subtle bugs when windows with event handlers pushed on them are involved.

This holds also for all other wxEvtHandler functions.

Reimplemented from wxEvtHandler.

void wxWindow::ProcessPendingEvents ( ) [protected]

See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.

bool wxWindow::ProcessThreadEvent (const wxEvent & *event*) [protected]

See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.

bool wxWindow::ProcessWindowEvent (wxEvent & *event*)

Convenient wrapper for ProcessEvent().

This is the same as writing

```
GetEventHandler()->ProcessEvent(event);
```

but more convenient. Notice that ProcessEvent() itself can't be called for wxWindow objects as it ignores the event handlers associated with the window; use this function instead.

bool wxWindow::ProcessWindowEventLocally (wxEvent & *event*)

Wrapper for wxEvtHandler::ProcessEventLocally().

This method is similar to ProcessWindowEvent() but can be used to search for the event handler only in this window and any event handlers pushed on top of it. Unlike ProcessWindowEvent() it won't propagate the event upwards. But it will use the validator and event handlers associated with this window, if any.

Since

2.9.1

void wxWindow::PushEventHandler (wxEvtHandler * *handler*)

Pushes this event handler onto the event stack for the window.

An event handler is an object that is capable of processing the events sent to a window. By default, the window is its own event handler, but an application may wish to substitute another, for example to allow central implementation of event-handling for a variety of different window classes.

wxWindow::PushEventHandler allows an application to set up a *stack* of event handlers, where an event not handled by one event handler is handed to the next one in the chain.

E.g. if you have two event handlers `A` and `B` and a wxWindow instance `W` and you call:

```
W->PushEventHandler(A);
W->PushEventHandler(B);
```

you will end up with the following situation:

Note that you can use wxWindow::PopEventHandler to remove the event handler.

Parameters

handler	Specifies the handler to be pushed. It must not be part of a wxEvtHandler chain; an assert will fail if it's not unlinked (see wxEvtHandler::IsUnlinked).

See also

How Events are Processed

virtual void wxWindow::QueueEvent (wxEvent * *event*) [protected], [virtual]

See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.

Reimplemented from wxEvtHandler.

virtual void wxWindow::Raise ( ) [virtual]

Raises the window to the top of the window hierarchy (Z-order).

Notice that this function only requests the window manager to raise this window to the top of Z-order. Depending on its configuration, the window manager may raise the window, not do it at all or indicate that a window requested to be raised in some other way, e.g. by flashing its icon if it is minimized.

Remarks

This function only works for wxTopLevelWindow-derived classes.

See also

Lower()

virtual void wxWindow::Refresh (bool *eraseBackground* = true, const wxRect * *rect* = NULL) [virtual]

Causes this window, and all of its children recursively (except under wxGTK1 where this is not implemented), to be repainted.

Note that repainting doesn't happen immediately but only during the next event loop iteration, if you need to update the window immediately you should use Update() instead.

Parameters

eraseBackground	If true, the background will be erased.
rect	If non-NULL, only the given rectangle will be treated as damaged.

See also

RefreshRect()

Reimplemented in wxMenuBar.

void wxWindow::RefreshRect (const wxRect & *rect*, bool *eraseBackground* = true)

Redraws the contents of the given rectangle: only the area inside it will be repainted.

This is the same as Refresh() but has a nicer syntax as it can be called with a temporary wxRect object as argument like this `RefreshRect(wxRect(x, y, w, h))`.

virtual bool wxWindow::RegisterHotKey (int *hotkeyId*, int *modifiers*, int *virtualKeyCode*) [virtual]

Registers a system wide hotkey.

Every time the user presses the hotkey registered here, this window will receive a hotkey event.

It will receive the event even if the application is in the background and does not have the input focus because the user is working with some other application.

Parameters

hotkeyId	Numeric identifier of the hotkey. For applications this must be between 0 and 0xBFFF. If this function is called from a shared DLL, it must be a system wide unique identifier between 0xC000 and 0xFFFF. This is a MSW specific detail.
modifiers	A bitwise combination of wxMOD_SHIFT, wxMOD_CONTROL, wxMOD_ALT or wxMOD_WIN specifying the modifier keys that have to be pressed along with the key.
virtualKeyCode	The virtual key code of the hotkey.

Returns

true if the hotkey was registered successfully. false if some other application already registered a hotkey with this modifier/virtualKeyCode combination.

Remarks

Use EVT_HOTKEY(hotkeyId, fnc) in the event table to capture the event. This function is currently only implemented under Windows. It is used in the Windows CE port for detecting hardware button presses.

See also

UnregisterHotKey()

void wxWindow::ReleaseMouse ( )

Releases mouse input captured with CaptureMouse().

See also

CaptureMouse(), HasCapture(), ReleaseMouse(), wxMouseCaptureLostEvent, wxMouseCaptureChanged↩ Event

virtual void wxWindow::RemoveChild (wxWindow * *child*) [virtual]

Removes a child window.

This is called automatically by window deletion functions so should not be required by the application programmer. Notice that this function is mostly internal to wxWidgets and shouldn't be called by the user code.

Parameters

child	Child window to remove.

bool wxWindow::RemoveEventHandler (wxEvtHandler * *handler*)

Find the given *handler* in the windows event handler stack and removes (but does not delete) it from the stack.

See wxEvtHandler::Unlink() for more info.

Parameters

handler	The event handler to remove, must be non-NULL and must be present in this windows event handlers stack.

Returns

Returns true if it was found and false otherwise (this also results in an assert failure so this function should only be called when the handler is supposed to be there).

See also

PushEventHandler(), PopEventHandler()

virtual bool wxWindow::Reparent (wxWindow * *newParent*) `[virtual]`

Reparents the window, i.e. the window will be removed from its current parent window (e.g.

a non-standard toolbar in a wxFrame) and then re-inserted into another.

Notice that currently you need to explicitly call wxNotebook::RemovePage() before reparenting a notebook page.

Parameters

newParent	New parent.

bool wxWindow::SafelyProcessEvent (wxEvent & *event*) [protected]

See ProcessEvent() for more info about why you shouldn't use this function and the reason for making this function protected in wxWindow.

void wxWindow::ScreenToClient (int * *x*, int * *y*) const

Converts from screen to client window coordinates.

Parameters

x	Stores the screen x coordinate and receives the client x coordinate.
y	Stores the screen x coordinate and receives the client x coordinate.

wxPoint wxWindow::ScreenToClient (const wxPoint & *pt*) const

Converts from screen to client window coordinates.

Parameters

pt	The screen position.

virtual bool wxWindow::ScrollLines (int *lines*) [virtual]

Scrolls the window by the given number of lines down (if *lines* is positive) or up.

Returns

Returns true if the window was scrolled, false if it was already on top/bottom and nothing was done.

Remarks

This function is currently only implemented under MSW and wxTextCtrl under wxGTK (it also works for wxScrolled classes under all platforms).

See also

ScrollPages()

Reimplemented in wxRibbonGallery, and wxRibbonPage.

virtual bool wxWindow::ScrollPages (int *pages*) [virtual]

Scrolls the window by the given number of pages down (if *pages* is positive) or up.

Returns

Returns true if the window was scrolled, false if it was already on top/bottom and nothing was done.

Remarks

This function is currently only implemented under MSW and wxGTK.

See also

ScrollLines()

virtual void wxWindow::ScrollWindow (int *dx*, int *dy*, const wxRect * *rect* = NULL) [virtual]

Physically scrolls the pixels in the window and move child windows accordingly.

Parameters

dx	Amount to scroll horizontally.
dy	Amount to scroll vertically.
rect	Rectangle to scroll, if it is NULL, the whole window is scrolled (this is always the case under wxGTK which doesn't support this parameter)

Remarks

Note that you can often use wxScrolled instead of using this function directly.

void wxWindow::SendDestroyEvent ( ) [protected]

Generate wxWindowDestroyEvent for this window.

This is called by the window itself when it is being destroyed and usually there is no need to call it but see wxWindowDestroyEvent for explanations of when you might want to do it.

virtual bool wxWindow::SendIdleEvents (wxIdleEvent & *event*) [virtual]

Send idle event to window and all subwindows.

Returns true if more idle time is requested.

virtual void wxWindow::SendSizeEvent (int *flags* = 0) [virtual]

This function sends a dummy size event to the window allowing it to re-layout its children positions.

It is sometimes useful to call this function after adding or deleting a children after the frame creation or if a child size changes. Note that if the frame is using either sizers or constraints for the children layout, it is enough to call wxWindow::Layout() directly and this function should not be used in this case.

If *flags* includes wxSEND_EVENT_POST value, this function posts the event, i.e. schedules it for later processing, instead of dispatching it directly. You can also use PostSizeEvent() as a more readable equivalent of calling this function with this flag.

Parameters

flags	May include wxSEND_EVENT_POST. Default value is 0.

void wxWindow::SendSizeEventToParent (int *flags* = 0)

Safe wrapper for GetParent()->SendSizeEvent().

This function simply checks that the window has a valid parent which is not in process of being deleted and calls SendSizeEvent() on it. It is used internally by windows such as toolbars changes to whose state should result in parent re-layout (e.g. when a toolbar is added to the top of the window, all the other windows must be shifted down).

See also

PostSizeEventToParent()

Parameters

flags	See description of this parameter in SendSizeEvent() documentation.

virtual void wxWindow::SetAcceleratorTable (const wxAcceleratorTable & *accel*) [virtual]

Sets the accelerator table for this window.

See wxAcceleratorTable.

void wxWindow::SetAccessible (wxAccessible * *accessible*)

Sets the accessible for this window.

Any existing accessible for this window will be deleted first, if not identical to *accessible*. See also wxAccessible.

void wxWindow::SetAutoLayout (bool *autoLayout*)

Determines whether the Layout() function will be called automatically when the window is resized.

This method is called implicitly by SetSizer() but if you use SetConstraints() you should call it manually or otherwise the window layout won't be correctly updated when its size changes.

Parameters

autoLayout	Set this to true if you wish the Layout() function to be called automatically when the window is resized.

See also

SetSizer(), SetConstraints()

virtual bool wxWindow::SetBackgroundColour (const wxColour & *colour*) [virtual]

Sets the background colour of the window.

Notice that as with SetForegroundColour(), setting the background colour of a native control may not affect the entire control and could be not supported at all depending on the control and platform.

Please see InheritAttributes() for explanation of the difference between this method and SetOwnBackground↩Colour().

Parameters

colour	The colour to be used as the background colour; pass wxNullColour to reset to the default colour. Note that you may want to use wxSystemSettings::GetColour() to retrieve a suitable colour to use rather than setting an hard-coded one.

Remarks

The background colour is usually painted by the default wxEraseEvent event handler function under Windows and automatically under GTK. Note that setting the background colour does not cause an immediate refresh, so you may wish to call wxWindow::ClearBackground or wxWindow::Refresh after calling this function. Using this function will disable attempts to use themes for this window, if the system supports them. Use with care since usually the themes represent the appearance chosen by the user to be used for all applications on the system.

Returns

true if the colour was really changed, false if it was already set to this colour and nothing was done.

See also

GetBackgroundColour(), SetForegroundColour(), GetForegroundColour(), ClearBackground(), Refresh(), wxEraseEvent, wxSystemSettings

Reimplemented in wxListCtrl.

virtual bool wxWindow::SetBackgroundStyle (wxBackgroundStyle *style*) [virtual]

Sets the background style of the window.

The default background style is `wxBG_STYLE_ERASE` which indicates that the window background may be erased in `EVT_ERASE_BACKGROUND` handler. This is a safe, compatibility default; however you may want to change it to `wxBG_STYLE_SYSTEM` if you don't define any erase background event handlers at all, to avoid unnecessary generation of erase background events and always let system erase the background. And you should change the background style to `wxBG_STYLE_PAINT` if you define an `EVT_PAINT` handler which completely overwrites the window background as in this case erasing it previously, either in `EVT_ERASE_BACKGROUND` handler or in the system default handler, would result in flicker as the background pixels will be repainted twice every time the window is redrawn. Do ensure that the background is entirely erased by your `EVT_PAINT` handler in this case however as otherwise garbage may be left on screen.

Notice that in previous versions of wxWidgets a common way to work around the above mentioned flickering problem was to define an empty `EVT_ERASE_BACKGROUND` handler. Setting background style to `wxBG_STYLE_PAINT` is a simpler and more efficient solution to the same problem.

Under wxGTK and wxOSX, you can use wxBG_STYLE_TRANSPARENT to obtain full transparency of the window background. Note that wxGTK supports this only since GTK 2.12 with a compositing manager enabled, call IsTransparentBackgroundSupported() to check whether this is the case.

Also, on order for `SetBackgroundStyle(wxBG_STYLE_TRANSPARENT)` to work, it must be called before Create(). If you're using your own wxWindow-derived class you should write your code in the following way:

```
class MyWidget : public wxWindow
{
public:
    MyWidget(wxWindow* parent, ...)
        : wxWindow() // Use default ctor here!
    {
        // Do this first:
        SetBackgroundStyle(wxBG_STYLE_TRANSPARENT);

        // And really create the window afterwards:
        Create(parent, ...);
    }
};
```

See also

SetBackgroundColour(), GetForegroundColour(), SetTransparent(), IsTransparentBackgroundSupported()

virtual void wxWindow::SetCanFocus (bool *canFocus*) [virtual]

This method is only implemented by ports which have support for native TAB traversal (such as GTK+ 2.0).

It is called by wxWidgets' container control code to give the native system a hint when doing TAB traversal. A call to this does not disable or change the effect of programmatically calling SetFocus().

See also

wxFocusEvent, wxPanel::SetFocus, wxPanel::SetFocusIgnoringChildren

void wxWindow::SetCaret (wxCaret * *caret*)

Sets the caret() associated with the window.

void wxWindow::SetClientSize (int *width,* int *height*)

This sets the size of the window client area in pixels.

Using this function to size a window tends to be more device-independent than SetSize(), since the application need not worry about what dimensions the border or title bar have when trying to fit the window around panel items, for example.

See also

Window Sizing Overview

void wxWindow::SetClientSize (const wxSize & *size*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxWindow::SetClientSize (const wxRect & *rect*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxWindow::SetConstraints (wxLayoutConstraints * *constraints*)

Sets the window to have the given layout constraints.

The window will then own the object, and will take care of its deletion. If an existing layout constraints object is already owned by the window, it will be deleted.

Parameters

constraints	The constraints to set. Pass NULL to disassociate and delete the window's constraints.

Remarks

You must call SetAutoLayout() to tell a window to use the constraints automatically in OnSize; otherwise, you must override OnSize and call Layout() explicitly. When setting both a wxLayoutConstraints and a wxSizer, only the sizer will have effect.

void wxWindow::SetContainingSizer (wxSizer * *sizer*)

This normally does not need to be called by user code.

It is called when a window is added to a sizer, and is used so the window can remove itself from the sizer when it is destroyed.

virtual bool wxWindow::SetCursor (const wxCursor & *cursor*) [virtual]

Sets the window's cursor.

Notice that the window cursor also sets it for the children of the window implicitly.

The *cursor* may be wxNullCursor in which case the window cursor will be reset back to default.

Parameters

cursor	Specifies the cursor that the window should normally display.

See also

wxSetCursor, wxCursor

void wxWindow::SetDoubleBuffered (bool *on*)

Turn on or off double buffering of the window if the system supports it.

virtual void wxWindow::SetDropTarget (wxDropTarget * *target*) `[virtual]`

Associates a drop target with this window.

If the window already has a drop target, it is deleted.

See also

GetDropTarget(), Drag and Drop Overview

void wxWindow::SetEventHandler (wxEvtHandler * *handler*)

Sets the event handler for this window.

Note that if you use this function you may want to use as the "next" handler of *handler* the window itself; in this way when *handler* doesn't process an event, the window itself will have a chance to do it.

Parameters

handler	Specifies the handler to be set. Cannot be NULL.

See also

How Events are Processed

virtual void wxWindow::SetExtraStyle (long *exStyle*) `[virtual]`

Sets the extra style bits for the window.

The currently defined extra style bits are reported in the class description.

virtual void wxWindow::SetFocus ( ) `[virtual]`

This sets the window to receive keyboard input.

See also

HasFocus(), wxFocusEvent, wxPanel::SetFocus, wxPanel::SetFocusIgnoringChildren

Reimplemented in wxPanel.

virtual void wxWindow::SetFocusFromKbd ( ) [virtual]

This function is called by wxWidgets keyboard navigation code when the user gives the focus to this window from keyboard (e.g. using TAB key).

By default this method simply calls SetFocus() but can be overridden to do something in addition to this in the derived classes.

virtual bool wxWindow::SetFont (const wxFont & *font*) [virtual]

Sets the font for this window.

This function should not be called for the parent window if you don't want its font to be inherited by its children, use SetOwnFont() instead in this case and see InheritAttributes() for more explanations.

Please notice that the given font is not automatically used for wxPaintDC objects associated with this window, you need to call wxDC::SetFont too. However this font is used by any standard controls for drawing their text as well as by GetTextExtent().

Parameters

font	Font to associate with this window, pass wxNullFont to reset to the default font.

Returns

true if the font was really changed, false if it was already set to this font and nothing was done.

See also

GetFont(), InheritAttributes()

Reimplemented in wxRichTextCtrl, wxAuiToolBar, wxAuiNotebook, and wxInfoBar.

virtual bool wxWindow::SetForegroundColour (const wxColour & *colour*) [virtual]

Sets the foreground colour of the window.

The meaning of foreground colour varies according to the window class; it may be the text colour or other colour, or it may not be used at all. Additionally, not all native controls support changing their foreground colour so this method may change their colour only partially or even not at all.

Please see InheritAttributes() for explanation of the difference between this method and SetOwnForeground↩ Colour().

Parameters

colour	The colour to be used as the foreground colour; pass wxNullColour to reset to the default colour.

Returns

true if the colour was really changed, false if it was already set to this colour and nothing was done.

See also

GetForegroundColour(), SetBackgroundColour(), GetBackgroundColour(), ShouldInheritColours()

void wxWindow::SetHelpText (const wxString & *helpText*)

Sets the help text to be used as context-sensitive help for this window.

Note that the text is actually stored by the current wxHelpProvider implementation, and not in the window object itself.

See also

GetHelpText(), wxHelpProvider::AddHelp()

void wxWindow::SetId (wxWindowID *winid*)

Sets the identifier of the window.

Remarks

Each window has an integer identifier. If the application has not provided one, an identifier will be generated. Normally, the identifier should be provided on creation and should not be modified subsequently.

See also

GetId(), Window IDs

virtual void wxWindow::SetInitialBestSize (const wxSize & *size*) [protected], [virtual]

Sets the initial window size if none is given (i.e. at least one of the components of the size passed to ctor/Create() is wxDefaultCoord).

Deprecated Use SetInitialSize() instead.

void wxWindow::SetInitialSize (const wxSize & *size* = wxDefaultSize)

A *smart* SetSize that will fill in default size components with the window's *best* size values.

Also sets the window's minsize to the value passed in for use with sizers. This means that if a full or partial size is passed to this function then the sizers will use that size instead of the results of GetBestSize() to determine the minimum needs of the window for layout.

Most controls will use this to set their initial size, and their min size to the passed in value (if any.)

See also

SetSize(), GetBestSize(), GetEffectiveMinSize(), Window Sizing Overview

virtual void wxWindow::SetLabel (const wxString & *label*) [virtual]

Sets the window's label.

Parameters

label	The window label.

See also

GetLabel()

Reimplemented in wxButton, wxControl, and wxCommandLinkButton.

virtual void wxWindow::SetLayoutDirection (wxLayoutDirection *dir*) [virtual]

Sets the layout direction for this window.

virtual void wxWindow::SetMaxClientSize (const wxSize & *size*) [virtual]

Sets the maximum client size of the window, to indicate to the sizer layout mechanism that this is the maximum possible size of its client area.

Note that this method is just a shortcut for:

```
SetMaxSize(ClientToWindowSize(size));
```

See also

SetMaxSize(), Window Sizing Overview

virtual void wxWindow::SetMaxSize (const wxSize & *size*) [virtual]

Sets the maximum size of the window, to indicate to the sizer layout mechanism that this is the maximum possible size.

See also

SetMaxClientSize(), Window Sizing Overview

Reimplemented in wxTopLevelWindow.

virtual void wxWindow::SetMinClientSize (const wxSize & *size*) [virtual]

Sets the minimum client size of the window, to indicate to the sizer layout mechanism that this is the minimum required size of window's client area.

You may need to call this if you change the window size after construction and before adding to its parent sizer.

Note, that just as with SetMinSize(), calling this method doesn't prevent the program from explicitly making the window smaller than the specified size.

Note that this method is just a shortcut for:

```
SetMinSize(ClientToWindowSize(size));
```

See also

SetMinSize(), Window Sizing Overview

virtual void wxWindow::SetMinSize (const wxSize & *size*) [virtual]

Sets the minimum size of the window, to indicate to the sizer layout mechanism that this is the minimum required size.

You may need to call this if you change the window size after construction and before adding to its parent sizer.

Notice that calling this method doesn't prevent the program from making the window explicitly smaller than the specified size by calling SetSize(), it just ensures that it won't become smaller than this size during the automatic layout.

See also

SetMinClientSize(), Window Sizing Overview

Reimplemented in wxTopLevelWindow.

virtual void wxWindow::SetName (const wxString & *name*) [virtual]

Sets the window's name.

Parameters

name	A name to set for the window.

See also

GetName()

virtual void wxWindow::SetNextHandler (wxEvtHandler * *handler*) [virtual]

wxWindows cannot be used to form event handler chains; this function thus will assert when called.

Note that instead you can use PushEventHandler() or SetEventHandler() to implement a stack of event handlers to override wxWindow's own event handling mechanism.

Reimplemented from wxEvtHandler.

void wxWindow::SetOwnBackgroundColour (const wxColour & *colour*)

Sets the background colour of the window but prevents it from being inherited by the children of this window.

See also

SetBackgroundColour(), InheritAttributes()

void wxWindow::SetOwnFont (const wxFont & *font*)

Sets the font of the window but prevents it from being inherited by the children of this window.

See also

SetFont(), InheritAttributes()

void wxWindow::SetOwnForegroundColour (const wxColour & *colour*)

Sets the foreground colour of the window but prevents it from being inherited by the children of this window.

See also

SetForegroundColour(), InheritAttributes()

void wxWindow::SetPalette (const wxPalette & *pal*)

Deprecated use wxDC::SetPalette instead.

void wxWindow::SetPosition (const wxPoint & *pt*)

A synonym for Centre().

virtual void wxWindow::SetPreviousHandler (wxEvtHandler * *handler*) [virtual]

wxWindows cannot be used to form event handler chains; this function thus will assert when called.

Note that instead you can use PushEventHandler() or SetEventHandler() to implement a stack of event handlers to override wxWindow's own event handling mechanism.

Reimplemented from wxEvtHandler.

virtual void wxWindow::SetScrollbar (int *orientation*, int *position*, int *thumbSize*, int *range*, bool *refresh* = true) [virtual]

Sets the scrollbar properties of a built-in scrollbar.

Parameters

orientation	Determines the scrollbar whose page size is to be set. May be wxHORIZONTAL or wxVERTICAL.
position	The position of the scrollbar in scroll units.
thumbSize	The size of the thumb, or visible portion of the scrollbar, in scroll units.
range	The maximum position of the scrollbar. Value of -1 can be used to ask for the scrollbar to be shown but in the disabled state: this can be used to avoid removing the scrollbar even when it is not needed (currently this is only implemented in wxMSW port).
refresh	true to redraw the scrollbar, false otherwise.

Remarks

Let's say you wish to display 50 lines of text, using the same font. The window is sized so that you can only see 16 lines at a time. You would use:

```
SetScrollbar(wxVERTICAL, 0, 16, 50);
```

Note that with the window at this size, the thumb position can never go above 50 minus 16, or 34. You can determine how many lines are currently visible by dividing the current view size by the character height in pixels. When defining your own scrollbar behaviour, you will always need to recalculate the scrollbar settings when the window size changes. You could therefore put your scrollbar calculations and SetScrollbar call into a function named AdjustScrollbars, which can be called initially and also from your wxSizeEvent handler function.

See also

Scrolled Windows, wxScrollBar, wxScrolled, wxScrollWinEvent

Reimplemented in wxScrollBar.

virtual void wxWindow::SetScrollPos (int *orientation,* int *pos,* bool *refresh* = `true`) [virtual]

Sets the position of one of the built-in scrollbars.

Parameters

orientation	Determines the scrollbar whose position is to be set. May be wxHORIZONTAL or wxVERTICAL.
pos	Position in scroll units.
refresh	true to redraw the scrollbar, false otherwise.

Remarks

This function does not directly affect the contents of the window: it is up to the application to take note of scrollbar attributes and redraw contents accordingly.

See also

SetScrollbar(), GetScrollPos(), GetScrollThumb(), wxScrollBar, wxScrolled

void wxWindow::SetSize (int *x,* int *y,* int *width,* int *height,* int *sizeFlags* = **wxSIZE_AUTO)**

Sets the size of the window in pixels.

Parameters

x	Required x position in pixels, or wxDefaultCoord to indicate that the existing value should be used.
y	Required y position in pixels, or wxDefaultCoord to indicate that the existing value should be used.
width	Required width in pixels, or wxDefaultCoord to indicate that the existing value should be used.
height	Required height position in pixels, or wxDefaultCoord to indicate that the existing value should be used.
sizeFlags	Indicates the interpretation of other parameters. It is a bit list of the following: • `wxSIZE_AUTO_WIDTH`: a wxDefaultCoord width value is taken to indicate a wxWidgets-supplied default width. • `wxSIZE_AUTO_HEIGHT`: a wxDefaultCoord height value is taken to indicate a wxWidgets-supplied default height. • `wxSIZE_AUTO`: wxDefaultCoord size values are taken to indicate a wxWidgets-supplied default size. • `wxSIZE_USE_EXISTING`: existing dimensions should be used if wxDefaultCoord values are supplied. • `wxSIZE_ALLOW_MINUS_ONE`: allow negative dimensions (i.e. value of wxDefaultCoord) to be interpreted as real dimensions, not default values. • `wxSIZE_FORCE`: normally, if the position and the size of the window are already the same as the parameters of this function, nothing is done. but with this flag a window resize may be forced even in this case (supported in wx 2.6.2 and later and only implemented for MSW and ignored elsewhere currently).

Remarks

This overload sets the position and optionally size, of the window. Parameters may be wxDefaultCoord to indicate either that a default should be supplied by wxWidgets, or that the current value of the dimension should be used.

See also

Move(), Window Sizing Overview

void wxWindow::SetSize (const wxRect & *rect*)

Sets the size of the window in pixels.

The size is specified using a wxRect, wxSize or by a couple of `int` objects.

Remarks

This form must be used with non-default width and height values.

See also

Move(), Window Sizing Overview

void wxWindow::SetSize (const wxSize & *size*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxWindow::SetSize (int *width,* int *height*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

virtual void wxWindow::SetSizeHints (const wxSize & *minSize,* const wxSize & *maxSize* = wxDefaultSize, const wxSize & *incSize* = wxDefaultSize) [virtual]

Use of this function for windows which are not toplevel windows (such as wxDialog or wxFrame) is discouraged.

Please use SetMinSize() and SetMaxSize() instead.

See also

wxTopLevelWindow::SetSizeHints, Window Sizing Overview

Reimplemented in wxTopLevelWindow.

virtual void wxWindow::SetSizeHints (int *minW,* int *minH,* int *maxW* = -1, int *maxH* = -1, int *incW* = -1, int *incH* = -1) [virtual]

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Reimplemented in wxTopLevelWindow.

void wxWindow::SetSizer (wxSizer * *sizer,* bool *deleteOld* = true)

Sets the window to have the given layout sizer.

The window will then own the object, and will take care of its deletion. If an existing layout constraints object is already owned by the window, it will be deleted if the *deleteOld* parameter is true.

Note that this function will also call SetAutoLayout() implicitly with true parameter if the *sizer* is non-NULL and false otherwise so that the sizer will be effectively used to layout the window children whenever it is resized.

Parameters

sizer	The sizer to set. Pass NULL to disassociate and conditionally delete the window's sizer. See below.
deleteOld	If true (the default), this will delete any pre-existing sizer. Pass false if you wish to handle deleting the old sizer yourself but remember to do it yourself in this case to avoid memory leaks.

Remarks

SetSizer enables and disables Layout automatically.

void wxWindow::SetSizerAndFit (wxSizer * *sizer,* bool *deleteOld* = `true`)

This method calls SetSizer() and then wxSizer::SetSizeHints which sets the initial window size to the size needed to accommodate all sizer elements and sets the size hints which, if this window is a top level one, prevent the user from resizing it to be less than this minimal size.

virtual void wxWindow::SetThemeEnabled (bool *enable*) `[virtual]`

This function tells a window if it should use the system's "theme" code to draw the windows' background instead of its own background drawing code.

This does not always have any effect since the underlying platform obviously needs to support the notion of themes in user defined windows. One such platform is GTK+ where windows can have (very colourful) backgrounds defined by a user's selected theme.

Dialogs, notebook pages and the status bar have this flag set to true by default so that the default look and feel is simulated best.

void wxWindow::SetToolTip (const wxString & *tipString*)

Attach a tooltip to the window.

wxToolTip pointer can be NULL in the overload taking the pointer, meaning to unset any existing tooltips; however UnsetToolTip() provides a more readable alternative to this operation.

Notice that these methods are always available, even if wxWidgets was compiled with `wxUSE_TOOLTIPS` set to 0, but don't do anything in this case.

See also

GetToolTip(), wxToolTip

void wxWindow::SetToolTip (wxToolTip * *tip*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

virtual bool wxWindow::SetTransparent (wxByte *alpha*) `[virtual]`

Set the transparency of the window.

If the system supports transparent windows, returns true, otherwise returns false and the window remains fully opaque. See also CanSetTransparent().

The parameter *alpha* is in the range 0..255 where 0 corresponds to a fully transparent window and 255 to the fully opaque one. The constants `wxIMAGE_ALPHA_TRANSPARENT` and `wxIMAGE_ALPHA_OPAQUE` can be used.

Reimplemented in wxTopLevelWindow.

virtual void wxWindow::SetValidator (const wxValidator & *validator*) `[virtual]`

Deletes the current validator (if any) and sets the window validator, having called wxValidator::Clone to create a new validator of this type.

void wxWindow::SetVirtualSize (int *width,* int *height*)

Sets the virtual size of the window in pixels.

See also

Window Sizing Overview

void wxWindow::SetVirtualSize (const wxSize & *size*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

void wxWindow::SetWindowStyle (long *style*)

See SetWindowStyleFlag() for more info.

virtual void wxWindow::SetWindowStyleFlag (long *style*) `[virtual]`

Sets the style of the window.

Please note that some styles cannot be changed after the window creation and that Refresh() might need to be called after changing the others for the change to take place immediately.

See Window styles for more information about flags.

See also

GetWindowStyleFlag()

Reimplemented in wxListCtrl, and wxAuiToolBar.

void wxWindow::SetWindowVariant (wxWindowVariant *variant*)

Chooses a different variant of the window display to use.

Window variants currently just differ in size, as can be seen from wxWindowVariant documentation. Under all platforms but OS X, this function does nothing more than change the font used by the window. However under OS X it is implemented natively and selects the appropriate variant of the native widget, which has better appearance than just scaled down or up version of the normal variant, so it should be preferred to directly tweaking the font size.

By default the controls naturally use the normal variant.

virtual bool wxWindow::ShouldInheritColours ( ) const `[virtual]`

Return true from here to allow the colours of this window to be changed by InheritAttributes().

Returning false forbids inheriting them from the parent window.

The base class version returns false, but this method is overridden in wxControl where it returns true.

Reimplemented in wxRichTextCtrl.

virtual bool wxWindow::Show (bool *show* = `true`) `[virtual]`

Shows or hides the window.

You may need to call Raise() for a top level window if you want to bring it to top, although this is not needed if Show() is called immediately after the frame creation.

Notice that the default state of newly created top level windows is hidden (to allow you to create their contents without flicker) unlike for all the other, not derived from wxTopLevelWindow, windows that are by default created in the shown state.

Parameters

show	If true displays the window. Otherwise, hides it.

Returns

true if the window has been shown or hidden or false if nothing was done because it already was in the requested state.

See also

IsShown(), Hide(), wxRadioBox::Show, wxShowEvent.

Reimplemented in wxDialog.

virtual bool wxWindow::ShowWithEffect (wxShowEffect *effect*, unsigned int *timeout* = `0`) `[virtual]`

This function shows a window, like Show(), but using a special visual effect if possible.

Parameters

effect	The effect to use.
timeout	The *timeout* parameter specifies the time of the animation, in milliseconds. If the default value of 0 is used, the default animation time for the current platform is used.

Note

Currently this function is only implemented in wxMSW and wxOSX (for wxTopLevelWindows only in Carbon version and for any kind of windows in Cocoa) and does the same thing as Show() in the other ports.

Since

2.9.0

See also

HideWithEffect()

void wxWindow::Thaw ( )

Re-enables window updating after a previous call to Freeze().

To really thaw the control, it must be called exactly the same number of times as Freeze().

If the window has any children, they are recursively thawed too.

See also

wxWindowUpdateLocker, Freeze(), IsFrozen()

bool wxWindow::ToggleWindowStyle (int *flag*)

Turns the given *flag* on if it's currently turned off and vice versa.

This function cannot be used if the value of the flag is 0 (which is often the case for default flags).

Also, please notice that not all styles can be changed after the control creation.

Returns

Returns true if the style was turned on by this function, false if it was switched off.

See also

SetWindowStyleFlag(), HasFlag()

virtual bool wxWindow::TransferDataFromWindow ( ) [virtual]

Transfers values from child controls to data areas specified by their validators.

Returns false if a transfer failed.

If the window has `wxWS_EX_VALIDATE_RECURSIVELY` extra style flag set, the method will also call TransferDataFromWindow() of all child windows.

See also

TransferDataToWindow(), wxValidator, Validate()

virtual bool wxWindow::TransferDataToWindow ( ) [virtual]

Transfers values to child controls from data areas specified by their validators.

If the window has `wxWS_EX_VALIDATE_RECURSIVELY` extra style flag set, the method will also call TransferDataToWindow() of all child windows.

Returns

Returns false if a transfer failed.

See also

TransferDataFromWindow(), wxValidator, Validate()

virtual bool wxWindow::UnregisterHotKey (int *hotkeyId*) [virtual]

Unregisters a system wide hotkey.

Parameters

hotkeyId	Numeric identifier of the hotkey. Must be the same id that was passed to RegisterHotKey().

Returns

true if the hotkey was unregistered successfully, false if the id was invalid.

Remarks

This function is currently only implemented under MSW.

See also

RegisterHotKey()

static void wxWindow::UnreserveControlId (wxWindowID *id*, int *count* = 1) [static]

Unreserve an ID or range of IDs that was reserved by NewControlId().

See Window IDs for more information.

Parameters

id	The starting ID of the range of IDs to unreserve.
count	The number of sequential IDs to unreserve.

See also

NewControlId(), wxIdManager, Window IDs

void wxWindow::UnsetToolTip ( )

Unset any existing tooltip.

Since

2.9.0

See also

SetToolTip()

virtual void wxWindow::Update ( ) [virtual]

Calling this method immediately repaints the invalidated area of the window and all of its children recursively (this normally only happens when the flow of control returns to the event loop).

Notice that this function doesn't invalidate any area of the window so nothing happens if nothing has been invalidated (i.e. marked as requiring a redraw). Use Refresh() first if you want to immediately redraw the window unconditionally.

virtual void wxWindow::UpdateWindowUI (long *flags* = wxUPDATE_UI_NONE) [virtual]

This function sends one or more wxUpdateUIEvent to the window.

The particular implementation depends on the window; for example a wxToolBar will send an update UI event for each toolbar button, and a wxFrame will send an update UI event for each menubar menu item.

You can call this function from your application to ensure that your UI is up-to-date at this point (as far as your wxUpdateUIEvent handlers are concerned). This may be necessary if you have called wxUpdateUIEvent::SetMode() or wxUpdateUIEvent::SetUpdateInterval() to limit the overhead that wxWidgets incurs by sending update UI events in idle time. *flags* should be a bitlist of one or more of the wxUpdateUI enumeration.

If you are calling this function from an OnInternalIdle or OnIdle function, make sure you pass the wxUPDATE_UI_FROMIDLE flag, since this tells the window to only update the UI elements that need to be updated in idle time. Some windows update their elements only when necessary, for example when a menu is about to be shown. The following is an example of how to call UpdateWindowUI from an idle function.

```
void MyWindow::OnInternalIdle()
{
    if (wxUpdateUIEvent::CanUpdate(this))
        UpdateWindowUI(wxUPDATE_UI_FROMIDLE);
}
```

See also

wxUpdateUIEvent, DoUpdateWindowUI(), OnInternalIdle()

bool wxWindow::UseBgCol ( ) const

Return true if a background colour has been set for this window.

virtual bool wxWindow::Validate ( ) [virtual]

Validates the current values of the child controls using their validators.

If the window has `wxWS_EX_VALIDATE_RECURSIVELY` extra style flag set, the method will also call Validate() of all child windows.

Returns

Returns false if any of the validations failed.

See also

TransferDataFromWindow(), TransferDataToWindow(), wxValidator

virtual void wxWindow::WarpPointer (int *x*, int *y*) [virtual]

Moves the pointer to the given position on the window.

Note

Apple Human Interface Guidelines forbid moving the mouse cursor programmatically so you should avoid using this function in Mac applications (and probably avoid using it under the other platforms without good reason as well).

Parameters

x	The new x position for the cursor.
y	The new y position for the cursor.

virtual wxSize wxWindow::WindowToClientSize (const wxSize & *size*) const `[virtual]`

Converts window size *size* to corresponding client area size In other words, the returned value is what would GetClientSize() return if this window had given window size.

Components with wxDefaultCoord value are left unchanged.

Note that the conversion is not always exact, it assumes that non-client area doesn't change and so doesn't take into account things like menu bar (un)wrapping or (dis)appearance of the scrollbars.

Since

2.8.8

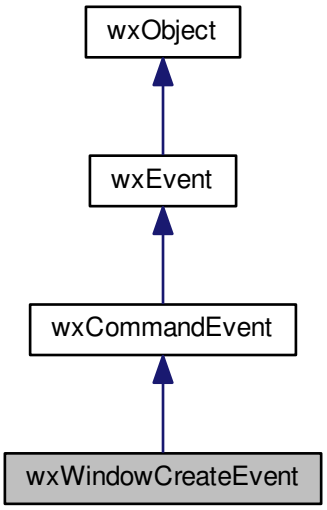
See also

ClientToWindowSize()

21.855 wxWindowCreateEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxWindowCreateEvent:



21.855.1 Detailed Description

This event is sent just after the actual window associated with a wxWindow object has been created.

Since it is derived from wxCommandEvent, the event propagates up the window hierarchy.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxWindowCreateEvent& event)

Event macros:

- EVT_WINDOW_CREATE(func): Process a `wxEVT_CREATE` event.

Library: wxCore

Category: Events

See also

Events and Event Handling, wxWindowDestroyEvent

Public Member Functions

- wxWindowCreateEvent (wxWindow *win=NULL)

  Constructor.

- wxWindow * GetWindow () const

  Return the window being created.

Additional Inherited Members

21.855.2 Constructor & Destructor Documentation

wxWindowCreateEvent::wxWindowCreateEvent (wxWindow * *win* = `NULL`)

Constructor.

21.855.3 Member Function Documentation

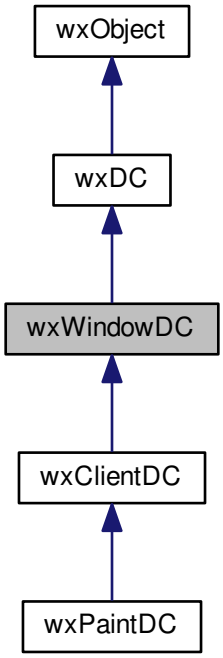
wxWindow* wxWindowCreateEvent::GetWindow ( ) const

Return the window being created.

21.856 wxWindowDC Class Reference

```
#include <wx/dcclient.h>
```

Inheritance diagram for wxWindowDC:



21.856.1 Detailed Description

A wxWindowDC must be constructed if an application wishes to paint on the whole area of a window (client and decorations).

This should normally be constructed as a temporary stack object; don't store a wxWindowDC object.

To draw on a window from inside an EVT_PAINT() handler, construct a wxPaintDC object instead.

To draw on the client area of a window from outside an EVT_PAINT() handler, construct a wxClientDC object.

A wxWindowDC object is initialized to use the same font and colours as the window it is associated with.

Library: wxCore

Category: Device Contexts

See also

wxDC, wxMemoryDC, wxPaintDC, wxClientDC, wxScreenDC

Public Member Functions

- wxWindowDC (wxWindow *window)

  Constructor.

Additional Inherited Members

21.856.2 Constructor & Destructor Documentation

wxWindowDC::wxWindowDC (wxWindow * *window*)

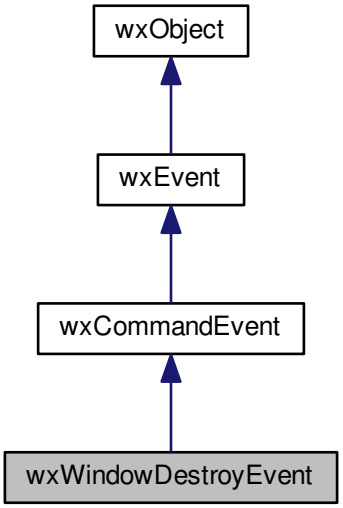
Constructor.

Pass a pointer to the window on which you wish to paint.

21.857 wxWindowDestroyEvent Class Reference

`#include <wx/event.h>`

Inheritance diagram for wxWindowDestroyEvent:



21.857.1 Detailed Description

This event is sent as early as possible during the window destruction process.

For the top level windows, as early as possible means that this is done by wxFrame or wxDialog destructor, i.e. after the destructor of the derived class was executed and so any methods specific to the derived class can't be called any more from this event handler. If you need to do this, you must call wxWindow::SendDestroyEvent() from your derived class destructor.

For the child windows, this event is generated just before deleting the window from wxWindow::Destroy() (which is also called when the parent window is deleted) or from the window destructor if operator `delete` was used directly (which is not recommended for this very reason).

It is usually pointless to handle this event in the window itself but it ca be very useful to receive notifications about the window destruction in the parent window or in any other object interested in this window.

Library: wxCore

Category: Events

See also

Events and Event Handling, wxWindowCreateEvent

Public Member Functions

- wxWindowDestroyEvent (wxWindow *win=NULL)

  Constructor.
- wxWindow * GetWindow () const

  Return the window being destroyed.

Additional Inherited Members

21.857.2 Constructor & Destructor Documentation

wxWindowDestroyEvent::wxWindowDestroyEvent (wxWindow * *win* = NULL)

Constructor.

21.857.3 Member Function Documentation

wxWindow* wxWindowDestroyEvent::GetWindow ( ) const

Return the window being destroyed.

21.858 wxWindowDisabler Class Reference

```
#include <wx/utils.h>
```

21.858.1 Detailed Description

This class disables all windows of the application (may be with the exception of one of them) in its constructor and enables them back in its destructor.

This is useful when you want to indicate to the user that the application is currently busy and cannot respond to user input.

Library: wxCore

Category: Miscellaneous

See also

wxBusyCursor

Public Member Functions

- wxWindowDisabler (bool disable=true)

  Disables all top level windows of the applications.

- wxWindowDisabler (wxWindow *winToSkip)

  Disables all top level windows of the applications with the exception of winToSkip if it is not NULL.

- ~wxWindowDisabler ()

  Reenables the windows disabled by the constructor.

21.858.2 Constructor & Destructor Documentation

wxWindowDisabler::wxWindowDisabler (bool *disable* = `true`)

Disables all top level windows of the applications.

If *disable* is `false` nothing is done. This can be convenient if the windows should be disabled depending on some condition.

Since

2.9.0

wxWindowDisabler::wxWindowDisabler (wxWindow * *winToSkip*)

Disables all top level windows of the applications with the exception of *winToSkip* if it is not NULL.

Notice that under MSW if *winToSkip* appears in the taskbar, the user will be able to close the entire application (even though its main window is disabled) by right clicking on the taskbar icon and selecting the appropriate "Close" command from the context menu. To prevent this from happening you may want to use wxFRAME_TOOL_WIND↩OW, if applicable, or wxFRAME_NO_TASKBAR style when creating the window that will remain enabled.

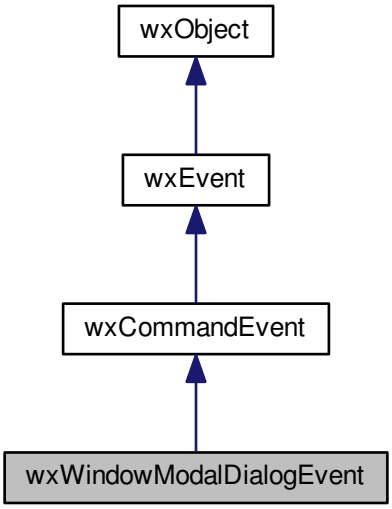
wxWindowDisabler::~wxWindowDisabler ( )

Reenables the windows disabled by the constructor.

21.859 wxWindowModalDialogEvent Class Reference

`#include <wx/dialog.h>`

Inheritance diagram for wxWindowModalDialogEvent:



21.859.1 Detailed Description

Event sent by wxDialog::ShowWindowModal() when the dialog closes.

Since

2.9.0

Public Member Functions

- wxWindowModalDialogEvent (wxEventType commandType=wxEVT_NULL, int id=0)

  Constructor.
- wxDialog * GetDialog () const

  Return the corresponding dialog.
- int GetReturnCode () const

  Return the dialog's return code.
- virtual wxEvent * Clone () const

  Clone the event.

Additional Inherited Members

21.859.2 Constructor & Destructor Documentation

wxWindowModalDialogEvent::wxWindowModalDialogEvent (wxEventType *commandType* = wxEVT_NULL, int *id* = 0)

Constructor.

21.859.3 Member Function Documentation

virtual wxEvent* wxWindowModalDialogEvent::Clone ( ) const `[virtual]`

Clone the event.

Implements wxEvent.

wxDialog* wxWindowModalDialogEvent::GetDialog ( ) const

Return the corresponding dialog.

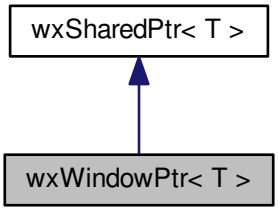
int wxWindowModalDialogEvent::GetReturnCode ( ) const

Return the dialog's return code.

21.860 wxWindowPtr< T > Class Template Reference

`#include <wx/windowptr.h>`

Inheritance diagram for wxWindowPtr< T >:



21.860.1 Detailed Description

template<typename T>class wxWindowPtr< T >

A reference-counted smart pointer for holding wxWindow instances.

This specialization of wxSharedPtr<T> is useful for holding wxWindow-derived objects. Unlike wxSharedPtr<T> or `std::shared_ptr<>`, it doesn't use the delete operator to destroy the value when reference count drops to zero, but calls wxWindow::Destroy() to safely destroy the window.

The template parameter T must be wxWindow or a class derived from it.

Library: wxCore

Category: Smart Pointers

Since

3.0

See also

wxSharedPtr<T>

Public Member Functions

- wxWindowPtr ()

  Default constructor.
- wxWindowPtr (T *ptr)

  Constructor.
- template<typename Deleter >
  wxWindowPtr (T *ptr, Deleter d)

  Constructor.
- wxWindowPtr (const wxWindowPtr< T > &tocopy)

  Copy constructor.
- wxWindowPtr< T > & operator= (T *ptr)

  Assignment operator.
- wxWindowPtr< T > & operator= (const wxWindowPtr< T > &tocopy)

  Assignment operator.
- void reset (T *ptr=NULL)

  Reset pointer to ptr.

21.860.2 Constructor & Destructor Documentation

template<typename T > wxWindowPtr< T >::wxWindowPtr ( )

Default constructor.

template<typename T > wxWindowPtr< T >::wxWindowPtr (T * *ptr*) `[explicit]`

Constructor.

Creates shared pointer from the raw pointer *ptr* and takes ownership of it.

template<typename T > template<typename Deleter > wxWindowPtr< T >::wxWindowPtr (T * *ptr*, Deleter *d*) `[explicit]`

Constructor.

Creates shared pointer from the raw pointer *ptr* and deleter *d* and takes ownership of it.

Parameters

ptr	The raw pointer.
d	Deleter - a functor that is called instead of delete to free the *ptr* raw pointer when its reference count drops to zero.

template<typename T > wxWindowPtr< T >::wxWindowPtr (const wxWindowPtr< T > & *tocopy*)

Copy constructor.

21.860.3 Member Function Documentation

template<typename T > wxWindowPtr<T>& wxWindowPtr< T >::operator= (T * *ptr*)

Assignment operator.

Releases any previously held pointer and creates a reference to *ptr*.

template<typename T > wxWindowPtr<T>& wxWindowPtr< T >::operator= (const wxWindowPtr< T > & *tocopy*)

Assignment operator.

Releases any previously held pointer and creates a reference to the same object as *topcopy*.

template<typename T > void wxWindowPtr< T >::reset (T * *ptr* = NULL)

Reset pointer to *ptr*.

If the reference count of the previously owned pointer was 1 it will be deleted.

21.861 wxWindowUpdateLocker Class Reference

`#include <wx/wupdlock.h>`

21.861.1 Detailed Description

This tiny class prevents redrawing of a wxWindow during its lifetime by using wxWindow::Freeze() and wxWindow::Thaw() methods.

It is typically used for creating automatic objects to temporarily suppress window updates before a batch of operations is performed:

```
void MyFrame::Foo()
    {
        m_text = new wxTextCtrl(this, ...);

        wxWindowUpdateLocker noUpdates(m_text);
        m_text->AppendText();
        ... many other operations with m_text...
        m_text->WriteText();
    }
```

Using this class is easier and safer than calling wxWindow::Freeze() and wxWindow::Thaw() because you don't risk to forget calling the latter.

Library: wxBase

Category: Miscellaneous

Public Member Functions

- wxWindowUpdateLocker (wxWindow *win)

  Creates an object preventing the updates of the specified win.
- ~wxWindowUpdateLocker ()

  Destructor reenables updates for the window this object is associated with.

21.861.2 Constructor & Destructor Documentation

wxWindowUpdateLocker::wxWindowUpdateLocker (wxWindow * *win*)

Creates an object preventing the updates of the specified *win*.

The parameter must be non-NULL and the window must exist for longer than wxWindowUpdateLocker object itself.

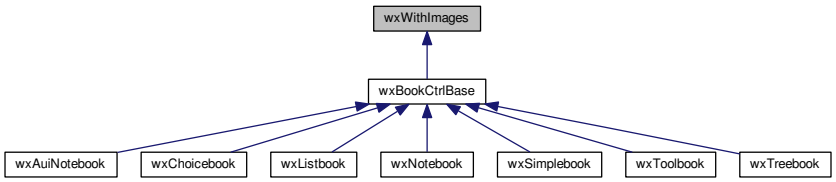
wxWindowUpdateLocker::~wxWindowUpdateLocker ( )

Destructor reenables updates for the window this object is associated with.

21.862 wxWithImages Class Reference

`#include <wx/withimages.h>`

Inheritance diagram for wxWithImages:



21.862.1 Detailed Description

A mixin class to be used with other classes that use a wxImageList.

Public Types

- enum { NO_IMAGE = -1 }

Public Member Functions

- wxWithImages ()
- virtual ~wxWithImages ()
- void AssignImageList (wxImageList *imageList)

  Sets the image list for the page control and takes ownership of the list.
- virtual void SetImageList (wxImageList *imageList)

  Sets the image list to use.
- wxImageList * GetImageList () const

  Returns the associated image list, may be NULL.

Protected Member Functions

- bool HasImageList () const

  Return true if we have a valid image list.
- wxIcon GetImage (int iconIndex) const

  Return the image with the given index from the image list.

21.862.2 Member Enumeration Documentation

anonymous enum

Enumerator

NO_IMAGE

21.862.3 Constructor & Destructor Documentation

wxWithImages::wxWithImages ( )

virtual wxWithImages::~wxWithImages ( ) `[virtual]`

21.862.4 Member Function Documentation

void wxWithImages::AssignImageList (wxImageList * *imageList*)

Sets the image list for the page control and takes ownership of the list.

See also

wxImageList, SetImageList()

wxIcon wxWithImages::GetImage (int *iconIndex*) const `[protected]`

Return the image with the given index from the image list.

If there is no image list or if index == NO_IMAGE, silently returns wxNullIcon.

wxImageList* wxWithImages::GetImageList ( ) const

Returns the associated image list, may be NULL.

See also

wxImageList, SetImageList()

bool wxWithImages::HasImageList ( ) const `[protected]`

Return true if we have a valid image list.

virtual void wxWithImages::SetImageList (wxImageList * *imageList*) `[virtual]`

Sets the image list to use.

It does not take ownership of the image list, you must delete it yourself.

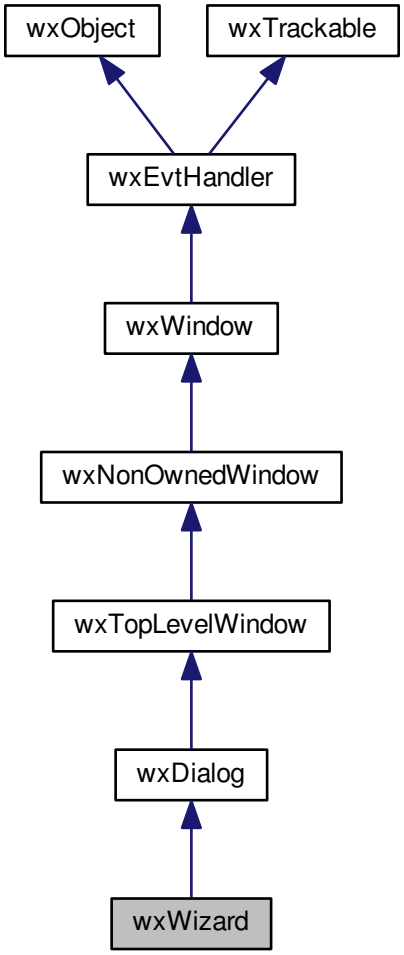
See also

wxImageList, AssignImageList()

21.863 wxWizard Class Reference

`#include <wx/wizard.h>`

Inheritance diagram for wxWizard:



21.863.1 Detailed Description

wxWizard is the central class for implementing 'wizard-like' dialogs.

These dialogs are mostly familiar to Windows users and are nothing other than a sequence of 'pages', each displayed inside a dialog which has the buttons to navigate to the next (and previous) pages.

The wizards are typically used to decompose a complex dialog into several simple steps and are mainly useful to the novice users, hence it is important to keep them as simple as possible.

To show a wizard dialog, you must first create an instance of the wxWizard class using either the non-default constructor or a default one followed by call to the wxWizard::Create function. Then you should add all pages you want the wizard to show and call wxWizard::RunWizard(). Finally, don't forget to call "`wizard->Destroy()`", otherwise your application will hang on exit due to an undestroyed window.

You can supply a bitmap to display on the left of the wizard, either for all pages or for individual pages. If you need to have the bitmap resize to the height of the wizard, call wxWizard::SetBitmapPlacement() and if necessary, wxWizard::SetBitmapBackgroundColour() and wxWizard::SetMinimumBitmapWidth().

To make wizard pages scroll when the display is too small to fit the whole dialog, you can switch layout adaptation on globally with wxDialog::EnableLayoutAdaptation() or per dialog with wxDialog::SetLayoutAdaptationMode(). For more about layout adaptation, see Automatic Scrolled Dialogs.

Events emitted by this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxWizardEvent& event)

Event macros for events emitted by this class: For some events, Veto() can be called to prevent the event from happening.

- EVT_WIZARD_PAGE_CHANGED(id, func): The page has just been changed (this event cannot be vetoed).
- EVT_WIZARD_PAGE_CHANGING(id, func): The page is being changed (this event can be vetoed).
- EVT_WIZARD_BEFORE_PAGE_CHANGED(id, func): Called after Next is clicked but before GetNext is called. Unlike EVT_WIZARD_CHANGING, the handler for this function can change state that might affect the return value of GetNext. This event can be vetoed.
- EVT_WIZARD_PAGE_SHOWN(id, func): The page was shown and laid out (this event cannot be vetoed).
- EVT_WIZARD_CANCEL(id, func): The user attempted to cancel the wizard (this event may also be vetoed).
- EVT_WIZARD_HELP(id, func): The wizard help button was pressed.
- EVT_WIZARD_FINISHED(id, func): The wizard finished button was pressed.

21.863.2 Extended styles

Use the wxWindow::SetExtraStyle() function to set the following style. You will need to use two-step construction (use the default constructor, call SetExtraStyle(), then call Create).

Extra Styles

This class supports the following extra styles:

- wxWIZARD_EX_HELPBUTTON: Shows a Help button using wxID_HELP.

See also wxDialog for other extended styles.

Library: wxAdvanced

Category: Common Dialogs

See also

wxWizardEvent, wxWizardPage, Wizard Sample

Public Member Functions

- wxWizard ()

  Default constructor.
- wxWizard (wxWindow *parent, int id=wxID_ANY, const wxString &title=wxEmptyString, const wxBitmap &bitmap=wxNullBitmap, const wxPoint &pos=wxDefaultPosition, long style=wxDEFAULT_DIALOG_STYLE)

  Constructor which really creates the wizard – if you use this constructor, you shouldn't call Create().
- bool Create (wxWindow *parent, int id=wxID_ANY, const wxString &title=wxEmptyString, const wxBitmap &bitmap=wxNullBitmap, const wxPoint &pos=wxDefaultPosition, long style=wxDEFAULT_DIALOG_STYLE)

  Creates the wizard dialog.
- virtual void FitToPage (const wxWizardPage *firstPage)

  This method is obsolete, use GetPageAreaSizer() instead.
- const wxBitmap & GetBitmap () const

  Returns the bitmap used for the wizard.
- const wxColour & GetBitmapBackgroundColour () const

  Returns the colour that should be used to fill the area not taken up by the wizard or page bitmap, if a non-zero bitmap placement flag has been set.
- int GetBitmapPlacement () const

  Returns the flags indicating how the wizard or page bitmap should be expanded and positioned to fit the page height.
- virtual wxWizardPage * GetCurrentPage () const

  Get the current page while the wizard is running.
- int GetMinimumBitmapWidth () const

  Returns the minimum width for the bitmap that will be constructed to contain the actual wizard or page bitmap if a non-zero bitmap placement flag has been set.
- virtual wxSizer * GetPageAreaSizer () const

  Returns pointer to page area sizer.
- virtual wxSize GetPageSize () const

  Returns the size available for the pages.
- virtual bool HasNextPage (wxWizardPage *page)

  Return true if this page is not the last one in the wizard.
- virtual bool HasPrevPage (wxWizardPage *page)

  Returns true if this page is not the last one in the wizard.
- virtual bool RunWizard (wxWizardPage *firstPage)

  Executes the wizard starting from the given page, returning true if it was successfully finished or false if user cancelled it.
- void SetBitmap (const wxBitmap &bitmap)

  Sets the bitmap used for the wizard.
- void SetBitmapBackgroundColour (const wxColour &colour)

  Sets the colour that should be used to fill the area not taken up by the wizard or page bitmap, if a non-zero bitmap placement flag has been set.
- void SetBitmapPlacement (int placement)

  Sets the flags indicating how the wizard or page bitmap should be expanded and positioned to fit the page height.
- virtual void SetBorder (int border)

  Sets width of border around page area.
- void SetMinimumBitmapWidth (int width)

  Sets the minimum width for the bitmap that will be constructed to contain the actual wizard or page bitmap if a non-zero bitmap placement flag has been set.
- virtual void SetPageSize (const wxSize &sizePage)

  Sets the minimal size to be made available for the wizard pages.

Additional Inherited Members

21.863.3 Constructor & Destructor Documentation

wxWizard::wxWizard ( )

Default constructor.

Use this if you wish to derive from wxWizard and then call Create(), for example if you wish to set an extra style with wxWindow::SetExtraStyle() between the two calls.

wxWizard::wxWizard (wxWindow * *parent*, int *id* = wxID_ANY, const wxString & *title* = wxEmptyString, const wxBitmap & *bitmap* = wxNullBitmap, const wxPoint & *pos* = wxDefaultPosition, long *style* = wxDEFAULT_DIALOG_STYLE)

Constructor which really creates the wizard – if you use this constructor, you shouldn't call Create().

Notice that unlike almost all other wxWidgets classes, there is no *size* parameter in the wxWizard constructor because the wizard will have a predefined default size by default. If you want to change this, you should use the GetPageAreaSizer() function.

Parameters

parent	The parent window, may be NULL.
id	The id of the dialog, will usually be just wxID_ANY.
title	The title of the dialog.
bitmap	The default bitmap used in the left side of the wizard. See also GetBitmap().
pos	The position of the dialog, it will be centered on the screen by default.
style	Window style is passed to wxDialog.

21.863.4 Member Function Documentation

bool wxWizard::Create (wxWindow * *parent*, int *id* = wxID_ANY, const wxString & *title* = wxEmptyString, const wxBitmap & *bitmap* = wxNullBitmap, const wxPoint & *pos* = wxDefaultPosition, long *style* = wxDEFAULT_DIALOG_STYLE)

Creates the wizard dialog.

Must be called if the default constructor had been used to create the object.

Notice that unlike almost all other wxWidgets classes, there is no *size* parameter in the wxWizard constructor because the wizard will have a predefined default size by default. If you want to change this, you should use the GetPageAreaSizer() function.

Parameters

parent	The parent window, may be NULL.
id	The id of the dialog, will usually be just -1.
title	The title of the dialog.
bitmap	The default bitmap used in the left side of the wizard. See also GetBitmap().
pos	The position of the dialog, it will be centered on the screen by default.
style	Window style is passed to wxDialog.

virtual void wxWizard::FitToPage (const wxWizardPage * *firstPage*) [virtual]

This method is obsolete, use GetPageAreaSizer() instead.

Sets the page size to be big enough for all the pages accessible via the given *firstPage*, i.e. this page, its next page and so on.

This method may be called more than once and it will only change the page size if the size required by the new page is bigger than the previously set one. This is useful if the decision about which pages to show is taken during run-time, as in this case, the wizard won't be able to get to all pages starting from a single one and you should call *Fit* separately for the others.

const wxBitmap& wxWizard::GetBitmap ( ) const

Returns the bitmap used for the wizard.

const wxColour& wxWizard::GetBitmapBackgroundColour ( ) const

Returns the colour that should be used to fill the area not taken up by the wizard or page bitmap, if a non-zero bitmap placement flag has been set.

See also SetBitmapPlacement().

int wxWizard::GetBitmapPlacement ( ) const

Returns the flags indicating how the wizard or page bitmap should be expanded and positioned to fit the page height.

By default, placement is 0 (no expansion is done).

See also SetBitmapPlacement() for the possible values.

virtual wxWizardPage* wxWizard::GetCurrentPage ( ) const [virtual]

Get the current page while the wizard is running.

NULL is returned if RunWizard() is not being executed now.

int wxWizard::GetMinimumBitmapWidth ( ) const

Returns the minimum width for the bitmap that will be constructed to contain the actual wizard or page bitmap if a non-zero bitmap placement flag has been set.

See also SetBitmapPlacement().

virtual wxSizer* wxWizard::GetPageAreaSizer ( ) const [virtual]

Returns pointer to page area sizer.

The wizard is laid out using sizers and the page area sizer is the place-holder for the pages. All pages are resized before being shown to match the wizard page area.

Page area sizer has a minimal size that is the maximum of several values. First, all pages (or other objects) added to the sizer. Second, all pages reachable by repeatedly applying wxWizardPage::GetNext() to any page inserted into the sizer.

Third, the minimal size specified using SetPageSize() and FitToPage(). Fourth, the total wizard height may be increased to accommodate the bitmap height. Fifth and finally, wizards are never smaller than some built-in minimal size to avoid wizards that are too small.

The caller can use wxSizer::SetMinSize to enlarge it beyond the minimal size. If `wxRESIZE_BORDER` was passed to constructor, user can resize wizard and consequently the page area (but not make it smaller than the minimal size).

It is recommended to add the first page to the page area sizer. For simple wizards, this will enlarge the wizard to fit the biggest page.

For non-linear wizards, the first page of every separate chain should be added. Caller-specified size can be accomplished using wxSizer::SetMinSize(). Adding pages to the page area sizer affects the default border width around page area that can be altered with SetBorder().

virtual wxSize wxWizard::GetPageSize ( ) const [virtual]

Returns the size available for the pages.

virtual bool wxWizard::HasNextPage (wxWizardPage * *page*) [virtual]

Return true if this page is not the last one in the wizard.

The base class version implements this by calling page->GetNext but this could be undesirable if, for example, the pages are created on demand only.

See also

HasPrevPage()

virtual bool wxWizard::HasPrevPage (wxWizardPage * *page*) [virtual]

Returns true if this page is not the last one in the wizard.

The base class version implements this by calling page->GetPrev but this could be undesirable if, for example, the pages are created on demand only.

See also

HasNextPage()

virtual bool wxWizard::RunWizard (wxWizardPage * *firstPage*) [virtual]

Executes the wizard starting from the given page, returning true if it was successfully finished or false if user cancelled it.

The *firstPage* cannot be NULL.

void wxWizard::SetBitmap (const wxBitmap & *bitmap*)

Sets the bitmap used for the wizard.

void wxWizard::SetBitmapBackgroundColour (const wxColour & *colour*)

Sets the colour that should be used to fill the area not taken up by the wizard or page bitmap, if a non-zero bitmap placement flag has been set.

See also SetBitmapPlacement().

void wxWizard::SetBitmapPlacement (int *placement*)

Sets the flags indicating how the wizard or page bitmap should be expanded and positioned to fit the page height.

By default, placement is 0 (no expansion is done). *placement* is a bitlist with the following possible values:

- **wxWIZARD_VALIGN_TOP:** Aligns the bitmap at the top.

- **wxWIZARD_VALIGN_CENTRE:** Centres the bitmap vertically.
- **wxWIZARD_VALIGN_BOTTOM:** Aligns the bitmap at the bottom.
- **wxWIZARD_HALIGN_LEFT:** Left-aligns the bitmap.
- **wxWIZARD_HALIGN_CENTRE:** Centres the bitmap horizontally.
- **wxWIZARD_HALIGN_RIGHT:** Right-aligns the bitmap.
- **wxWIZARD_TILE:** todo 52.

See also SetMinimumBitmapWidth().

virtual void wxWizard::SetBorder (int *border*) `[virtual]`

Sets width of border around page area.

Default is zero. For backward compatibility, if there are no pages in GetPageAreaSizer(), the default is 5 pixels.

If there is a five point border around all controls in a page and the border around page area is left as zero, a five point white space along all dialog borders will be added to the control border in order to space page controls ten points from the dialog border and non-page controls.

void wxWizard::SetMinimumBitmapWidth (int *width*)

Sets the minimum width for the bitmap that will be constructed to contain the actual wizard or page bitmap if a non-zero bitmap placement flag has been set.

If this is not set when using bitmap placement, the initial layout may be incorrect.

See also SetBitmapPlacement().

virtual void wxWizard::SetPageSize (const wxSize & *sizePage*) `[virtual]`

Sets the minimal size to be made available for the wizard pages.

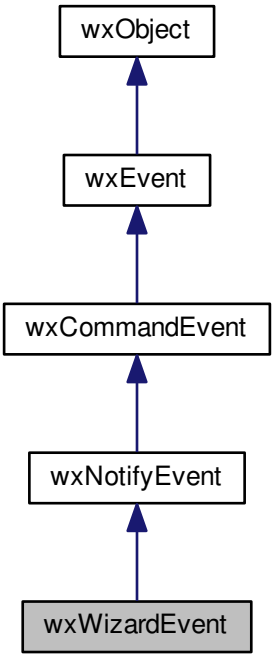
The wizard will take into account the size of the bitmap (if any) itself. Also, the wizard will never be smaller than the default size.

The recommended way to use this function is to lay out all wizard pages using the sizers (even though the wizard is not resizable) and then use wxSizer::CalcMin() in a loop to calculate the maximum of minimal sizes of the pages and pass it to SetPageSize().

21.864 wxWizardEvent Class Reference

`#include <wx/wizard.h>`

Inheritance diagram for wxWizardEvent:



21.864.1 Detailed Description

wxWizardEvent class represents an event generated by the wxWizard: this event is first sent to the page itself and, if not processed there, goes up the window hierarchy as usual.

Events using this class

The following event handler macros redirect the events to member function handlers '**func**' with prototypes like: void handlerFuncName(wxWizardEvent& event)

Event macros:

- EVT_WIZARD_PAGE_CHANGED(id, func): The page has been just changed (this event cannot be vetoed).
- EVT_WIZARD_PAGE_CHANGING(id, func): The page is being changed (this event can be vetoed).
- EVT_WIZARD_BEFORE_PAGE_CHANGED(id, func): Called after Next is clicked but before GetNext is called. Unlike EVT_WIZARD_CHANGING, the handler for this function can change state that might affect the return value of GetNext. This event can be vetoed.
- EVT_WIZARD_PAGE_SHOWN(id, func): The page was shown and laid out (this event cannot be vetoed).
- EVT_WIZARD_CANCEL(id, func): The user attempted to cancel the wizard (this event may also be vetoed).
- EVT_WIZARD_HELP(id, func): The wizard help button was pressed.
- EVT_WIZARD_FINISHED(id, func): The wizard finished button was pressed.

Library: wxAdvanced

Category: Events

See also

wxWizard, Wizard Sample

Public Member Functions

- wxWizardEvent (wxEventType type=wxEVT_NULL, int id=wxID_ANY, bool direction=true, wxWizardPage *page=0)

  Constructor.

- bool GetDirection () const

  Return the direction in which the page is changing: for `EVT_WIZARD_PAGE_CHANGING`, return true if we're going forward or false otherwise and for `EVT_WIZARD_PAGE_CHANGED` return true if we came from the previous page and false if we returned from the next one.

- wxWizardPage * GetPage () const

  Returns the wxWizardPage which was active when this event was generated.

Additional Inherited Members

21.864.2 Constructor & Destructor Documentation

wxWizardEvent::wxWizardEvent (wxEventType *type* = wxEVT_NULL, int *id* = wxID_ANY, bool *direction* = `true`, wxWizardPage * *page* = `0`)

Constructor.

It is not normally used by the user code as the objects of this type are constructed by wxWizard.

21.864.3 Member Function Documentation

bool wxWizardEvent::GetDirection ( ) const

Return the direction in which the page is changing: for `EVT_WIZARD_PAGE_CHANGING`, return true if we're going forward or false otherwise and for `EVT_WIZARD_PAGE_CHANGED` return true if we came from the previous page and false if we returned from the next one.

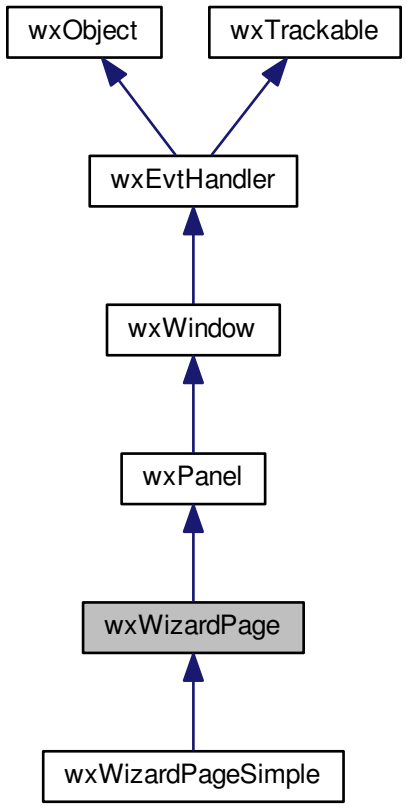
wxWizardPage* wxWizardEvent::GetPage ( ) const

Returns the wxWizardPage which was active when this event was generated.

21.865 wxWizardPage Class Reference

```
#include <wx/wizard.h>
```

Inheritance diagram for wxWizardPage:



21.865.1 Detailed Description

wxWizardPage is one of the screens in wxWizard: it must know what are the following and preceding pages (which may be NULL for the first/last page).

Except for this extra knowledge, wxWizardPage is just a panel, so the controls may be placed directly on it in the usual way.

This class allows the programmer to decide the order of pages in the wizard dynamically (during run-time) and so provides maximal flexibility. Usually, however, the order of pages is known in advance in which case wxWizard↩ PageSimple class is enough and it is simpler to use.

21.865.2 Virtual functions to override

To use this class, you must override wxWizardPage::GetPrev() and wxWizardPage::GetNext() pure virtual functions (or you may use wxWizardPageSimple instead). wxWizardPage::GetBitmap() can also be overridden, but this should be very rarely needed.

Library: wxAdvanced

Category: Miscellaneous Windows

See also

wxWizard, Wizard Sample

Public Member Functions

- wxWizardPage ()

  Default constructor.
- wxWizardPage (wxWizard *parent, const wxBitmap &bitmap=wxNullBitmap)

  Constructor accepts an optional bitmap which will be used for this page instead of the default one for this wizard (note that all bitmaps used should be of the same size).
- bool Create (wxWizard *parent, const wxBitmap &bitmap=wxNullBitmap)

  Creates the wizard page.
- virtual wxBitmap GetBitmap () const

  This method is called by wxWizard to get the bitmap to display alongside the page.
- virtual wxWizardPage * GetNext () const =0

  Get the page which should be shown when the user chooses the "Next" button: if NULL is returned, this button will be disabled.
- virtual wxWizardPage * GetPrev () const =0

  Get the page which should be shown when the user chooses the "Back" button: if NULL is returned, this button will be disabled.

Additional Inherited Members

21.865.3 Constructor & Destructor Documentation

wxWizardPage::wxWizardPage ( )

Default constructor.

wxWizardPage::wxWizardPage (wxWizard * *parent,* const wxBitmap & *bitmap* = wxNullBitmap)

Constructor accepts an optional bitmap which will be used for this page instead of the default one for this wizard (note that all bitmaps used should be of the same size).

Notice that no other parameters are needed because the wizard will resize and reposition the page anyhow.

Parameters

parent	The parent wizard
bitmap	The page-specific bitmap if different from the global one

21.865.4 Member Function Documentation

bool wxWizardPage::Create (wxWizard * *parent,* const wxBitmap & *bitmap* = wxNullBitmap)

Creates the wizard page.

Must be called if the default constructor had been used to create the object.

Parameters

parent	The parent wizard
bitmap	The page-specific bitmap if different from the global one

virtual wxBitmap wxWizardPage::GetBitmap ( ) const `[virtual]`

This method is called by wxWizard to get the bitmap to display alongside the page.

By default, `m_bitmap` member variable which was set in the wxWizardPage() constructor.

If the bitmap was not explicitly set (i.e. if wxNullBitmap is returned), the default bitmap for the wizard should be used.

The only cases when you would want to override this function is if the page bitmap depends dynamically on the user choices, i.e. almost never.

virtual wxWizardPage* wxWizardPage::GetNext ( ) const `[pure virtual]`

Get the page which should be shown when the user chooses the `"Next"` button: if NULL is returned, this button will be disabled.

The last page of the wizard will usually return NULL from here, but the others will not.

See also

GetPrev()

virtual wxWizardPage* wxWizardPage::GetPrev ( ) const `[pure virtual]`

Get the page which should be shown when the user chooses the `"Back"` button: if NULL is returned, this button will be disabled.

The first page of the wizard will usually return NULL from here, but the others will not.

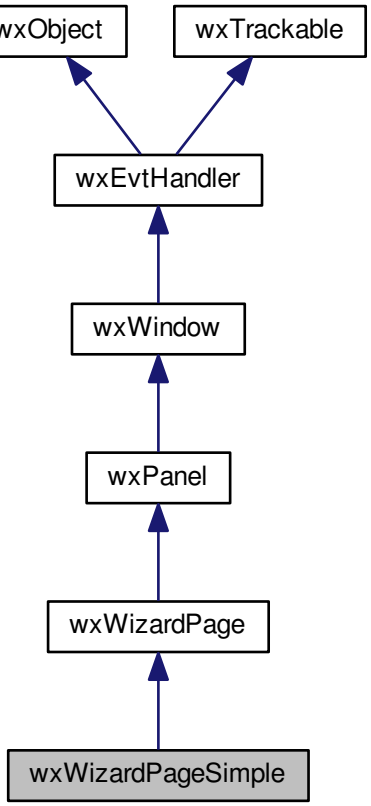
See also

GetNext()

21.866 wxWizardPageSimple Class Reference

`#include <wx/wizard.h>`

Inheritance diagram for wxWizardPageSimple:



21.866.1 Detailed Description

wxWizardPageSimple is the simplest possible wxWizardPage implementation: it just returns the pointers given to its constructor from wxWizardPage::GetNext() and wxWizardPage::GetPrev() functions.

This makes it very easy to use the objects of this class in the wizards where the pages order is known statically - on the other hand, if this is not the case you must derive your own class from wxWizardPage instead.

Library: wxAdvanced

Category: Miscellaneous Windows

See also

wxWizard, Wizard Sample

Public Member Functions

- wxWizardPageSimple ()

  Default constructor.
- wxWizardPageSimple (wxWizard *parent, wxWizardPage *prev=NULL, wxWizardPage *next=NULL, const wxBitmap &bitmap=wxNullBitmap)

  Constructor takes the previous and next pages.
- bool Create (wxWizard *parent=NULL, wxWizardPage *prev=NULL, wxWizardPage *next=NULL, const wxBitmap &bitmap=wxNullBitmap)

  Creates the wizard page.
- wxWizardPageSimple & Chain (wxWizardPageSimple *next)

  A helper chaining this page with the next one.
- void SetNext (wxWizardPage *next)

  Sets the next page.
- void SetPrev (wxWizardPage *prev)

  Sets the previous page.

Static Public Member Functions

- static void Chain (wxWizardPageSimple *first, wxWizardPageSimple *second)

  A convenience function to make the pages follow each other.

Additional Inherited Members

21.866.2 Constructor & Destructor Documentation

wxWizardPageSimple::wxWizardPageSimple ( )

Default constructor.

wxWizardPageSimple::wxWizardPageSimple (wxWizard * *parent*, wxWizardPage * *prev* = NULL, wxWizardPage * *next* = NULL, const wxBitmap & *bitmap* = wxNullBitmap)

Constructor takes the previous and next pages.

They may be modified later by SetPrev() or SetNext().

21.866.3 Member Function Documentation

wxWizardPageSimple& wxWizardPageSimple::Chain (wxWizardPageSimple * *next*)

A helper chaining this page with the next one.

Notice that this method returns a reference to the next page, so the calls to it can, in turn, be chained:

```
wxWizardPageSimple* firstPage = new FirstPage;
(*firstPage).Chain(new SecondPage)
            .Chain(new ThirdPage)
            .Chain(new LastPage);
```

This makes this method the simplest way to define the order of changes in fully static wizards, i.e. in those where the order doesn't depend on the choices made by the user in the wizard pages during run-time.

Parameters

next	A non-NULL pointer to the next page.

Returns

Reference to *next* on which Chain() can be called again.

Since

2.9.5

static void wxWizardPageSimple::Chain (wxWizardPageSimple * *first*, wxWizardPageSimple * *second*) [static]

A convenience function to make the pages follow each other.

Example:

```
wxRadioboxPage *page3 = new wxRadioboxPage(wizard);
wxValidationPage *page4 = new wxValidationPage(wizard);

wxWizardPageSimple::Chain(page3, page4);
```

bool wxWizardPageSimple::Create (wxWizard * *parent* = NULL, wxWizardPage * *prev* = NULL, wxWizardPage * *next* = NULL, const wxBitmap & *bitmap* = wxNullBitmap)

Creates the wizard page.

Must be called if the default constructor had been used to create the object.

void wxWizardPageSimple::SetNext (wxWizardPage * *next*)

Sets the next page.

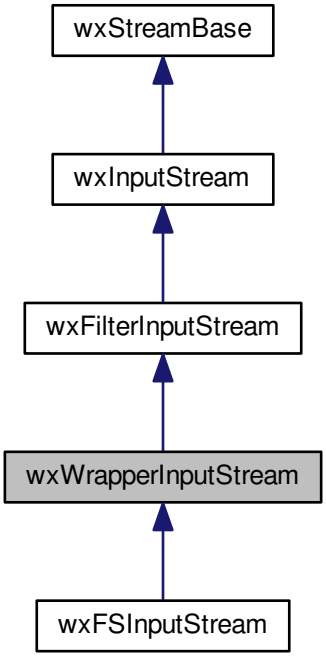
void wxWizardPageSimple::SetPrev (wxWizardPage * *prev*)

Sets the previous page.

21.867 wxWrapperInputStream Class Reference

`#include <wx/stream.h>`

Inheritance diagram for wxWrapperInputStream:



21.867.1 Detailed Description

A wrapper input stream is a kind of filter stream which forwards all the operations to its base stream.

This is useful to build utility classes such as wxFSInputStream.

Note

The interface of this class is the same as that of wxInputStream. Only a constructor differs and it is documented below.

Library: wxBase

Category: Streams

See also

wxFSInputStream, wxFilterInputStream

Since

2.9.4

Public Member Functions

- wxWrapperInputStream (wxInputStream &stream)

Initializes a wrapper stream.

- wxWrapperInputStream (wxInputStream *stream)

  Initializes a wrapper stream.

Protected Member Functions

- wxWrapperInputStream ()

  Default constructor, use InitParentStream() to finish initialization.

- void InitParentStream (wxInputStream &stream)

  Set up the wrapped stream for an object initialized using the default constructor.

- void InitParentStream (wxInputStream *stream)

  Set up the wrapped stream for an object initialized using the default constructor.

21.867.2 Constructor & Destructor Documentation

wxWrapperInputStream::wxWrapperInputStream (wxInputStream & *stream*)

Initializes a wrapper stream.

If the parent stream is passed as a pointer then the new wrapper stream takes ownership of it. If it is passed by reference then it does not.

wxWrapperInputStream::wxWrapperInputStream (wxInputStream * *stream*)

Initializes a wrapper stream.

If the parent stream is passed as a pointer then the new wrapper stream takes ownership of it. If it is passed by reference then it does not.

wxWrapperInputStream::wxWrapperInputStream ( ) [protected]

Default constructor, use InitParentStream() to finish initialization.

This constructor can be used by the derived classes from their own constructors when the parent stream can't be specified immediately. The derived class must call InitParentStream() later to do it.

21.867.3 Member Function Documentation

void wxWrapperInputStream::InitParentStream (wxInputStream & *stream*) [protected]

Set up the wrapped stream for an object initialized using the default constructor.

The ownership logic is the same as for the non-default constructor, i.e. this object takes ownership of the stream if it's passed by pointer but not if it's passed by reference.

void wxWrapperInputStream::InitParentStream (wxInputStream * *stream*) [protected]

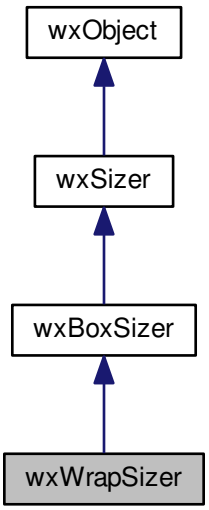
Set up the wrapped stream for an object initialized using the default constructor.

The ownership logic is the same as for the non-default constructor, i.e. this object takes ownership of the stream if it's passed by pointer but not if it's passed by reference.

21.868 wxWrapSizer Class Reference

`#include <wx/wrapsizer.h>`

Inheritance diagram for wxWrapSizer:



21.868.1 Detailed Description

A wrap sizer lays out its items in a single line, like a box sizer – as long as there is space available in that direction.

Once all available space in the primary direction has been used, a new line is added and items are added there.

So a wrap sizer has a primary orientation for adding items, and adds lines as needed in the secondary direction.

Library: wxCore

Category: Window Layout

See also

wxBoxSizer, wxSizer, Sizers Overview

Public Member Functions

- wxWrapSizer (int orient=wxHORIZONTAL, int flags=wxWRAPSIZER_DEFAULT_FLAGS)

  Constructor for a wxWrapSizer.
- virtual bool InformFirstDirection (int direction, int size, int availableOtherDir)

  Not used by an application.
- virtual void RecalcSizes ()

  Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

- virtual wxSize CalcMin ()

  Implements the calculation of a box sizer's minimal.

Protected Member Functions

- virtual bool IsSpaceItem (wxSizerItem *item) const

  Can be overridden in the derived classes to treat some normal items as spacers.

Additional Inherited Members

21.868.2 Constructor & Destructor Documentation

wxWrapSizer::wxWrapSizer (int *orient* = wxHORIZONTAL, int *flags* = wxWRAPSIZER_DEFAULT_FLAGS)

Constructor for a wxWrapSizer.

orient determines the primary direction of the sizer (the most common case being `wxHORIZONTAL`). The flags parameter can be a combination of the values `wxEXTEND_LAST_ON_EACH_LINE` which will cause the last item on each line to use any remaining space on that line and `wxREMOVE_LEADING_SPACES` which removes any spacer elements from the beginning of a row.

Both of these flags are on by default.

21.868.3 Member Function Documentation

virtual wxSize wxWrapSizer::CalcMin ( ) [virtual]

Implements the calculation of a box sizer's minimal.

It is used internally only and must not be called by the user. Documented for information.

Reimplemented from wxBoxSizer.

virtual bool wxWrapSizer::InformFirstDirection (int *direction*, int *size*, int *availableOtherDir*) [virtual]

Not used by an application.

This is the mechanism by which sizers can inform sub-items of the first determined size component. The sub-item can then better determine its size requirements.

Returns true if the information was used (and the sub-item min size was updated).

Reimplemented from wxSizer.

virtual bool wxWrapSizer::IsSpaceItem (wxSizerItem * *item*) const [protected], [virtual]

Can be overridden in the derived classes to treat some normal items as spacers.

This method is used to determine whether the given *item* should be considered to be a spacer for the purposes of `wxREMOVE_LEADING_SPACES` implementation. By default only returns true for the real spacers.

virtual void wxWrapSizer::RecalcSizes ( ) [virtual]

Implements the calculation of a box sizer's dimensions and then sets the size of its children (calling wxWindow::SetSize if the child is a window).

It is used internally only and must not be called by the user (call Layout() if you want to resize). Documented for information.

Reimplemented from wxBoxSizer.

21.869 wxXLocale Class Reference

`#include <wx/xlocale.h>`

21.869.1 Detailed Description

This class represents a locale object used by so-called xlocale API.

Unlike wxLocale it doesn't provide any non-trivial operations but simply provides a portable wrapper for POSIX `locale_t` type.

It exists solely to be provided as an argument to various `wxFoo_l()` functions which are the extensions of the standard locale-dependent functions (hence the name xlocale). These functions do exactly the same thing as the corresponding standard `foo()` except that instead of using the global program locale they use the provided wxXLocale object.

See Locale-dependent functions for a list of wxXLocale-enabled functions.

Conversely, if a program wanted to output the number in French locale, even if the current locale is different, it could use wxXLocale(wxLANGUAGE_FRENCH).

21.869.2 Availability

This class is fully implemented only under the platforms where xlocale POSIX API or equivalent is available. Currently the xlocale API is available under most of the recent Unix systems (including Linux, various BSD and Mac OS X) and Microsoft Visual C++ standard library provides a similar API starting from version 8 (Visual Studio 2005).

If neither POSIX API nor Microsoft proprietary equivalent are available, this class is still available but works in degraded mode: the only supported locale is the C one and attempts to create wxXLocale object for any other locale will fail. You can use the preprocessor macro `wxHAS_XLOCALE_SUPPORT` to test if full xlocale API is available or only skeleton C locale support is present.

Notice that wxXLocale is new in wxWidgets 2.9.0 and is not compiled in if `wxUSE_XLOCALE` was set to 0 during the library compilation.

Library: wxBase

Category: Application and System configuration

Predefined objects/pointers:

- wxNullXLocale

See also

wxLocale

Public Member Functions

- wxXLocale ()

  Creates an uninitialized locale object, IsOk() method will return false.

- wxXLocale (wxLanguage lang)

  Creates the locale object corresponding to the specified language.
- wxXLocale (const char *loc)

  Creates the locale object corresponding to the specified locale string.
- bool IsOk () const

  Returns true if this object is initialized, i.e. represents a valid locale or false otherwise.
- bool operator== (const wxXLocale &loc) const

  Comparison operator.

Static Public Member Functions

- static wxXLocale & GetCLocale ()

  Returns the global object representing the "C" locale.

21.869.3 Constructor & Destructor Documentation

wxXLocale::wxXLocale ( )

Creates an uninitialized locale object, IsOk() method will return false.

wxXLocale::wxXLocale (wxLanguage *lang*)

Creates the locale object corresponding to the specified language.

wxXLocale::wxXLocale (const char * *loc*)

Creates the locale object corresponding to the specified locale string.

The locale string is system-dependent, use constructor taking wxLanguage for better portability.

21.869.4 Member Function Documentation

static wxXLocale& wxXLocale::GetCLocale ( ) [static]

Returns the global object representing the "C" locale.

For an even shorter access to this object a global `wxCLocale` variable (implemented as a macro) is provided and can be used instead of calling this method.

bool wxXLocale::IsOk ( ) const

Returns true if this object is initialized, i.e. represents a valid locale or false otherwise.

bool wxXLocale::operator== (const wxXLocale & *loc*) const

Comparison operator.

21.870 wxXmlAttribute Class Reference

`#include <wx/xml/xml.h>`

21.870.1 Detailed Description

Represents a node attribute.

Example: in `<img src="hello.gif" id="3"/>`, `src` is an attribute with value `hello.gif` and `id` is an attribute with value `3`.

Library: wxXML

Category: XML

See also

wxXmlDocument, wxXmlNode

Public Member Functions

- wxXmlAttribute ()

  Default constructor.
- wxXmlAttribute (const wxString &name, const wxString &value, wxXmlAttribute *next=NULL)

  Creates the attribute with given name and value.
- virtual ~wxXmlAttribute ()

  The virtual destructor.
- wxString GetName () const

  Returns the name of this attribute.
- wxXmlAttribute * GetNext () const

  Returns the sibling of this attribute or NULL if there are no siblings.
- wxString GetValue () const

  Returns the value of this attribute.
- void SetName (const wxString &name)

  Sets the name of this attribute.
- void SetNext (wxXmlAttribute *next)

  Sets the sibling of this attribute.
- void SetValue (const wxString &value)

  Sets the value of this attribute.

21.870.2 Constructor & Destructor Documentation

wxXmlAttribute::wxXmlAttribute ( )

Default constructor.

wxXmlAttribute::wxXmlAttribute (const wxString & *name*, const wxString & *value*, wxXmlAttribute * *next* = `NULL`)

Creates the attribute with given *name* and *value*.

If *next* is not NULL, then sets it as sibling of this attribute.

virtual wxXmlAttribute::~wxXmlAttribute ( ) `[virtual]`

The virtual destructor.

21.870.3 Member Function Documentation

wxString wxXmlAttribute::GetName ( ) const

Returns the name of this attribute.

wxXmlAttribute* wxXmlAttribute::GetNext ( ) const

Returns the sibling of this attribute or NULL if there are no siblings.

wxString wxXmlAttribute::GetValue ( ) const

Returns the value of this attribute.

void wxXmlAttribute::SetName (const wxString & *name*)

Sets the name of this attribute.

void wxXmlAttribute::SetNext (wxXmlAttribute * *next*)

Sets the sibling of this attribute.

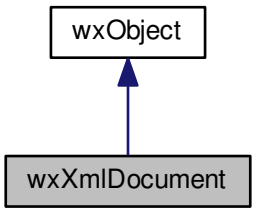
void wxXmlAttribute::SetValue (const wxString & *value*)

Sets the value of this attribute.

21.871 wxXmlDocument Class Reference

`#include <wx/xml/xml.h>`

Inheritance diagram for wxXmlDocument:



21.871.1 Detailed Description

This class holds XML data/document as parsed by XML parser in the root node.

wxXmlDocument internally uses the expat library which comes with wxWidgets to parse the given stream.

A simple example of using XML classes is:

```
wxXmlDocument doc;
if (!doc.Load("myfile.xml"))
    return false;

// start processing the XML file
if (doc.GetRoot()->GetName() != "myroot-node")
    return false;

// examine prologue
wxXmlNode *prolog = doc.GetDocumentNode()->GetChildren();
while (prolog) {

    if (prolog->GetType() == wxXML_PI_NODE && prolog->
      GetName() == "target") {

        // process Process Instruction contents
        wxString pi = prolog->GetContent();

        ...

    }
}

wxXmlNode *child = doc.GetRoot()->GetChildren();
while (child) {

    if (child->GetName() == "tag1") {

        // process text enclosed by tag1/tag1
        wxString content = child->GetNodeContent();

        ...

        // process attributes of tag1
        wxString attrvalue1 =
            child->GetAttribute("attr1", "default-value");
        wxString attrvalue2 =
            child->GetAttribute("attr2", "default-value");

        ...

    } else if (child->GetName() == "tag2") {

        // process tag2 ...
    }

    child = child->GetNext();
}
```

Note that if you want to preserve the original formatting of the loaded file including whitespaces and indentation, you need to turn off whitespace-only textnode removal and automatic indentation:

```
wxXmlDocument doc;
doc.Load("myfile.xml", "UTF-8", wxXMLDOC_KEEP_WHITESPACE_NODES);

// myfile2.xml will be identical to myfile.xml saving it this way:
doc.Save("myfile2.xml", wxXML_NO_INDENTATION);
```

Using default parameters, you will get a reformatted document which in general is different from the original loaded content:

```
wxXmlDocument doc;
doc.Load("myfile.xml");
doc.Save("myfile2.xml");  // myfile2.xml != myfile.xml
```

Library: wxXML

Category: XML

See also

wxXmlNode, wxXmlAttribute

Public Member Functions

- wxXmlDocument ()

  Default constructor.
- wxXmlDocument (const wxXmlDocument &doc)

  Copy constructor.
- wxXmlDocument (const wxString &filename, const wxString &encoding="UTF-8"))

  Loads the given filename using the given encoding.
- wxXmlDocument (wxInputStream &stream, const wxString &encoding="UTF-8")

  Loads the XML document from given stream using the given encoding.
- virtual ~wxXmlDocument ()

  Virtual destructor.
- void AppendToProlog (wxXmlNode *node)

  Appends a Process Instruction or Comment node to the document prologue.
- wxXmlNode * DetachDocumentNode ()

  Detaches the document node and returns it.
- wxXmlNode * DetachRoot ()

  Detaches the root entity node and returns it.
- wxString GetEncoding () const

  Returns encoding of in-memory representation of the document (same as passed to Load() or constructor, defaults to UTF-8).
- const wxString & GetFileEncoding () const

  Returns encoding of document (may be empty).
- wxXmlNode * GetDocumentNode () const

  Returns the document node of the document.
- wxXmlNode * GetRoot () const

  Returns the root element node of the document.
- const wxString & GetVersion () const

  Returns the version of document.
- bool IsOk () const

  Returns true if the document has been loaded successfully.
- virtual bool Load (const wxString &filename, const wxString &encoding="UTF-8", int flags=wxXMLDOC_NONE)

  Parses filename as an xml document and loads its data.
- virtual bool Load (wxInputStream &stream, const wxString &encoding="UTF-8", int flags=wxXMLDOC_NONE)

  Like Load(const wxString&, const wxString&, int) but takes the data from given input stream.
- virtual bool Save (const wxString &filename, int indentstep=2) const

  Saves XML tree creating a file named with given string.
- virtual bool Save (wxOutputStream &stream, int indentstep=2) const

  Saves XML tree in the given output stream.
- void SetDocumentNode (wxXmlNode *node)

  Sets the document node of this document.
- void SetEncoding (const wxString &enc)

  Sets the encoding of the document.
- void SetFileEncoding (const wxString &encoding)

  Sets the enconding of the file which will be used to save the document.
- void SetRoot (wxXmlNode *node)

  Sets the root element node of this document.
- void SetVersion (const wxString &version)

  Sets the version of the XML file which will be used to save the document.
- wxXmlDocument & operator= (const wxXmlDocument &doc)

  Deep copies the given document.

Static Public Member Functions

- static wxVersionInfo GetLibraryVersionInfo ()
  Get expat library version information.

Additional Inherited Members

21.871.2 Constructor & Destructor Documentation

wxXmlDocument::wxXmlDocument ( )

Default constructor.

wxXmlDocument::wxXmlDocument (const wxXmlDocument & *doc*)

Copy constructor.

Deep copies all the XML tree of the given document.

wxXmlDocument::wxXmlDocument (const wxString & *filename,* const wxString & *encoding* = "UTF-8")

Loads the given filename using the given encoding.

See Load().

wxXmlDocument::wxXmlDocument (wxInputStream & *stream,* const wxString & *encoding* = "UTF-8")

Loads the XML document from given stream using the given encoding.

See Load().

virtual wxXmlDocument::~wxXmlDocument ( ) [virtual]

Virtual destructor.

Frees the document root node.

21.871.3 Member Function Documentation

void wxXmlDocument::AppendToProlog (wxXmlNode * *node*)

Appends a Process Instruction or Comment node to the document prologue.

Calling this function will create a prologue or attach the node to the end of an existing prologue.

Since

2.9.2

wxXmlNode* wxXmlDocument::DetachDocumentNode ( )

Detaches the document node and returns it.

The document node will be set to NULL and thus IsOk() will return false after calling this function.

Note that the caller is responsible for deleting the returned node in order to avoid memory leaks.

Since

2.9.2

wxXmlNode* wxXmlDocument::DetachRoot ( )

Detaches the root entity node and returns it.

After calling this function, the document node will remain together with any prologue nodes, but IsOk() will return false since the root entity will be missing.

Note that the caller is responsible for deleting the returned node in order to avoid memory leaks.

wxXmlNode* wxXmlDocument::GetDocumentNode ( ) const

Returns the document node of the document.

Since

2.9.2

wxString wxXmlDocument::GetEncoding ( ) const

Returns encoding of in-memory representation of the document (same as passed to Load() or constructor, defaults to UTF-8).

Note

this is meaningless in Unicode build where data are stored as `wchar_t*`.

const wxString& wxXmlDocument::GetFileEncoding ( ) const

Returns encoding of document (may be empty).

Note

This is the encoding original file was saved in, **not** the encoding of in-memory representation!

static wxVersionInfo wxXmlDocument::GetLibraryVersionInfo ( ) [static]

Get expat library version information.

Since

2.9.2

See also

wxVersionInfo

wxXmlNode* wxXmlDocument::GetRoot ( ) const

Returns the root element node of the document.

const wxString& wxXmlDocument::GetVersion ( ) const

Returns the version of document.

This is the value in the <?xml version="1.0"?> header of the XML document. If the version attribute was not explicitly given in the header, this function returns an empty string.

bool wxXmlDocument::IsOk ( ) const

Returns true if the document has been loaded successfully.

virtual bool wxXmlDocument::Load (const wxString & *filename*, const wxString & *encoding* = "UTF-8", int *flags* = wxXMLDOC_NONE) [virtual]

Parses *filename* as an xml document and loads its data.

If *flags* does not contain wxXMLDOC_KEEP_WHITESPACE_NODES, then, while loading, all nodes of type wxXML_TEXT_NODE (see wxXmlNode) are automatically skipped if they contain whitespaces only.

The removal of these nodes makes the load process slightly faster and requires less memory however makes impossible to recreate exactly the loaded text with a Save() call later. Read the initial description of this class for more info.

Returns true on success, false otherwise.

virtual bool wxXmlDocument::Load (wxInputStream & *stream*, const wxString & *encoding* = "UTF-8", int *flags* = wxXMLDOC_NONE) [virtual]

Like Load(const wxString&, const wxString&, int) but takes the data from given input stream.

wxXmlDocument& wxXmlDocument::operator= (const wxXmlDocument & *doc*)

Deep copies the given document.

virtual bool wxXmlDocument::Save (const wxString & *filename*, int *indentstep* = 2) const [virtual]

Saves XML tree creating a file named with given string.

If *indentstep* is greater than or equal to zero, then, while saving, an automatic indentation is added with steps composed by indentstep spaces.

If *indentstep* is wxXML_NO_INDENTATION, then, automatic indentation is turned off.

virtual bool wxXmlDocument::Save (wxOutputStream & *stream*, int *indentstep* = 2) const [virtual]

Saves XML tree in the given output stream.

See Save(const wxString&, int) for a description of *indentstep*.

void wxXmlDocument::SetDocumentNode (wxXmlNode * *node*)

Sets the document node of this document.

Deletes any previous document node. Use DetachDocumentNode() and then SetDocumentNode() if you want to replace the document node without deleting the old document tree.

Since

2.9.2

void wxXmlDocument::SetEncoding (const wxString & *enc*)

Sets the encoding of the document.

void wxXmlDocument::SetFileEncoding (const wxString & *encoding*)

Sets the enconding of the file which will be used to save the document.

void wxXmlDocument::SetRoot (wxXmlNode * *node*)

Sets the root element node of this document.

Will create the document node if necessary. Any previous root element node is deleted.

void wxXmlDocument::SetVersion (const wxString & *version*)

Sets the version of the XML file which will be used to save the document.

21.872 wxXmlNode Class Reference

`#include <wx/xml/xml.h>`

21.872.1 Detailed Description

Represents a node in an XML document.

See wxXmlDocument.

Node has a name and may have content and attributes.

Most common node types are `wxXML_TEXT_NODE` (name and attributes are irrelevant) and `wxXML_ELEMENT_NODE`.

Example: in `<title>hi</title>` there is an element with the name `title` and irrelevant content and one child of type `wxXML_TEXT_NODE` with `hi` as content.

The `wxXML_PI_NODE` type sets the name to the PI target and the contents to the instructions. Note that whilst the PI instructions are often in the form of pseudo-attributes these do not use the nodes attribute system. It is the users responsibility to code and decode the instruction text.

If `wxUSE_UNICODE` is 0, all strings are encoded in the encoding given to wxXmlDocument::Load (default is UTF-8).

Library: wxXML

Category: XML

See also

wxXmlDocument, wxXmlAttribute

Public Member Functions

- wxXmlNode (wxXmlNode *parent, wxXmlNodeType type, const wxString &name, const wxString &content=wxEmptyString, wxXmlAttribute *attrs=NULL, wxXmlNode *next=NULL, int lineNo=-1)

  Creates this XML node and eventually insert it into an existing XML tree.
- wxXmlNode (wxXmlNodeType type, const wxString &name, const wxString &content=wxEmptyString, int lineNo=-1)

  A simplified version of the first constructor form, assuming a NULL parent.
- wxXmlNode (const wxXmlNode &node)

  Copy constructor.
- virtual ~wxXmlNode ()

  The virtual destructor.
- virtual void AddAttribute (const wxString &name, const wxString &value)

  Appends a attribute with given name and value to the list of attributes for this node.
- virtual void AddAttribute (wxXmlAttribute *attr)

  Appends given attribute to the list of attributes for this node.
- virtual void AddChild (wxXmlNode *child)

  Adds node child as the last child of this node.
- virtual bool DeleteAttribute (const wxString &name)

  Removes the first attributes which has the given name from the list of attributes for this node.
- bool GetAttribute (const wxString &attrName, wxString *value) const

  Returns true if a attribute named attrName could be found.
- wxString GetAttribute (const wxString &attrName, const wxString &defaultVal=wxEmptyString) const

  Returns the value of the attribute named attrName if it does exist.
- wxXmlAttribute * GetAttributes () const

  Return a pointer to the first attribute of this node.
- wxXmlNode * GetChildren () const

  Returns the first child of this node.
- const wxString & GetContent () const

  Returns the content of this node.
- int GetDepth (wxXmlNode *grandparent=NULL) const

  Returns the number of nodes which separate this node from `grandparent`.
- bool GetNoConversion () const

  Returns a flag indicating whether encoding conversion is necessary when saving.
- int GetLineNumber () const

  Returns line number of the node in the input XML file or `-1` if it is unknown.
- const wxString & GetName () const

  Returns the name of this node.
- wxXmlNode * GetNext () const

  Returns a pointer to the sibling of this node or NULL if there are no siblings.
- wxString GetNodeContent () const

  Returns the content of the first child node of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`.
- wxXmlNode * GetParent () const

  Returns a pointer to the parent of this node or NULL if this node has no parent.
- wxXmlNodeType GetType () const

  Returns the type of this node.
- bool HasAttribute (const wxString &attrName) const

  Returns true if this node has a attribute named attrName.
- virtual bool InsertChild (wxXmlNode *child, wxXmlNode *followingNode)

  Inserts the child node immediately before followingNode in the children list.
- virtual bool InsertChildAfter (wxXmlNode *child, wxXmlNode *precedingNode)

Inserts the child node immediately after precedingNode in the children list.

- bool IsWhitespaceOnly () const

  Returns true if the content of this node is a string containing only whitespaces (spaces, tabs, new lines, etc).

- virtual bool RemoveChild (wxXmlNode *child)

  Removes the given node from the children list.

- void SetAttributes (wxXmlAttribute *attr)

  Sets as first attribute the given wxXmlAttribute object.

- void SetChildren (wxXmlNode *child)

  Sets as first child the given node.

- void SetContent (const wxString &con)

  Sets the content of this node.

- void SetName (const wxString &name)

  Sets the name of this node.

- void SetNext (wxXmlNode *next)

  Sets as sibling the given node.

- void SetNoConversion (bool noconversion)

  Sets a flag to indicate whether encoding conversion is necessary when saving.

- void SetParent (wxXmlNode *parent)

  Sets as parent the given node.

- void SetType (wxXmlNodeType type)

  Sets the type of this node.

- wxXmlNode & operator= (const wxXmlNode &node)

  See the copy constructor for more info.

21.872.2 Constructor & Destructor Documentation

wxXmlNode::wxXmlNode (wxXmlNode * *parent*, wxXmlNodeType *type*, const wxString & *name*, const wxString & *content* = wxEmptyString, wxXmlAttribute * *attrs* = NULL, wxXmlNode * *next* = NULL, int *lineNo* = -1)

Creates this XML node and eventually insert it into an existing XML tree.

Parameters

parent	The parent node to which append this node instance. If this argument is NULL this new node will be floating and it can be appended later to another one using the AddChild() or InsertChild() functions. Otherwise the child is already added to the XML tree by this constructor and it shouldn't be done again.
type	One of the wxXmlNodeType enumeration value.
name	The name of the node. This is the string which appears between angular brackets.
content	The content of the node. Only meaningful when type is wxXML_TEXT_NODE or wxXML_CDATA_SECTION_NODE.
attrs	If not NULL, this wxXmlAttribute object and its eventual siblings are attached to the node.
next	If not NULL, this node and its eventual siblings are attached to the node.
lineNo	Number of line this node was present at in input file or -1.

wxXmlNode::wxXmlNode (wxXmlNodeType *type*, const wxString & *name*, const wxString & *content* = wxEmptyString, int *lineNo* = -1)

A simplified version of the first constructor form, assuming a NULL parent.

Parameters

type	One of the wxXmlNodeType enumeration value.
name	The name of the node. This is the string which appears between angular brackets.
content	The content of the node. Only meaningful when type is `wxXML_TEXT_NODE` or `wxXML_↩ CDATA_SECTION_NODE`.
lineNo	Number of line this node was present at in input file or -1.

wxXmlNode::wxXmlNode (const wxXmlNode & *node*)

Copy constructor.

Note that this does NOT copy siblings and parent pointer, i.e. GetParent() and GetNext() will return NULL after using copy ctor and are never unmodified by operator=(). On the other hand, it DOES copy children and attributes.

virtual wxXmlNode::~wxXmlNode ( ) [virtual]

The virtual destructor.

Deletes attached children and attributes.

21.872.3 Member Function Documentation

virtual void wxXmlNode::AddAttribute (const wxString & *name,* const wxString & *value*) [virtual]

Appends a attribute with given *name* and *value* to the list of attributes for this node.

virtual void wxXmlNode::AddAttribute (wxXmlAttribute * *attr*) [virtual]

Appends given attribute to the list of attributes for this node.

virtual void wxXmlNode::AddChild (wxXmlNode * *child*) [virtual]

Adds node *child* as the last child of this node.

Note

Note that this function works in O(n) time where *n* is the number of existing children. Consequently, adding large number of child nodes using this method can be expensive, because it has O(n^2) time complexity in number of nodes to be added. Use InsertChildAfter() to populate XML tree in linear time.

See also

InsertChild(), InsertChildAfter()

virtual bool wxXmlNode::DeleteAttribute (const wxString & *name*) [virtual]

Removes the first attributes which has the given *name* from the list of attributes for this node.

bool wxXmlNode::GetAttribute (const wxString & *attrName,* wxString * *value*) const

Returns true if a attribute named attrName could be found.

The value of that attribute is saved in value (which must not be NULL).

wxString wxXmlNode::GetAttribute (const wxString & *attrName,* const wxString & *defaultVal* = wxEmptyString) const

Returns the value of the attribute named *attrName* if it does exist.

If it does not exist, the *defaultVal* is returned.

wxXmlAttribute* wxXmlNode::GetAttributes ( ) const

Return a pointer to the first attribute of this node.

wxXmlNode* wxXmlNode::GetChildren ( ) const

Returns the first child of this node.

To get a pointer to the second child of this node (if it does exist), use the GetNext() function on the returned value.

const wxString& wxXmlNode::GetContent ( ) const

Returns the content of this node.

Can be an empty string. Be aware that for nodes of type `wxXML_ELEMENT_NODE` (the most used node type) the content is an empty string. See GetNodeContent() for more details.

int wxXmlNode::GetDepth (wxXmlNode * *grandparent* = `NULL`) const

Returns the number of nodes which separate this node from `grandparent`.

This function searches only the parents of this node until it finds *grandparent* or the NULL node (which is the parent of non-linked nodes or the parent of a wxXmlDocument's root element node).

int wxXmlNode::GetLineNumber ( ) const

Returns line number of the node in the input XML file or `-1` if it is unknown.

const wxString& wxXmlNode::GetName ( ) const

Returns the name of this node.

Can be an empty string (e.g. for nodes of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`).

wxXmlNode* wxXmlNode::GetNext ( ) const

Returns a pointer to the sibling of this node or NULL if there are no siblings.

bool wxXmlNode::GetNoConversion ( ) const

Returns a flag indicating whether encoding conversion is necessary when saving.

The default is false.

You can improve saving efficiency considerably by setting this value.

wxString wxXmlNode::GetNodeContent ( ) const

Returns the content of the first child node of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`.

This function is very useful since the XML snippet "`tagnametagcontent/tagname`" is represented by expat with the following tag tree:

```
wxXML_ELEMENT_NODE name="tagname", content=""
|-- wxXML_TEXT_NODE name="", content="tagcontent"
```

or eventually:

```
wxXML_ELEMENT_NODE name="tagname", content=""
|-- wxXML_CDATA_SECTION_NODE name="", content="tagcontent"
```

An empty string is returned if the node has no children of type `wxXML_TEXT_NODE` or `wxXML_CDATA_SECTION_NODE`, or if the content of the first child of such types is empty.

wxXmlNode* wxXmlNode::GetParent ( ) const

Returns a pointer to the parent of this node or NULL if this node has no parent.

wxXmlNodeType wxXmlNode::GetType ( ) const

Returns the type of this node.

bool wxXmlNode::HasAttribute (const wxString & *attrName*) const

Returns true if this node has a attribute named *attrName*.

virtual bool wxXmlNode::InsertChild (wxXmlNode * *child*, wxXmlNode * *followingNode*) [virtual]

Inserts the *child* node immediately before *followingNode* in the children list.

Returns

true if *followingNode* has been found and the *child* node has been inserted.

Note

For historical reasons, *followingNode* may be NULL. In that case, then *child* is prepended to the list of children and becomes the first child of this node, i.e. it behaves identically to using the first children (as returned by GetChildren()) for *followingNode*).

See also

AddChild(), InsertChildAfter()

virtual bool wxXmlNode::InsertChildAfter (wxXmlNode * *child*, wxXmlNode * *precedingNode*) [virtual]

Inserts the *child* node immediately after *precedingNode* in the children list.

Returns

true if *precedingNode* has been found and the *child* node has been inserted.

Parameters

child	The child to insert.
precedingNode	The node to insert *child* after. As a special case, this can be NULL if this node has no children yet – in that case, *child* will become this node's only child node.

Since

2.8.8

See also

InsertChild(), AddChild()

bool wxXmlNode::IsWhitespaceOnly ( ) const

Returns true if the content of this node is a string containing only whitespaces (spaces, tabs, new lines, etc).

Note that this function is locale-independent since the parsing of XML documents must always produce the exact same tree regardless of the locale it runs under.

wxXmlNode& wxXmlNode::operator= (const wxXmlNode & *node*)

See the copy constructor for more info.

virtual bool wxXmlNode::RemoveChild (wxXmlNode * *child*) [virtual]

Removes the given node from the children list.

Returns true if the node was found and removed or false if the node could not be found. Note that the caller is responsible for deleting the removed node in order to avoid memory leaks.

void wxXmlNode::SetAttributes (wxXmlAttribute * *attr*)

Sets as first attribute the given wxXmlAttribute object.

The caller is responsible for deleting any previously present attributes attached to this node.

void wxXmlNode::SetChildren (wxXmlNode * *child*)

Sets as first child the given node.

The caller is responsible for deleting any previously present children node.

void wxXmlNode::SetContent (const wxString & *con*)

Sets the content of this node.

void wxXmlNode::SetName (const wxString & *name*)

Sets the name of this node.

void wxXmlNode::SetNext (wxXmlNode * *next*)

Sets as sibling the given node.

The caller is responsible for deleting any previously present sibling node.

void wxXmlNode::SetNoConversion (bool *noconversion*)

Sets a flag to indicate whether encoding conversion is necessary when saving.

The default is false.

You can improve saving efficiency considerably by setting this value.

void wxXmlNode::SetParent (wxXmlNode * *parent*)

Sets as parent the given node.

The caller is responsible for deleting any previously present parent node.

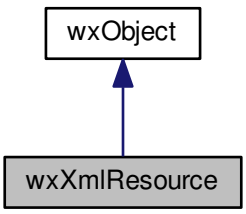
void wxXmlNode::SetType (wxXmlNodeType *type*)

Sets the type of this node.

21.873 wxXmlResource Class Reference

`#include <wx/xrc/xmlres.h>`

Inheritance diagram for wxXmlResource:



21.873.1 Detailed Description

This is the main class for interacting with the XML-based resource system.

The class holds XML resources from one or more .xml files, binary files or zip archive files.

Note that this is a singleton class and you'll never allocate/deallocate it. Just use the static wxXmlResource::Get() getter.

See also

XML Based Resource System (XRC), XRC File Format

Library: wxXRC

Category: XML Based Resource System (XRC)

Public Member Functions

- wxXmlResource (const wxString &filemask, int flags=wxXRC_USE_LOCALE, const wxString &domain=wxEmptyString)

  Constructor.
- wxXmlResource (int flags=wxXRC_USE_LOCALE, const wxString &domain=wxEmptyString)

  Constructor.
- virtual ~wxXmlResource ()

  Destructor.
- void AddHandler (wxXmlResourceHandler *handler)

  Initializes only a specific handler (or custom handler).
- void InsertHandler (wxXmlResourceHandler *handler)

  Add a new handler at the begining of the handler list.
- bool AttachUnknownControl (const wxString &name, wxWindow *control, wxWindow *parent=NULL)

  Attaches an unknown control to the given panel/window/dialog.
- void ClearHandlers ()

  Removes all handlers and deletes them (this means that any handlers added using AddHandler() must be allocated on the heap).
- int CompareVersion (int major, int minor, int release, int revision) const

  Compares the XRC version to the argument.
- const wxString & GetDomain () const

  Returns the domain (message catalog) that will be used to load translatable strings in the XRC.
- int GetFlags () const

  Returns flags, which may be a bitlist of wxXmlResourceFlags enumeration values.
- const wxXmlNode * GetResourceNode (const wxString &name) const

  Returns the wxXmlNode containing the definition of the object with the given name or NULL.
- long GetVersion () const

  *Returns version information (a.b.c.d = d + 256*c + 2562*b + 2563*a).*
- void InitAllHandlers ()

  Initializes handlers for all supported controls/windows.
- bool Load (const wxString &filemask)

  Loads resources from XML files that match given filemask.
- bool LoadFile (const wxFileName &file)

  Simpler form of Load() for loading a single XRC file.
- bool LoadAllFiles (const wxString &dirname)

  Loads all .xrc files from directory dirname.
- wxBitmap LoadBitmap (const wxString &name)

  Loads a bitmap resource from a file.
- wxDialog * LoadDialog (wxWindow *parent, const wxString &name)

  Loads a dialog.
- bool LoadDialog (wxDialog *dlg, wxWindow *parent, const wxString &name)

  Loads a dialog.
- wxFrame * LoadFrame (wxWindow *parent, const wxString &name)

  Loads a frame from the resource.
- bool LoadFrame (wxFrame *frame, wxWindow *parent, const wxString &name)

  Loads the contents of a frame onto an existing wxFrame.

- wxIcon LoadIcon (const wxString &name)

  Loads an icon resource from a file.
- wxMenu * LoadMenu (const wxString &name)

  Loads menu from resource.
- wxPanel * LoadPanel (wxWindow *parent, const wxString &name)

  Loads a panel.
- bool LoadPanel (wxPanel *panel, wxWindow *parent, const wxString &name)

  Loads a panel.
- wxToolBar * LoadToolBar (wxWindow *parent, const wxString &name)

  Loads a toolbar.
- void SetDomain (const wxString &domain)

  Sets the domain (message catalog) that will be used to load translatable strings in the XRC.
- void SetFlags (int flags)

  Sets flags (bitlist of wxXmlResourceFlags enumeration values).
- bool Unload (const wxString &filename)

  This function unloads a resource previously loaded by Load().

- wxMenuBar * LoadMenuBar (wxWindow *parent, const wxString &name)

  Loads a menubar from resource.
- wxMenuBar * LoadMenuBar (const wxString &name)

  Loads a menubar from resource.

- wxObject * LoadObject (wxWindow *parent, const wxString &name, const wxString &classname)

  Load an object from the resource specifying both the resource name and the class name.
- bool LoadObject (wxObject *instance, wxWindow *parent, const wxString &name, const wxString &classname)

  Load an object from the resource specifying both the resource name and the class name.

- wxObject * LoadObjectRecursively (wxWindow *parent, const wxString &name, const wxString &classname)

  Load an object from anywhere in the resource tree.
- bool LoadObjectRecursively (wxObject *instance, wxWindow *parent, const wxString &name, const wxString &classname)

  Load an object from anywhere in the resource tree.

Static Public Member Functions

- static void AddSubclassFactory (wxXmlSubclassFactory *factory)

  Registers subclasses factory for use in XRC.
- static wxString FindXRCIDById (int numId)

  Returns a string ID corresponding to the given numeric ID.
- static wxXmlResource * Get ()

  Gets the global resources object or creates one if none exists.
- static int GetXRCID (const wxString &str_id, int value_if_not_found=wxID_NONE)

  Returns a numeric ID that is equivalent to the string ID used in an XML resource.
- static wxXmlResource * Set (wxXmlResource *res)

  Sets the global resources object and returns a pointer to the previous one (may be NULL).

Protected Member Functions

- void ReportError (const wxXmlNode *context, const wxString &message)

  Reports error in XRC resources to the user.

- virtual void DoReportError (const wxString &xrcFile, const wxXmlNode *position, const wxString &message)

  Implementation of XRC resources errors reporting.

Additional Inherited Members

21.873.2 Constructor & Destructor Documentation

wxXmlResource::wxXmlResource (const wxString & *filemask*, int *flags* = wxXRC_USE_LOCALE, const wxString & *domain* = wxEmptyString)

Constructor.

Parameters

filemask	The XRC file, archive file, or wildcard specification that will be used to load all resource files inside a zip archive.
flags	One or more value of the wxXmlResourceFlags enumeration.
domain	The name of the gettext catalog to search for translatable strings. By default all loaded catalogs will be searched. This provides a way to allow the strings to only come from a specific catalog.

wxXmlResource::wxXmlResource (int *flags* = wxXRC_USE_LOCALE, const wxString & *domain* = wxEmptyString)

Constructor.

Parameters

flags	One or more value of the wxXmlResourceFlags enumeration.
domain	The name of the gettext catalog to search for translatable strings. By default all loaded catalogs will be searched. This provides a way to allow the strings to only come from a specific catalog.

virtual wxXmlResource::~wxXmlResource ( ) [virtual]

Destructor.

21.873.3 Member Function Documentation

void wxXmlResource::AddHandler (wxXmlResourceHandler * *handler*)

Initializes only a specific handler (or custom handler).

Convention says that the handler name is equal to the control's name plus 'XmlHandler', for example wxTextCtrl↩ XmlHandler, wxHtmlWindowXmlHandler.

The XML resource compiler (wxxrc) can create include file that contains initialization code for all controls used within the resource. Note that this handler must be allocated on the heap, since it will be deleted by ClearHandlers() later.

static void wxXmlResource::AddSubclassFactory (wxXmlSubclassFactory * *factory*) [static]

Registers subclasses factory for use in XRC.

This is useful only for language bindings developers who need a way to implement subclassing in wxWidgets ports that don't support wxRTTI (e.g. wxPython).

bool wxXmlResource::AttachUnknownControl (const wxString & *name*, wxWindow * *control*, wxWindow * *parent* = NULL)

Attaches an unknown control to the given panel/window/dialog.

Unknown controls are used in conjunction with <object class="unknown">.

void wxXmlResource::ClearHandlers ( )

Removes all handlers and deletes them (this means that any handlers added using AddHandler() must be allocated on the heap).

int wxXmlResource::CompareVersion (int *major*, int *minor*, int *release*, int *revision*) const

Compares the XRC version to the argument.

Returns -1 if the XRC version is less than the argument, +1 if greater, and 0 if they are equal.

virtual void wxXmlResource::DoReportError (const wxString & *xrcFile*, const wxXmlNode * *position*, const wxString & *message*) [protected], [virtual]

Implementation of XRC resources errors reporting.

This method is called by ReportError() and shouldn't be called directly; use ReportError() or wxXmlResourceHandler::ReportError() to log errors.

Default implementation uses wxLogError().

Parameters

xrcFile	File the error occurred in or empty string if it couldn't be determined.
position	XML node where the error occurred or NULL if it couldn't be determined.
message	Text of the error message. See ReportError() documentation for details of the string's format.

Note

You may override this method in a derived class to customize errors reporting. If you do so, you'll need to either use the derived class in all your code or call wxXmlResource::Set() to change the global wxXmlResource instance to your class.

Since

2.9.0

See also

ReportError()

static wxString wxXmlResource::FindXRCIDById (int *numId*) `[static]`

Returns a string ID corresponding to the given numeric ID.

The string returned is such that calling GetXRCID() with it as parameter yields *numId*. If there is no string identifier corresponding to the given numeric one, an empty string is returned.

Notice that, unlike GetXRCID(), this function is slow as it checks all of the identifiers used in XRC.

Since

2.9.0

static wxXmlResource* wxXmlResource::Get ( ) `[static]`

Gets the global resources object or creates one if none exists.

const wxString& wxXmlResource::GetDomain ( ) const

Returns the domain (message catalog) that will be used to load translatable strings in the XRC.

int wxXmlResource::GetFlags ( ) const

Returns flags, which may be a bitlist of wxXmlResourceFlags enumeration values.

const wxXmlNode* wxXmlResource::GetResourceNode (const wxString & *name*) const

Returns the wxXmlNode containing the definition of the object with the given name or NULL.

This function recursively searches all the loaded XRC files for an object with the specified *name*. If the object is found, the wxXmlNode corresponding to it is returned, so this function can be used to access additional information defined in the XRC file and not used by wxXmlResource itself, e.g. contents of application-specific XML tags.

Parameters

name	The name of the resource which must be unique for this function to work correctly, if there is more than one resource with the given name the choice of the one returned by this function is undefined.

Returns

The node corresponding to the resource with the given name or NULL.

long wxXmlResource::GetVersion ( ) const

Returns version information (a.b.c.d = d + 256*c + 2562*b + 2563*a).

static int wxXmlResource::GetXRCID (const wxString & *str_id,* int *value_if_not_found* = wxID_NONE) `[static]`

Returns a numeric ID that is equivalent to the string ID used in an XML resource.

If an unknown *str_id* is requested (i.e. other than wxID_XXX or integer), a new record is created which associates the given string with a number.

If *value_if_not_found* is `wxID_NONE`, the number is obtained via wxNewId(). Otherwise *value_if_not_found* is used.

Macro `XRCID(name)` is provided for convenient use in event tables.

Note

IDs returned by XRCID() cannot be used with the `EVT_*_RANGE` macros, because the order in which they are assigned to symbolic *name* values is not guaranteed.

void wxXmlResource::InitAllHandlers ( )

Initializes handlers for all supported controls/windows.

This will make the executable quite big because it forces linking against most of the wxWidgets library.

void wxXmlResource::InsertHandler (wxXmlResourceHandler * *handler*)

Add a new handler at the begining of the handler list.

bool wxXmlResource::Load (const wxString & *filemask*)

Loads resources from XML files that match given filemask.

Example:

```
if (!wxXmlResource::Get()->Load("rc/*.xrc"))
    wxLogError("Couldn't load resources!");
```

Note

If wxUSE_FILESYS is enabled, this method understands wxFileSystem URLs (see wxFileSystem::FindFirst()). If you are sure that the argument is name of single XRC file (rather than an URL or a wildcard), use LoadFile() instead.

See also

LoadFile(), LoadAllFiles()

bool wxXmlResource::LoadAllFiles (const wxString & *dirname*)

Loads all .xrc files from directory *dirname*.

Tries to load as many files as possible; if there's an error while loading one file, it still attempts to load other files.

Since

2.9.0

See also

LoadFile(), Load()

wxBitmap wxXmlResource::LoadBitmap (const wxString & *name*)

Loads a bitmap resource from a file.

wxDialog* wxXmlResource::LoadDialog (wxWindow * *parent,* const wxString & *name*)

Loads a dialog.

parent points to parent window (if any).

bool wxXmlResource::LoadDialog (wxDialog * *dlg*, wxWindow * *parent*, const wxString & *name*)

Loads a dialog.

parent points to parent window (if any).

This form is used to finish creation of an already existing instance (the main reason for this is that you may want to use derived class with a new event table). Example:

```
MyDialog dlg;
wxXmlResource::Get()->LoadDialog(&dlg, mainFrame, "my_dialog");
dlg.ShowModal();
```

bool wxXmlResource::LoadFile (const wxFileName & *file*)

Simpler form of Load() for loading a single XRC file.

Since

2.9.0

See also

Load(), LoadAllFiles()

wxFrame* wxXmlResource::LoadFrame (wxWindow * *parent*, const wxString & *name*)

Loads a frame from the resource.

parent points to parent window (if any).

bool wxXmlResource::LoadFrame (wxFrame * *frame*, wxWindow * *parent*, const wxString & *name*)

Loads the contents of a frame onto an existing wxFrame.

This form is used to finish creation of an already existing instance (the main reason for this is that you may want to use derived class with a new event table).

wxIcon wxXmlResource::LoadIcon (const wxString & *name*)

Loads an icon resource from a file.

wxMenu* wxXmlResource::LoadMenu (const wxString & *name*)

Loads menu from resource.

Returns NULL on failure.

wxMenuBar* wxXmlResource::LoadMenuBar (wxWindow * *parent*, const wxString & *name*)

Loads a menubar from resource.

Returns NULL on failure.

wxMenuBar* **wxXmlResource::LoadMenuBar** (**const wxString &** *name*)

Loads a menubar from resource.

Returns NULL on failure.

wxObject* **wxXmlResource::LoadObject** (**wxWindow** * *parent,* **const wxString &** *name,* **const wxString &** *classname*)

Load an object from the resource specifying both the resource name and the class name.

The first overload lets you load nonstandard container windows and returns NULL on failure. The second one lets you finish the creation of an existing instance and returns false on failure.

In either case, only the resources defined at the top level of XRC files can be loaded by this function, use LoadObjectRecursively() if you need to load an object defined deeper in the hierarchy.

bool wxXmlResource::LoadObject (**wxObject** * *instance,* **wxWindow** * *parent,* **const wxString &** *name,* **const wxString &** *classname*)

Load an object from the resource specifying both the resource name and the class name.

The first overload lets you load nonstandard container windows and returns NULL on failure. The second one lets you finish the creation of an existing instance and returns false on failure.

In either case, only the resources defined at the top level of XRC files can be loaded by this function, use LoadObjectRecursively() if you need to load an object defined deeper in the hierarchy.

wxObject* **wxXmlResource::LoadObjectRecursively** (**wxWindow** * *parent,* **const wxString &** *name,* **const wxString &** *classname*)

Load an object from anywhere in the resource tree.

These methods are similar to LoadObject() but may be used to load an object from anywhere in the resource tree and not only the top level. Note that you will very rarely need to do this as in normal use the entire container window (defined at the top level) is loaded and not its individual children but this method can be useful in some particular situations.

Since

2.9.1

bool wxXmlResource::LoadObjectRecursively (**wxObject** * *instance,* **wxWindow** * *parent,* **const wxString &** *name,* **const wxString &** *classname*)

Load an object from anywhere in the resource tree.

These methods are similar to LoadObject() but may be used to load an object from anywhere in the resource tree and not only the top level. Note that you will very rarely need to do this as in normal use the entire container window (defined at the top level) is loaded and not its individual children but this method can be useful in some particular situations.

Since

2.9.1

wxPanel* **wxXmlResource::LoadPanel** (**wxWindow** * *parent,* **const wxString &** *name*)

Loads a panel.

parent points to the parent window.

bool wxXmlResource::LoadPanel (wxPanel * *panel,* wxWindow * *parent,* const wxString & *name*)

Loads a panel.

parent points to the parent window. This form is used to finish creation of an already existing instance.

wxToolBar* wxXmlResource::LoadToolBar (wxWindow * *parent,* const wxString & *name*)

Loads a toolbar.

void wxXmlResource::ReportError (const wxXmlNode * *context,* const wxString & *message*) `[protected]`

Reports error in XRC resources to the user.

Any errors in XRC input files should be reported using this method (or its wxXmlResourceHandler::ReportError() equivalent). Unlike wxLogError(), this method presents the error to the user in a more usable form. In particular, the output is compiler-like and contains information about the exact location of the error.

Parameters

context	XML node the error occurred in or relates to. This can be NULL, but should be the most specific node possible, as its line number is what is reported to the user.
message	Text of the error message. This string should always be in English (i.e. not wrapped in _()). It shouldn't be a sentence – it should start with lower-case letter and shouldn't have a trailing period or exclamation point.

Since

2.9.0

See also

wxXmlResourceHandler::ReportError(), DoReportError()

static wxXmlResource* wxXmlResource::Set (wxXmlResource * *res*) `[static]`

Sets the global resources object and returns a pointer to the previous one (may be NULL).

void wxXmlResource::SetDomain (const wxString & *domain*)

Sets the domain (message catalog) that will be used to load translatable strings in the XRC.

void wxXmlResource::SetFlags (int *flags*)

Sets flags (bitlist of wxXmlResourceFlags enumeration values).

bool wxXmlResource::Unload (const wxString & *filename*)

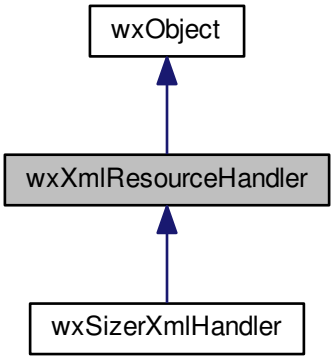
This function unloads a resource previously loaded by Load().

Returns true if the resource was successfully unloaded and false if it hasn't been found in the list of loaded resources.

21.874 wxXmlResourceHandler Class Reference

`#include <wx/xrc/xmlres.h>`

Inheritance diagram for wxXmlResourceHandler:



21.874.1 Detailed Description

wxSizerXmlHandler is a class for resource handlers capable of creating a wxSizer object from an XML node.

wxXmlResourceHandler is an abstract base class for resource handlers capable of creating a control from an XML node.

See also

wxXmlResourceHandler, wxSizer

Library: wxXRC

Category: XML Based Resource System (XRC)

See XML Based Resource System (XRC) for details.

Library: wxXRC

Category: XML Based Resource System (XRC)

Public Member Functions

- wxXmlResourceHandler ()

  Default constructor.
- virtual ~wxXmlResourceHandler ()

  Destructor.

- wxObject * CreateResource (wxXmlNode *node, wxObject *parent, wxObject *instance)

  Creates an object (menu, dialog, control, ...) from an XML node.
- virtual wxObject * DoCreateResource ()=0

  Called from CreateResource after variables were filled.
- virtual bool CanHandle (wxXmlNode *node)=0

  Returns true if it understands this node and can create a resource from it, false otherwise.
- void SetParentResource (wxXmlResource *res)

  Sets the parent resource.

Protected Member Functions

- void AddStyle (const wxString &name, int value)

  Add a style flag (e.g.
- void AddWindowStyles ()

  Add styles common to all wxWindow-derived classes.
- void CreateChildren (wxObject *parent, bool this_hnd_only=false)

  Creates children.
- void CreateChildrenPrivately (wxObject *parent, wxXmlNode *rootnode=NULL)

  Helper function.
- wxObject * CreateResFromNode (wxXmlNode *node, wxObject *parent, wxObject *instance=NULL)

  Creates a resource from a node.
- wxAnimation * GetAnimation (const wxString &param="animation")

  Creates an animation (see wxAnimation) from the filename specified in param.
- wxBitmap GetBitmap (const wxString &param="bitmap", const wxArtClient &defaultArtClient=wxART_OTHER, wxSize size=wxDefaultSize)

  Gets a bitmap.
- wxBitmap GetBitmap (const wxXmlNode *node, const wxArtClient &defaultArtClient=wxART_OTHER, wxSize size=wxDefaultSize)

  Gets a bitmap from an XmlNode.
- bool GetBool (const wxString &param, bool defaultv=false)

  Gets a bool flag (1, t, yes, on, true are true, everything else is false).
- wxColour GetColour (const wxString &param, const wxColour &defaultColour=wxNullColour)

  Gets colour in HTML syntax (#RRGGBB).
- wxFileSystem & GetCurFileSystem ()

  Returns the current file system.
- wxCoord GetDimension (const wxString &param, wxCoord defaultv=0, wxWindow *windowToUse=0)

  Gets a dimension (may be in dialog units).
- wxDirection GetDirection (const wxString &param, wxDirection dirDefault=wxLEFT)

  Gets a direction.
- wxFont GetFont (const wxString &param="font")

  Gets a font.
- int GetID ()

  Returns the XRCID.
- wxIcon GetIcon (const wxString &param="icon", const wxArtClient &defaultArtClient=wxART_OTHER, wxSize size=wxDefaultSize)

  Returns an icon.
- wxIcon GetIcon (const wxXmlNode *node, const wxArtClient &defaultArtClient=wxART_OTHER, wxSize size=wxDefaultSize)

  Gets an icon from an XmlNode.
- wxIconBundle GetIconBundle (const wxString &param, const wxArtClient &defaultArtClient=wxART_OTHER)

Returns an icon bundle.

- wxImageList * GetImageList (const wxString &param="imagelist")

  Creates an image list from the param markup data.
- long GetLong (const wxString &param, long defaultv=0)

  Gets the integer value from the parameter.
- float GetFloat (const wxString &param, float defaultv=0)

  Gets a float value from the parameter.
- wxString GetName ()

  Returns the resource name.
- bool IsObjectNode (const wxXmlNode *node) const

  Checks if the given node is an object node.
- wxString GetNodeContent (wxXmlNode *node)

  Gets node content from wxXML_ENTITY_NODE.
- wxXmlNode * GetNodeParent (const wxXmlNode *node) const

  Gets the parent of the node given.
- wxXmlNode * GetNodeNext (const wxXmlNode *node) const

  Gets the next sibling node related to the given node, possibly NULL.
- wxXmlNode * GetNodeChildren (const wxXmlNode *node) const

  Gets the first child of the given node or NULL.
- wxXmlNode * GetParamNode (const wxString &param)

  Finds the node or returns NULL.
- wxString GetParamValue (const wxString &param)

  Finds the parameter value or returns the empty string.
- wxString GetParamValue (const wxXmlNode *node)

  Returns the node parameter value.
- wxPoint GetPosition (const wxString &param="pos")

  Gets the position (may be in dialog units).
- wxSize GetSize (const wxString &param="size", wxWindow *windowToUse=0)

  Gets the size (may be in dialog units).
- int GetStyle (const wxString &param="style", int defaults=0)

  Gets style flags from text in form "flag | flag2| flag3 |..." Only understands flags added with AddStyle().
- wxString GetText (const wxString &param, bool translate=true)

  Gets text from param and does some conversions:
- bool HasParam (const wxString &param)

  Check to see if a parameter exists.
- bool IsOfClass (wxXmlNode *node, const wxString &classname)

  Convenience function.
- void SetupWindow (wxWindow *wnd)

  Sets common window options.
- void ReportError (wxXmlNode *context, const wxString &message)

  Reports error in XRC resources to the user.
- void ReportError (const wxString &message)

  Like ReportError(wxXmlNode, const wxString&), but uses the node of currently processed object (m_node) as the context.*
- void ReportParamError (const wxString &param, const wxString &message)

  Like ReportError(wxXmlNode, const wxString&), but uses the node of parameter param of the currently processed object as the context.*
- wxXmlResource * GetResource () const

  After CreateResource has been called this will return the current wxXmlResource object.
- wxXmlNode * GetNode () const

  After CreateResource has been called this will return the XML node being processed.

- wxString GetClass () const

  After CreateResource has been called this will return the class name of the XML resource node being processed.
- wxObject * GetParent () const

  After CreateResource has been called this will return the current item's parent, if any.
- wxObject * GetInstance () const

  After CreateResource has been called this will return the instance that the XML resource content should be created upon, if it has already been created.
- wxWindow * GetParentAsWindow () const

  After CreateResource has been called this will return the item's parent as a wxWindow.

Additional Inherited Members

21.874.2 Constructor & Destructor Documentation

wxXmlResourceHandler::wxXmlResourceHandler ( )

Default constructor.

virtual wxXmlResourceHandler::~wxXmlResourceHandler ( ) [virtual]

Destructor.

21.874.3 Member Function Documentation

void wxXmlResourceHandler::AddStyle (const wxString & *name,* int *value*) [protected]

Add a style flag (e.g.

`wxMB_DOCKABLE`) to the list of flags understood by this handler.

void wxXmlResourceHandler::AddWindowStyles ( ) [protected]

Add styles common to all wxWindow-derived classes.

virtual bool wxXmlResourceHandler::CanHandle (wxXmlNode * *node*) [pure virtual]

Returns true if it understands this node and can create a resource from it, false otherwise.

Note

You must not call any wxXmlResourceHandler methods except IsOfClass() from this method! The instance is not yet initialized with node data at the time CanHandle() is called and it is only safe to operate on node directly or to call IsOfClass().

Implemented in wxSizerXmlHandler.

void wxXmlResourceHandler::CreateChildren (wxObject * *parent,* bool *this_hnd_only* = `false`) [protected]

Creates children.

void wxXmlResourceHandler::CreateChildrenPrivately (wxObject * *parent*, wxXmlNode * *rootnode* = NULL) [protected]

Helper function.

wxObject* wxXmlResourceHandler::CreateResFromNode (wxXmlNode * *node*, wxObject * *parent*, wxObject * *instance* = NULL) [protected]

Creates a resource from a node.

wxObject* wxXmlResourceHandler::CreateResource (wxXmlNode * *node*, wxObject * *parent*, wxObject * *instance*)

Creates an object (menu, dialog, control, ...) from an XML node.

Should check for validity. *parent* is a higher-level object (usually window, dialog or panel) that is often necessary to create the resource.

If **instance** is non-NULL it should not create a new instance via 'new' but should rather use this one, and call its Create method.

virtual wxObject* wxXmlResourceHandler::DoCreateResource ( ) [pure virtual]

Called from CreateResource after variables were filled.

Implemented in wxSizerXmlHandler.

wxAnimation* wxXmlResourceHandler::GetAnimation (const wxString & *param* = "animation") [protected]

Creates an animation (see wxAnimation) from the filename specified in *param*.

wxBitmap wxXmlResourceHandler::GetBitmap (const wxString & *param* = "bitmap", const wxArtClient & *defaultArtClient* = wxART_OTHER, wxSize *size* = wxDefaultSize) [protected]

Gets a bitmap.

wxBitmap wxXmlResourceHandler::GetBitmap (const wxXmlNode * *node*, const wxArtClient & *defaultArtClient* = wxART_OTHER, wxSize *size* = wxDefaultSize) [protected]

Gets a bitmap from an XmlNode.

Since

2.9.1

bool wxXmlResourceHandler::GetBool (const wxString & *param*, bool *defaultv* = false) [protected]

Gets a bool flag (1, t, yes, on, true are true, everything else is false).

wxString wxXmlResourceHandler::GetClass ( ) const [protected]

After CreateResource has been called this will return the class name of the XML resource node being processed.

Since

2.9.5

wxColour wxXmlResourceHandler::GetColour (const wxString & *param*, const wxColour & *defaultColour* = wxNullColour) [protected]

Gets colour in HTML syntax (#RRGGBB).

wxFileSystem& wxXmlResourceHandler::GetCurFileSystem ( ) [protected]

Returns the current file system.

wxCoord wxXmlResourceHandler::GetDimension (const wxString & *param*, wxCoord *defaultv* = 0, wxWindow * *windowToUse* = 0) [protected]

Gets a dimension (may be in dialog units).

wxDirection wxXmlResourceHandler::GetDirection (const wxString & *param*, wxDirection *dirDefault* = wxLEFT) [protected]

Gets a direction.

If the given *param* is not present or has empty value, *dirDefault* is returned by default. Otherwise the value of the parameter is parsed and a warning is generated if it's not one of `wxLEFT`, `wxTOP`, `wxRIGHT` or `wxBOTTOM`.

Since

2.9.3

float wxXmlResourceHandler::GetFloat (const wxString & *param*, float *defaultv* = 0) [protected]

Gets a float value from the parameter.

wxFont wxXmlResourceHandler::GetFont (const wxString & *param* = "font") [protected]

Gets a font.

wxIcon wxXmlResourceHandler::GetIcon (const wxString & *param* = "icon", const wxArtClient & *defaultArtClient* = wxART_OTHER, wxSize *size* = wxDefaultSize) [protected]

Returns an icon.

wxIcon wxXmlResourceHandler::GetIcon (const wxXmlNode * *node*, const wxArtClient & *defaultArtClient* = wxART_OTHER, wxSize *size* = wxDefaultSize) [protected]

Gets an icon from an XmlNode.

Since

2.9.1

wxIconBundle wxXmlResourceHandler::GetIconBundle (const wxString & *param,* const wxArtClient & *defaultArtClient* = wxART_OTHER) [protected]

Returns an icon bundle.

Note

Bundles can be loaded either with stock IDs or from files that contain more than one image (e.g. Windows icon files). If a file contains only single image, a bundle with only one icon will be created.

Since

2.9.0

int wxXmlResourceHandler::GetID ( ) [protected]

Returns the XRCID.

wxImageList* wxXmlResourceHandler::GetImageList (const wxString & *param* = "imagelist") [protected]

Creates an image list from the *param* markup data.

Returns

The new instance of wxImageList or NULL if no data is found.

Since

2.9.1

wxObject* wxXmlResourceHandler::GetInstance ( ) const [protected]

After CreateResource has been called this will return the instance that the XML resource content should be created upon, if it has already been created.

If NULL then the handler should create the object itself.

Since

2.9.5

long wxXmlResourceHandler::GetLong (const wxString & *param,* long *defaultv* = 0) [protected]

Gets the integer value from the parameter.

wxString wxXmlResourceHandler::GetName ( ) [protected]

Returns the resource name.

wxXmlNode* wxXmlResourceHandler::GetNode ( ) const [protected]

After CreateResource has been called this will return the XML node being processed.

Since

2.9.5

wxXmlNode* **wxXmlResourceHandler::GetNodeChildren** (**const wxXmlNode** * ***node***) **const** [protected]

Gets the first child of the given node or NULL.

This method is safe to call with NULL argument, it just returns NULL in this case.

Since

3.1.0

wxString **wxXmlResourceHandler::GetNodeContent** (**wxXmlNode** * ***node***) [protected]

Gets node content from wxXML_ENTITY_NODE.

wxXmlNode* **wxXmlResourceHandler::GetNodeNext** (**const wxXmlNode** * ***node***) **const** [protected]

Gets the next sibling node related to the given node, possibly NULL.

This method is safe to call with NULL argument, it just returns NULL in this case.

Since

3.1.0

wxXmlNode* **wxXmlResourceHandler::GetNodeParent** (**const wxXmlNode** * ***node***) **const** [protected]

Gets the parent of the node given.

This method is safe to call with NULL argument, it just returns NULL in this case.

Since

3.1.0

wxXmlNode* **wxXmlResourceHandler::GetParamNode** (**const wxString** & ***param***) [protected]

Finds the node or returns NULL.

wxString **wxXmlResourceHandler::GetParamValue** (**const wxString** & ***param***) [protected]

Finds the parameter value or returns the empty string.

wxString **wxXmlResourceHandler::GetParamValue** (**const wxXmlNode** * ***node***) [protected]

Returns the node parameter value.

Since

2.9.1

wxObject* wxXmlResourceHandler::GetParent ( ) const [protected]

After CreateResource has been called this will return the current item's parent, if any.

Since

2.9.5

wxWindow* wxXmlResourceHandler::GetParentAsWindow ( ) const [protected]

After CreateResource has been called this will return the item's parent as a wxWindow.

Since

2.9.5

wxPoint wxXmlResourceHandler::GetPosition (const wxString & *param* = "pos") [protected]

Gets the position (may be in dialog units).

wxXmlResource* wxXmlResourceHandler::GetResource ( ) const [protected]

After CreateResource has been called this will return the current wxXmlResource object.

Since

2.9.5

wxSize wxXmlResourceHandler::GetSize (const wxString & *param* = "size", wxWindow * *windowToUse* = 0) [protected]

Gets the size (may be in dialog units).

int wxXmlResourceHandler::GetStyle (const wxString & *param* = "style", int *defaults* = 0) [protected]

Gets style flags from text in form "flag | flag2| flag3 |..." Only understands flags added with AddStyle().

wxString wxXmlResourceHandler::GetText (const wxString & *param*, bool *translate* = true) [protected]

Gets text from param and does some conversions:

- replaces \n, \r, \t by respective characters (according to C syntax)
- replaces $ by and $$ by $ (needed for _File to File translation because of XML syntax)
- calls wxGetTranslations (unless disabled in wxXmlResource)

bool wxXmlResourceHandler::HasParam (const wxString & *param*) [protected]

Check to see if a parameter exists.

bool wxXmlResourceHandler::IsObjectNode (const wxXmlNode * *node*) const [protected]

Checks if the given node is an object node.

Object nodes are those named "object" or "object_ref".

Since

3.1.0

bool wxXmlResourceHandler::IsOfClass (wxXmlNode * *node,* const wxString & *classname*) [protected]

Convenience function.

Returns true if the node has a property class equal to classname, e.g. object class="wxDialog".

void wxXmlResourceHandler::ReportError (wxXmlNode * *context,* const wxString & *message*) [protected]

Reports error in XRC resources to the user.

See wxXmlResource::ReportError() for more information.

Since

2.9.0

void wxXmlResourceHandler::ReportError (const wxString & *message*) [protected]

Like ReportError(wxXmlNode*, const wxString&), but uses the node of currently processed object (m_node) as the context.

Since

2.9.0

void wxXmlResourceHandler::ReportParamError (const wxString & *param,* const wxString & *message*) [protected]

Like ReportError(wxXmlNode*, const wxString&), but uses the node of parameter *param* of the currently processed object as the context.

This is convenience function for reporting errors in particular parameters.

Since

2.9.0

void wxXmlResourceHandler::SetParentResource (wxXmlResource * *res*)

Sets the parent resource.

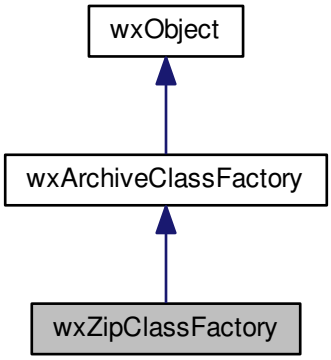
void wxXmlResourceHandler::SetupWindow (wxWindow * *wnd*) [protected]

Sets common window options.

21.875 wxZipClassFactory Class Reference

`#include <wx/zipstrm.h>`

Inheritance diagram for wxZipClassFactory:



21.875.1 Detailed Description

Class factory for the zip archive format.

See the base class for details.

Library: wxBase

Category: Archive support, Streams

See also

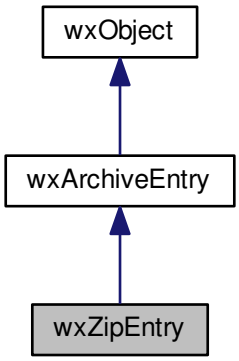
Archive Formats, Generic Archive Programming, wxZipEntry, wxZipInputStream, wxZipOutputStream

Additional Inherited Members

21.876 wxZipEntry Class Reference

`#include <wx/zipstrm.h>`

Inheritance diagram for wxZipEntry:



21.876.1 Detailed Description

Holds the meta-data for an entry in a zip.

21.876.2 Field availability

When reading a zip from a stream that is seekable, wxZipEntry::GetNextEntry() returns a fully populated wxZipEntry object except for wxZipEntry::GetLocalExtra(). wxZipEntry::GetLocalExtra() becomes available when the entry is opened, either by calling wxZipInputStream::OpenEntry() or by making an attempt to read the entry's data.

For zips on non-seekable streams, the following fields are always available when wxZipEntry::GetNextEntry() returns:

- wxZipEntry::GetDateTime
- wxZipEntry::GetInternalFormat
- wxZipEntry::GetInternalName
- wxZipEntry::GetFlags
- wxZipEntry::GetLocalExtra
- wxZipEntry::GetMethod
- wxZipEntry::GetName

- wxZipEntry::GetOffset
- wxZipEntry::IsDir

The following fields are also usually available when GetNextEntry() returns, however, if the zip was also written to a non-seekable stream the zipper is permitted to store them after the entry's data. In that case they become available when the entry's data has been read to Eof(), or CloseEntry() has been called. (GetFlags() & wxZIP_SUMS_FOLLOW) != 0 indicates that one or more of these come after the data:

- wxZipEntry::GetCompressedSize
- wxZipEntry::GetCrc
- wxZipEntry::GetSize

The following are stored at the end of the zip, and become available when the end of the zip has been reached, i.e. after GetNextEntry() returns NULL and Eof() is true:

- wxZipEntry::GetComment
- wxZipEntry::GetExternalAttributes
- wxZipEntry::GetExtra
- wxZipEntry::GetMode
- wxZipEntry::GetSystemMadeBy
- wxZipEntry::IsReadOnly
- wxZipEntry::IsMadeByUnix
- wxZipEntry::IsText

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, wxZipInputStream, wxZipOutputStream, wxZipNotifier

Public Member Functions

- wxZipEntry (const wxString &name=wxEmptyString, const wxDateTime &dt=Now(), wxFileOffset size=wxInvalidOffset)
- wxZipEntry (const wxZipEntry &entry)

  Copy constructor.
- wxZipEntry * Clone () const

  Make a copy of this entry.
- wxFileOffset GetCompressedSize () const

  The compressed size of this entry in bytes.
- wxUint32 GetCrc () const

  CRC32 for this entry's data.
- int GetFlags () const

  Returns a combination of the bits flags in the enumeration `wxZipFlags`.
- bool IsMadeByUnix () const

Returns true if GetSystemMadeBy() is a flavour of unix.

- wxZipEntry & operator= (const wxZipEntry &entry)

  Assignment operator.

- wxString GetComment () const

  Gets and sets the short comment for this entry.
- void SetComment (const wxString &comment)

  Gets and sets the short comment for this entry.

- wxUint32 GetExternalAttributes () const

  The low 8 bits are always the DOS/Windows file attributes for this entry.
- void SetExternalAttributes (wxUint32 attr)

  The low 8 bits are always the DOS/Windows file attributes for this entry.

- const char * GetExtra () const

  The extra field from the entry's central directory record.
- size_t GetExtraLen () const

  The extra field from the entry's central directory record.
- void SetExtra (const char *extra, size_t len)

  The extra field from the entry's central directory record.

- const char * GetLocalExtra () const

  The extra field from the entry's local record.
- size_t GetLocalExtraLen () const

  The extra field from the entry's local record.
- void SetLocalExtra (const char *extra, size_t len)

  The extra field from the entry's local record.

- int GetMethod () const

  The compression method.
- void SetMethod (int method)

  The compression method.

- int GetMode () const

  If IsMadeByUnix() is true then returns the unix permission bits stored in GetExternalAttributes().
- void SetMode (int mode)

  Sets the DOS attributes in GetExternalAttributes() to be consistent with the mode given.

- int GetSystemMadeBy () const

  The originating file-system.
- void SetSystemMadeBy (int system)

  The originating file-system.

- wxString GetInternalName (const wxString &name, wxPathFormat format=wxPATH_NATIVE, bool *pIsDir=NULL)

  A static member that translates a filename into the internal format used within the archive.
- wxString GetInternalName () const

  Returns the entry's filename in the internal format used within the archive.

- bool IsText () const

  Indicates that this entry's data is text in an 8-bit encoding.

- void SetIsText (bool isText=true)

  Indicates that this entry's data is text in an 8-bit encoding.

- void SetNotifier (wxZipNotifier &notifier)

  Sets the notifier (see wxZipNotifier) for this entry.

- void UnsetNotifier ()

  Sets the notifier (see wxZipNotifier) for this entry.

Additional Inherited Members

21.876.3 Constructor & Destructor Documentation

wxZipEntry::wxZipEntry (const wxString & *name* = wxEmptyString, const wxDateTime & *dt* = Now(), wxFileOffset *size* = wxInvalidOffset)

wxZipEntry::wxZipEntry (const wxZipEntry & *entry*)

Copy constructor.

21.876.4 Member Function Documentation

wxZipEntry* wxZipEntry::Clone ( ) const

Make a copy of this entry.

wxString wxZipEntry::GetComment ( ) const

Gets and sets the short comment for this entry.

wxFileOffset wxZipEntry::GetCompressedSize ( ) const

The compressed size of this entry in bytes.

wxUint32 wxZipEntry::GetCrc ( ) const

CRC32 for this entry's data.

wxUint32 wxZipEntry::GetExternalAttributes ( ) const

The low 8 bits are always the DOS/Windows file attributes for this entry.

The values of these attributes are given in the enumeration wxZipAttributes.

The remaining bits can store platform specific permission bits or attributes, and their meaning depends on the value of SetSystemMadeBy(). If IsMadeByUnix() is true then the high 16 bits are unix mode bits.

The following other accessors access these bits:

- IsReadOnly() / SetIsReadOnly()
- IsDir() / SetIsDir()
- GetMode() / SetMode()

const char* wxZipEntry::GetExtra ( ) const

The extra field from the entry's central directory record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

size_t wxZipEntry::GetExtraLen ( ) const

The extra field from the entry's central directory record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

int wxZipEntry::GetFlags ( ) const

Returns a combination of the bits flags in the enumeration `wxZipFlags`.

wxString wxZipEntry::GetInternalName (const wxString & *name*, wxPathFormat *format* = wxPATH_NATIVE, bool * *pIsDir* = NULL)

A static member that translates a filename into the internal format used within the archive.

If the third parameter is provided, the bool pointed to is set to indicate whether the name looks like a directory name (i.e. has a trailing path separator).

See also

Looking Up an Archive Entry by Name

wxString wxZipEntry::GetInternalName ( ) const [virtual]

Returns the entry's filename in the internal format used within the archive.

The name can include directory components, i.e. it can be a full path.

The names of directory entries are returned without any trailing path separator. This gives a canonical name that can be used in comparisons.

Implements wxArchiveEntry.

const char* wxZipEntry::GetLocalExtra ( ) const

The extra field from the entry's local record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

size_t wxZipEntry::GetLocalExtraLen ( ) const

The extra field from the entry's local record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

int wxZipEntry::GetMethod ( ) const

The compression method.

The enumeration wxZipMethod lists the possible values.

The default constructor sets this to `wxZIP_METHOD_DEFAULT`, which allows wxZipOutputStream to choose the method when writing the entry.

int wxZipEntry::GetMode ( ) const

If IsMadeByUnix() is true then returns the unix permission bits stored in GetExternalAttributes().

Otherwise synthesises them from the DOS attributes.

int wxZipEntry::GetSystemMadeBy ( ) const

The originating file-system.

The default constructor sets this to `wxZIP_SYSTEM_MSDOS`. Set it to `wxZIP_SYSTEM_UNIX` in order to be able to store unix permissions using SetMode().

bool wxZipEntry::IsMadeByUnix ( ) const

Returns true if GetSystemMadeBy() is a flavour of unix.

bool wxZipEntry::IsText ( ) const

Indicates that this entry's data is text in an 8-bit encoding.

wxZipEntry& wxZipEntry::operator= (const wxZipEntry & *entry*)

Assignment operator.

void wxZipEntry::SetComment (const wxString & *comment*)

Gets and sets the short comment for this entry.

void wxZipEntry::SetExternalAttributes (wxUint32 *attr*)

The low 8 bits are always the DOS/Windows file attributes for this entry.

The values of these attributes are given in the enumeration wxZipAttributes.

The remaining bits can store platform specific permission bits or attributes, and their meaning depends on the value of SetSystemMadeBy(). If IsMadeByUnix() is true then the high 16 bits are unix mode bits.

The following other accessors access these bits:

- IsReadOnly() / SetIsReadOnly()
- IsDir() / SetIsDir()
- GetMode() / SetMode()

void wxZipEntry::SetExtra (const char * *extra*, size_t *len*)

The extra field from the entry's central directory record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

void wxZipEntry::SetIsText (bool *isText* = `true`)

Indicates that this entry's data is text in an 8-bit encoding.

void wxZipEntry::SetLocalExtra (const char * *extra*, size_t *len*)

The extra field from the entry's local record.

The extra field is used to store platform or application specific data. See Pkware's document 'appnote.txt' for information on its format.

void wxZipEntry::SetMethod (int *method*)

The compression method.

The enumeration wxZipMethod lists the possible values.

The default constructor sets this to `wxZIP_METHOD_DEFAULT`, which allows wxZipOutputStream to choose the method when writing the entry.

void wxZipEntry::SetMode (int *mode*)

Sets the DOS attributes in GetExternalAttributes() to be consistent with the *mode* given.

If IsMadeByUnix() is true then also stores *mode* in GetExternalAttributes(). Note that the default constructor sets GetSystemMadeBy() to `wxZIP_SYSTEM_MSDOS` by default. So to be able to store unix permissions when creating zips, call SetSystemMadeBy(wxZIP_SYSTEM_UNIX).

void wxZipEntry::SetNotifier (wxZipNotifier & *notifier*)

Sets the notifier (see wxZipNotifier) for this entry.

Whenever the wxZipInputStream updates this entry, it will then invoke the associated notifier's wxZipNotifier::OnEntryUpdated() method.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an zip in a pipeline (i.e. between non-seekable streams).

See also

Archives on Non-Seekable Streams, wxZipNotifier

void wxZipEntry::SetSystemMadeBy (int *system*)

The originating file-system.

The default constructor sets this to `wxZIP_SYSTEM_MSDOS`. Set it to `wxZIP_SYSTEM_UNIX` in order to be able to store unix permissions using SetMode().

void wxZipEntry::UnsetNotifier ( ) [virtual]

Sets the notifier (see wxZipNotifier) for this entry.

Whenever the wxZipInputStream updates this entry, it will then invoke the associated notifier's wxZipNotifier::OnEntryUpdated() method.

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an zip in a pipeline (i.e. between non-seekable streams).

See also

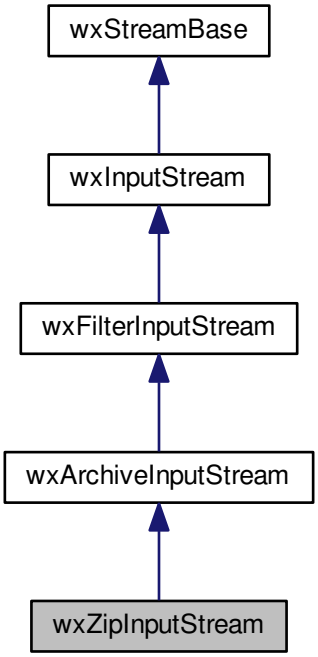
Archives on Non-Seekable Streams, wxZipNotifier

Reimplemented from wxArchiveEntry.

21.877 wxZipInputStream Class Reference

`#include <wx/zipstrm.h>`

Inheritance diagram for wxZipInputStream:



21.877.1 Detailed Description

Input stream for reading zip files.

wxZipInputStream::GetNextEntry() returns a wxZipEntry object containing the meta-data for the next entry in the zip (and gives away ownership). Reading from the wxZipInputStream then returns the entry's data. Eof() becomes

true after an attempt has been made to read past the end of the entry's data. When there are no more entries, GetNextEntry() returns NULL and sets Eof().

Note that in general zip entries are not seekable, and wxZipInputStream::SeekI() always returns wxInvalidOffset.

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, wxZipEntry, wxZipOutputStream

Public Member Functions

- bool CloseEntry ()

  Closes the current entry.
- wxString GetComment ()

  Returns the zip comment.
- wxZipEntry * GetNextEntry ()

  Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a wxZipEntry object, giving away ownership.
- int GetTotalEntries ()

  For a zip on a seekable stream returns the total number of entries in the zip.
- bool OpenEntry (wxZipEntry &entry)

  Closes the current entry if one is open, then opens the entry specified by the entry object.

- wxZipInputStream (wxInputStream &stream, wxMBConv &conv=wxConvLocal)

  Constructor.
- wxZipInputStream (wxInputStream *stream, wxMBConv &conv=wxConvLocal)

  Constructor.

Additional Inherited Members

21.877.2 Constructor & Destructor Documentation

wxZipInputStream::wxZipInputStream (wxInputStream & *stream,* wxMBConv & *conv* = `wxConvLocal`)

Constructor.

In a Unicode build the second parameter *conv* is used to translate the filename and comment fields into Unicode. It has no effect on the stream's data. If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

wxZipInputStream::wxZipInputStream (wxInputStream * *stream,* wxMBConv & *conv* = `wxConvLocal`)

Constructor.

In a Unicode build the second parameter *conv* is used to translate the filename and comment fields into Unicode. It has no effect on the stream's data. If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

21.877.3 Member Function Documentation

bool wxZipInputStream::CloseEntry ( ) [virtual]

Closes the current entry.

On a non-seekable stream reads to the end of the current entry first.

Implements wxArchiveInputStream.

wxString wxZipInputStream::GetComment ( )

Returns the zip comment.

This is stored at the end of the zip, therefore when reading a zip from a non-seekable stream, it returns the empty string until the end of the zip has been reached, i.e. when GetNextEntry() returns NULL.

wxZipEntry* wxZipInputStream::GetNextEntry ( )

Closes the current entry if one is open, then reads the meta-data for the next entry and returns it in a wxZipEntry object, giving away ownership.

The stream is then open and can be read.

int wxZipInputStream::GetTotalEntries ( )

For a zip on a seekable stream returns the total number of entries in the zip.

For zips on non-seekable streams returns the number of entries returned so far by GetNextEntry().

bool wxZipInputStream::OpenEntry (wxZipEntry & *entry*)

Closes the current entry if one is open, then opens the entry specified by the *entry* object.

entry should be from the same zip file, and the zip should be on a seekable stream.

See also

overview_archive_byname

21.878 wxZipNotifier Class Reference

```
#include <wx/zipstrm.h>
```

21.878.1 Detailed Description

If you need to know when a wxZipInputStream updates a wxZipEntry, you can create a notifier by deriving from this abstract base class, overriding wxZipNotifier::OnEntryUpdated().

An instance of your notifier class can then be assigned to wxZipEntry objects, using wxZipEntry::SetNotifier().

Setting a notifier is not usually necessary. It is used to handle certain cases when modifying an zip in a pipeline (i.e. between non-seekable streams). See Archives on Non-Seekable Streams.

Library: wxBase

Category: Archive support, Streams

See also

Archives on Non-Seekable Streams, wxZipEntry, wxZipInputStream, wxZipOutputStream

Public Member Functions

- virtual void OnEntryUpdated (wxZipEntry &entry)=0

  Override this to receive notifications when an wxZipEntry object changes.

21.878.2 Member Function Documentation

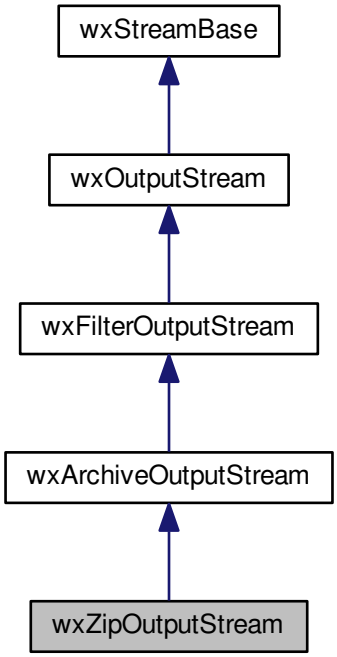
virtual void wxZipNotifier::OnEntryUpdated (wxZipEntry & *entry*) `[pure virtual]`

Override this to receive notifications when an wxZipEntry object changes.

21.879 wxZipOutputStream Class Reference

`#include <wx/zipstrm.h>`

Inheritance diagram for wxZipOutputStream:



21.879.1 Detailed Description

Output stream for writing zip files.

wxZipOutputStream::PutNextEntry() is used to create a new entry in the output zip, then the entry's data is written to the wxZipOutputStream. Another call to wxZipOutputStream::PutNextEntry() closes the current entry and begins the next.

Library: wxBase

Category: Archive support, Streams

See also

Archive Formats, wxZipEntry, wxZipInputStream

Public Member Functions

- virtual ~wxZipOutputStream ()

  The destructor calls Close() to finish writing the zip if it has not been called already.
- bool Close ()

  Finishes writing the zip, returning true if successful.
- bool CloseEntry ()

  Close the current entry.
- bool CopyArchiveMetaData (wxZipInputStream &inputStream)

  Transfers the zip comment from the wxZipInputStream to this output stream.
- bool CopyEntry (wxZipEntry *entry, wxZipInputStream &inputStream)

  Takes ownership of entry and uses it to create a new entry in the zip.
- bool PutNextDirEntry (const wxString &name, const wxDateTime &dt=wxDateTime::Now())

  Create a new directory entry (see wxArchiveEntry::IsDir) with the given name and timestamp.
- void SetComment (const wxString &comment)

  Sets a comment for the zip as a whole.

- wxZipOutputStream (wxOutputStream &stream, int level=-1, wxMBConv &conv=wxConvLocal)

  Constructor.
- wxZipOutputStream (wxOutputStream *stream, int level=-1, wxMBConv &conv=wxConvLocal)

  Constructor.

- int GetLevel () const

  Set the compression level that will be used the next time an entry is created.
- void SetLevel (int level)

  Set the compression level that will be used the next time an entry is created.

- bool PutNextEntry (wxZipEntry *entry)

  Takes ownership of entry and uses it to create a new entry in the zip.
- bool PutNextEntry (const wxString &name, const wxDateTime &dt=wxDateTime::Now(), wxFileOffset size=wxInvalidOffset)

  Create a new entry with the given name, timestamp and size.

Additional Inherited Members

21.879.2 Constructor & Destructor Documentation

wxZipOutputStream::wxZipOutputStream (wxOutputStream & *stream,* int *level* = -1, wxMBConv & *conv* = wxConvLocal)

Constructor.

level is the compression level to use. It can be a value between 0 and 9 or -1 to use the default value which currently is equivalent to 6.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not. In a Unicode build the third parameter *conv* is used to translate the filename and comment fields to an 8-bit encoding. It has no effect on the stream's data.

wxZipOutputStream::wxZipOutputStream (wxOutputStream * *stream,* int *level* = -1, wxMBConv & *conv* = wxConvLocal)

Constructor.

level is the compression level to use. It can be a value between 0 and 9 or -1 to use the default value which currently is equivalent to 6.

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not. In a Unicode build the third parameter *conv* is used to translate the filename and comment fields to an 8-bit encoding. It has no effect on the stream's data.

virtual wxZipOutputStream::~wxZipOutputStream ( ) [virtual]

The destructor calls Close() to finish writing the zip if it has not been called already.

21.879.3 Member Function Documentation

bool wxZipOutputStream::Close ( ) [virtual]

Finishes writing the zip, returning true if successful.

Called by the destructor if not called explicitly.

Reimplemented from wxArchiveOutputStream.

bool wxZipOutputStream::CloseEntry ( ) [virtual]

Close the current entry.

It is called implicitly whenever another new entry is created with CopyEntry() or PutNextEntry(), or when the zip is closed.

Implements wxArchiveOutputStream.

bool wxZipOutputStream::CopyArchiveMetaData (wxZipInputStream & *inputStream*)

Transfers the zip comment from the wxZipInputStream to this output stream.

bool wxZipOutputStream::CopyEntry (wxZipEntry * *entry*, wxZipInputStream & *inputStream*)

Takes ownership of *entry* and uses it to create a new entry in the zip.

entry is then opened in *inputStream* and its contents copied to this stream.

CopyEntry() is much more efficient than transferring the data using Read() and Write() since it will copy them without decompressing and recompressing them.

For zips on seekable streams, *entry* must be from the same zip file as *inputStream*. For non-seekable streams, *entry* must also be the last thing read from *inputStream*.

int wxZipOutputStream::GetLevel ( ) const

Set the compression level that will be used the next time an entry is created.

It can be a value between 0 and 9 or -1 to use the default value which currently is equivalent to 6.

bool wxZipOutputStream::PutNextDirEntry (const wxString & *name*, const wxDateTime & *dt* = wxDateTime::Now()) [virtual]

Create a new directory entry (see wxArchiveEntry::IsDir) with the given name and timestamp.

PutNextEntry() can also be used to create directory entries, by supplying a name with a trailing path separator.

Implements wxArchiveOutputStream.

bool wxZipOutputStream::PutNextEntry (wxZipEntry * *entry*)

Takes ownership of *entry* and uses it to create a new entry in the zip.

bool wxZipOutputStream::PutNextEntry (const wxString & *name*, const wxDateTime & *dt* = wxDateTime::Now(), wxFileOffset *size* = wxInvalidOffset) [virtual]

Create a new entry with the given name, timestamp and size.

Implements wxArchiveOutputStream.

void wxZipOutputStream::SetComment (const wxString & *comment*)

Sets a comment for the zip as a whole.

It is written at the end of the zip.

void wxZipOutputStream::SetLevel (int *level*)

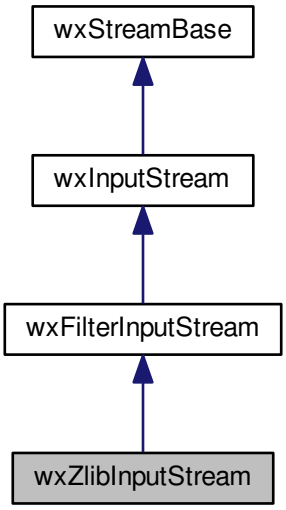
Set the compression level that will be used the next time an entry is created.

It can be a value between 0 and 9 or -1 to use the default value which currently is equivalent to 6.

21.880 wxZlibInputStream Class Reference

```
#include <wx/zstream.h>
```

Inheritance diagram for wxZlibInputStream:



21.880.1 Detailed Description

This filter stream decompresses a stream that is in zlib or gzip format.

Note that reading the gzip format requires zlib version 1.2.1 or greater, (the builtin version does support gzip format).

The stream is not seekable, wxInputStream::SeekI returns wxInvalidOffset. Also wxStreamBase::GetSize() is not supported, it always returns 0.

Library: wxBase

Category: Archive support, Streams

See also

wxInputStream, wxZlibOutputStream.

Public Member Functions

- wxZlibInputStream (wxInputStream &stream, int flags=wxZLIB_AUTO)

  If the parent stream is passed as a pointer then the new filter stream takes ownership of it.
- wxZlibInputStream (wxInputStream *stream, int flags=wxZLIB_AUTO)

  If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

- bool SetDictionary (const char *data, const size_t datalen)

  Sets the dictionary to the specified chunk of data.
- bool SetDictionary (const wxMemoryBuffer &buf)

  Sets the dictionary to the specified chunk of data.

Static Public Member Functions

- static bool CanHandleGZip ()

  Returns true if zlib library in use can handle gzip compressed data.

Additional Inherited Members

21.880.2 Constructor & Destructor Documentation

wxZlibInputStream::wxZlibInputStream (wxInputStream & *stream,* int *flags* = wxZLIB_AUTO)

If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

If it is passed by reference then it does not.

The *flags* wxZLIB_ZLIB and wxZLIB_GZIP specify whether the input data is in zlib or gzip format. If wxZLIB_AUTO is used, then zlib will autodetect the stream type, this is the default.

If *flags* is wxZLIB_NO_HEADER, then the data is assumed to be a raw deflate stream without either zlib or gzip headers. This is a lower level mode, which is not usually used directly. It can be used to read a raw deflate stream embedded in a higher level protocol.

The values of the wxZLibFlags enumeration can be used.

wxZlibInputStream::wxZlibInputStream (wxInputStream * *stream,* int *flags* = wxZLIB_AUTO)

If the parent stream is passed as a pointer then the new filter stream takes ownership of it.

If it is passed by reference then it does not.

The *flags* wxZLIB_ZLIB and wxZLIB_GZIP specify whether the input data is in zlib or gzip format. If wxZLIB_AUTO is used, then zlib will autodetect the stream type, this is the default.

If *flags* is wxZLIB_NO_HEADER, then the data is assumed to be a raw deflate stream without either zlib or gzip headers. This is a lower level mode, which is not usually used directly. It can be used to read a raw deflate stream embedded in a higher level protocol.

The values of the wxZLibFlags enumeration can be used.

21.880.3 Member Function Documentation

static bool wxZlibInputStream::CanHandleGZip ( ) [static]

Returns true if zlib library in use can handle gzip compressed data.

bool wxZlibInputStream::SetDictionary (const char * *data,* const size_t *datalen*)

Sets the dictionary to the specified chunk of data.

This can improve compression rate but note that the dictionary has to be the same when you deflate the data as when you inflate the data, otherwise you will inflate corrupted data.

Returns true if the dictionary was successfully set.

bool wxZlibInputStream::SetDictionary (const wxMemoryBuffer & *buf*)

Sets the dictionary to the specified chunk of data.

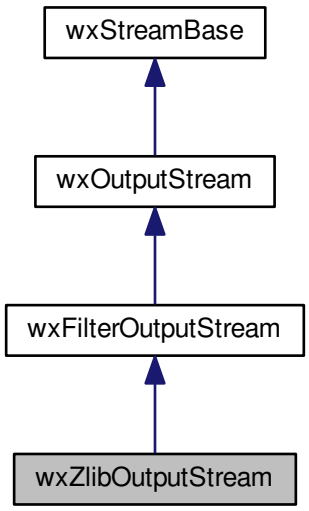
This can improve compression rate but note that the dictionary has to be the same when you deflate the data as when you inflate the data, otherwise you will inflate corrupted data.

Returns true if the dictionary was successfully set.

21.881 wxZlibOutputStream Class Reference

`#include <wx/zstream.h>`

Inheritance diagram for wxZlibOutputStream:



21.881.1 Detailed Description

This stream compresses all data written to it.

The compressed output can be in zlib or gzip format. Note that writing the gzip format requires zlib version 1.2.1 or greater (the builtin version does support gzip format).

The stream is not seekable, wxOutputStream::SeekO() returns wxInvalidOffset.

Library: wxBase

Category: Archive support, Streams

See also

wxOutputStream, wxZlibInputStream

Public Member Functions

- wxZlibOutputStream (wxOutputStream &stream, int level=-1, int flags=wxZLIB_ZLIB)

Creates a new write-only compressed stream.

- wxZlibOutputStream (wxOutputStream *stream, int level=-1, int flags=wxZLIB_ZLIB)

  Creates a new write-only compressed stream.

- bool SetDictionary (const char *data, const size_t datalen)

  Sets the dictionary to the specified chunk of data.

- bool SetDictionary (const wxMemoryBuffer &buf)

  Sets the dictionary to the specified chunk of data.

Static Public Member Functions

- static bool CanHandleGZip ()

  Returns true if zlib library in use can handle gzip compressed data.

Additional Inherited Members

21.881.2 Constructor & Destructor Documentation

wxZlibOutputStream::wxZlibOutputStream (wxOutputStream & *stream,* int *level* = −1, int *flags* = wxZLIB_ZLIB)

Creates a new write-only compressed stream.

level means level of compression. It is number between 0 and 9 (including these values) where 0 means no compression and 9 best but slowest compression. -1 is default value (currently equivalent to 6).

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

The *flags* wxZLIB_ZLIB and wxZLIB_GZIP specify whether the output data will be in zlib or gzip format. wxZLIB_↩ ZLIB is the default.

If *flags* is wxZLIB_NO_HEADER, then a raw deflate stream is output without either zlib or gzip headers. This is a lower level mode, which is not usually used directly. It can be used to embed a raw deflate stream in a higher level protocol.

The values of the wxZlibCompressionLevels and wxZLibFlags enumerations can be used.

wxZlibOutputStream::wxZlibOutputStream (wxOutputStream * *stream,* int *level* = −1, int *flags* = wxZLIB_ZLIB)

Creates a new write-only compressed stream.

level means level of compression. It is number between 0 and 9 (including these values) where 0 means no compression and 9 best but slowest compression. -1 is default value (currently equivalent to 6).

If the parent stream is passed as a pointer then the new filter stream takes ownership of it. If it is passed by reference then it does not.

The *flags* wxZLIB_ZLIB and wxZLIB_GZIP specify whether the output data will be in zlib or gzip format. wxZLIB_↩ ZLIB is the default.

If *flags* is wxZLIB_NO_HEADER, then a raw deflate stream is output without either zlib or gzip headers. This is a lower level mode, which is not usually used directly. It can be used to embed a raw deflate stream in a higher level protocol.

The values of the wxZlibCompressionLevels and wxZLibFlags enumerations can be used.

21.881.3 Member Function Documentation

static bool wxZlibOutputStream::CanHandleGZip ( ) [static]

Returns true if zlib library in use can handle gzip compressed data.

bool wxZlibOutputStream::SetDictionary (const char * *data,* const size_t *datalen*)

Sets the dictionary to the specified chunk of data.

This can improve compression rate but note that the dictionary has to be the same when you deflate the data as when you inflate the data, otherwise you will inflate corrupted data.

Returns true if the dictionary was successfully set.

bool wxZlibOutputStream::SetDictionary (const wxMemoryBuffer & *buf*)

Sets the dictionary to the specified chunk of data.

This can improve compression rate but note that the dictionary has to be the same when you deflate the data as when you inflate the data, otherwise you will inflate corrupted data.

Returns true if the dictionary was successfully set.

Chapter 22

File Documentation

22.75 docs/doxygen/mainpages/constants.h File Reference

22.76 docs/doxygen/mainpages/copyright.h File Reference

22.77 docs/doxygen/mainpages/devtips.h File Reference

22.78 docs/doxygen/mainpages/introduction.h File Reference

22.79 docs/doxygen/mainpages/libs.h File Reference

22.80 docs/doxygen/mainpages/manual.h File Reference

22.81 docs/doxygen/mainpages/platdetails.h File Reference

22.82 docs/doxygen/mainpages/samples.h File Reference

22.83 docs/doxygen/mainpages/screenshots.h File Reference

22.84 docs/doxygen/mainpages/topics.h File Reference

22.85 docs/doxygen/mainpages/translations.h File Reference

22.86 docs/doxygen/mainpages/utilities.h File Reference

22.87 docs/doxygen/overviews/app.h File Reference

22.88 interface/wx/app.h File Reference

Classes

- class wxAppConsole

  This class is essential for writing console-only or hybrid apps without having to define `wxUSE_GUI=0`.
- class wxApp

  The wxApp class represents the application itself when `wxUSE_GUI=1`.

Macros

- #define wxDECLARE_APP(className)

  This is used in headers to create a forward declaration of the wxGetApp() function implemented by wxIMPLEMENT_APP().
- #define wxIMPLEMENT_APP(className)

  This is used in the application class implementation file to make the application class known to wxWidgets for dynamic construction.
- #define wxDISABLE_DEBUG_SUPPORT()

  Use this macro to disable all debugging code in release build when not using wxIMPLEMENT_APP().

Functions

- wxAppDerivedClass & wxGetApp ()

  This function doesn't exist in wxWidgets but it is created by using the wxIMPLEMENT_APP() macro.
- bool wxHandleFatalExceptions (bool doIt=true)

  If doIt is true, the fatal exceptions (also known as general protection faults under Windows or segmentation violations in the Unix world) will be caught and passed to wxApp::OnFatalException.
- bool wxInitialize ()

  This function is used in wxBase only and only if you don't create wxApp object at all.
- void wxUninitialize ()

  This function is for use in console (wxBase) programs only.
- void wxWakeUpIdle ()

  This function wakes up the (internal and platform dependent) idle system, i.e.
- bool wxYield ()

  Calls wxAppConsole::Yield.
- bool wxSafeYield (wxWindow *win=NULL, bool onlyIfNeeded=false)

  Calls wxApp::SafeYield.
- int wxEntry (int &argc, wxChar **argv)

  This function initializes wxWidgets in a platform-dependent way.
- int wxEntry (HINSTANCE hInstance, HINSTANCE hPrevInstance=NULL, char *pCmdLine=NULL, int nCmdShow=SW_SHOWNORMAL)

  *See wxEntry(int&,wxChar**) for more info about this function.*
- void wxExit ()

  Exits application after calling wxApp::OnExit.

Variables

- wxApp * wxTheApp

  The global pointer to the singleton wxApp object.

22.88.1 Variable Documentation

wxApp* wxTheApp

The global pointer to the singleton wxApp object.

See also

wxApp::GetInstance()

22.89 docs/doxygen/overviews/archive.h File Reference

22.90 interface/wx/archive.h File Reference

Classes

- class wxArchiveInputStream

  This is an abstract base class which serves as a common interface to archive input streams such as wxZipInputStream.
- class wxArchiveOutputStream

This is an abstract base class which serves as a common interface to archive output streams such as wxZipOutputStream.

- class wxArchiveEntry

  This is an abstract base class which serves as a common interface to archive entry classes such as wxZipEntry.

- class wxArchiveClassFactory

  Allows the creation of streams to handle archive formats such as zip and tar.

- class wxArchiveNotifier

  If you need to know when a wxArchiveInputStream updates a wxArchiveEntry object, you can create a notifier by deriving from this abstract base class, overriding wxArchiveNotifier::OnEntryUpdated.

- class wxArchiveIterator

  An input iterator template class that can be used to transfer an archive's catalogue to a container.

22.91 docs/doxygen/overviews/aui.h File Reference

22.92 docs/doxygen/overviews/backwardcompatibility.h File Reference

22.93 docs/doxygen/overviews/bitmap.h File Reference

22.94 interface/wx/bitmap.h File Reference

Classes

- class wxBitmapHandler

  This is the base class for implementing bitmap file loading/saving, and bitmap creation from data.

- class wxBitmap

  This class encapsulates the concept of a platform-dependent bitmap, either monochrome or colour or colour with alpha channel support.

- class wxMask

  This class encapsulates a monochrome mask bitmap, where the masked area is black and the unmasked area is white.

Macros

- #define wxBITMAP_SCREEN_DEPTH (-1)

  In wxBitmap and wxBitmapHandler context this value means: "use the screen depth".

Variables

- wxBitmap wxNullBitmap

  An empty wxBitmap object.

22.94.1 Macro Definition Documentation

#define wxBITMAP_SCREEN_DEPTH (-1)

In wxBitmap and wxBitmapHandler context this value means: "use the screen depth".

22.94.2 Variable Documentation

wxBitmap wxNullBitmap

An empty wxBitmap object.

22.95 docs/doxygen/overviews/bookctrl.h File Reference

22.96 interface/wx/bookctrl.h File Reference

Classes

- class wxBookCtrlBase

  A book control is a convenient way of displaying multiple pages of information, displayed one page at a time.
- class wxBookCtrlEvent

  This class represents the events generated by book controls (wxNotebook, wxListbook, wxChoicebook, wxTreebook, wxAuiNotebook).

Macros

- #define wxBK_DEFAULT 0x0000

  wxBookCtrl flags (common for wxNotebook, wxListbook, wxChoicebook, wxTreebook)
- #define wxBK_TOP 0x0010
- #define wxBK_BOTTOM 0x0020
- #define wxBK_LEFT 0x0040
- #define wxBK_RIGHT 0x0080
- #define wxBK_ALIGN_MASK (wxBK_TOP | wxBK_BOTTOM | wxBK_LEFT | wxBK_RIGHT)
- #define wxBookCtrl TheBestBookCtrlForTheCurrentPlatform

  wxBookCtrl is defined to one of the 'real' book controls.

Enumerations

- enum {
  wxBK_HITTEST_NOWHERE = 1,
  wxBK_HITTEST_ONICON = 2,
  wxBK_HITTEST_ONLABEL = 4,
  wxBK_HITTEST_ONITEM = wxBK_HITTEST_ONICON | wxBK_HITTEST_ONLABEL,
  wxBK_HITTEST_ONPAGE = 8 }

  Bit flags returned by wxBookCtrl::HitTest().

22.96.1 Macro Definition Documentation

#define wxBK_ALIGN_MASK (wxBK_TOP | wxBK_BOTTOM | wxBK_LEFT | wxBK_RIGHT)

#define wxBK_BOTTOM 0x0020

#define wxBK_DEFAULT 0x0000

wxBookCtrl flags (common for wxNotebook, wxListbook, wxChoicebook, wxTreebook)

#define wxBK_LEFT 0x0040

#define wxBK_RIGHT 0x0080

#define wxBK_TOP 0x0010

#define wxBookCtrl TheBestBookCtrlForTheCurrentPlatform

wxBookCtrl is defined to one of the 'real' book controls.

See wxBookCtrl Overview for more info.

22.96.2 Enumeration Type Documentation

anonymous enum

Bit flags returned by wxBookCtrl::HitTest().

Notice that wxOSX currently only returns wxBK_HITTEST_ONLABEL or wxBK_HITTEST_NOWHERE and never the other values, so you should only test for these two in the code that should be portable under OS X.

Enumerator

wxBK_HITTEST_NOWHERE No tab at the specified point.

wxBK_HITTEST_ONICON The point is over an icon.

wxBK_HITTEST_ONLABEL The point is over a tab label.

wxBK_HITTEST_ONITEM The point if over a tab item but not over its icon or label.

wxBK_HITTEST_ONPAGE The point is over the page area.

22.97 interface/wx/persist/bookctrl.h File Reference

Classes

- class wxPersistentBookCtrl

  Persistence adapter for wxBookCtrlBase.

Functions

- wxPersistentObject * wxCreatePersistentObject (wxBookCtrlBase *book)

  Overload allowing persistence adapter creation for wxBookCtrlBase-derived objects.

22.97.1 Function Documentation

wxPersistentObject* wxCreatePersistentObject (wxBookCtrlBase * *book*)

Overload allowing persistence adapter creation for wxBookCtrlBase-derived objects.

22.98 docs/doxygen/overviews/bufferclasses.h File Reference

22.99 docs/doxygen/overviews/changes_since28.h File Reference

22.100 docs/doxygen/overviews/commondialogs.h File Reference

22.101 docs/doxygen/overviews/config.h File Reference

22.102 interface/wx/config.h File Reference

Classes

- class wxConfigBase

  wxConfigBase defines the basic interface of all config classes.
- class wxConfigPathChanger

  A handy little class which changes the current path in a wxConfig object and restores it in dtor.

Enumerations

- enum {
  wxCONFIG_USE_LOCAL_FILE = 1,
  wxCONFIG_USE_GLOBAL_FILE = 2,
  wxCONFIG_USE_RELATIVE_PATH = 4,
  wxCONFIG_USE_NO_ESCAPE_CHARACTERS = 8,
  wxCONFIG_USE_SUBDIR = 16 }

22.102.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxCONFIG_USE_LOCAL_FILE

wxCONFIG_USE_GLOBAL_FILE

wxCONFIG_USE_RELATIVE_PATH

wxCONFIG_USE_NO_ESCAPE_CHARACTERS

wxCONFIG_USE_SUBDIR

22.103 docs/doxygen/overviews/container.h File Reference

22.104 docs/doxygen/overviews/cpprttidisabled.h File Reference

22.105 docs/doxygen/overviews/customwidgets.h File Reference

22.106 docs/doxygen/overviews/dataobject.h File Reference

22.107 docs/doxygen/overviews/datetime.h File Reference

22.108 interface/wx/datetime.h File Reference

Classes

- class wxDateTime

  wxDateTime class represents an absolute moment in time.
- class wxDateTime::TimeZone

  Class representing a time zone.
- struct wxDateTime::Tm

  Contains broken down date-time representation.
- class wxDateTimeWorkDays
- class wxDateSpan

  This class is a "logical time span" and is useful for implementing program logic for such things as "add one month to the date" which, in general, doesn't mean to add 60∗60∗24∗31 seconds to it, but to take the same date the next month (to understand that this is indeed different consider adding one month to Feb, 15 – we want to get Mar, 15, of course).
- class wxTimeSpan

  wxTimeSpan class represents a time interval.
- class wxDateTimeHolidayAuthority

Macros

- #define wxInvalidDateTime wxDefaultDateTime

Variables

- const wxDateTime wxDefaultDateTime

  Global instance of an empty wxDateTime object.

22.108.1 Macro Definition Documentation

#define wxInvalidDateTime wxDefaultDateTime

22.108.2 Variable Documentation

const wxDateTime wxDefaultDateTime

Global instance of an empty wxDateTime object.

Todo Would it be better to rename this wxNullDateTime so it's consistent with the rest of the "empty/invalid/null" global objects?

22.109 docs/doxygen/overviews/dc.h File Reference

22.110 interface/wx/dc.h File Reference

Classes

- struct wxFontMetrics

  Simple collection of various font metrics.
- class wxDC

  A wxDC is a "device context" onto which graphics and text can be drawn.

- class wxDCClipper

  wxDCClipper is a helper class for setting a clipping region on a wxDC during its lifetime.
- class wxDCBrushChanger

  wxDCBrushChanger is a small helper class for setting a brush on a wxDC and unsetting it automatically in the destructor, restoring the previous one.
- class wxDCPenChanger

  wxDCPenChanger is a small helper class for setting a pen on a wxDC and unsetting it automatically in the destructor, restoring the previous one.
- class wxDCTextColourChanger

  wxDCTextColourChanger is a small helper class for setting a foreground text colour on a wxDC and unsetting it automatically in the destructor, restoring the previous one.
- class wxDCFontChanger

  wxDCFontChanger is a small helper class for setting a font on a wxDC and unsetting it automatically in the destructor, restoring the previous one.

Enumerations

- enum wxRasterOperationMode {
  wxCLEAR,
  wxXOR,
  wxINVERT,
  wxOR_REVERSE,
  wxAND_REVERSE,
  wxCOPY,
  wxAND,
  wxAND_INVERT,
  wxNO_OP,
  wxNOR,
  wxEQUIV,
  wxSRC_INVERT,
  wxOR_INVERT,
  wxNAND,
  wxOR,
  wxSET }

  Logical raster operations which can be used with wxDC::SetLogicalFunction and some other wxDC functions (e.g.
- enum wxFloodFillStyle {
  wxFLOOD_SURFACE = 1,
  wxFLOOD_BORDER }

  Flood styles used by wxDC::FloodFill.
- enum wxMappingMode {
  wxMM_TEXT = 1,
  wxMM_METRIC,
  wxMM_LOMETRIC,
  wxMM_TWIPS,
  wxMM_POINTS }

  The mapping used to transform logical units to device units.

22.110.1 Enumeration Type Documentation

enum wxFloodFillStyle

Flood styles used by wxDC::FloodFill.

Enumerator

wxFLOOD_SURFACE The flooding occurs until a colour other than the given colour is encountered.

wxFLOOD_BORDER The area to be flooded is bounded by the given colour.

enum wxMappingMode

The mapping used to transform *logical* units to *device* units.

See wxDC::SetMapMode.

Enumerator

wxMM_TEXT Each logical unit is 1 device pixel. This is the default mapping mode for all wxDC-derived classes.

wxMM_METRIC Each logical unit is 1 millimeter.

wxMM_LOMETRIC Each logical unit is 1/10 of a millimeter.

wxMM_TWIPS Each logical unit is 1/20 of a *"printer point"*, or 1/1440 of an inch (also known as "twip"). Equivalent to about 17.64 micrometers.

wxMM_POINTS Each logical unit is a *"printer point"* i.e. 1/72 of an inch. Equivalent to about 353 micrometers.

enum wxRasterOperationMode

Logical raster operations which can be used with wxDC::SetLogicalFunction and some other wxDC functions (e.g. wxDC::Blit and wxDC::StretchBlit).

The description of the values below refer to how a generic *src* source pixel and the corresponding *dst* destination pixel gets combined together to produce the final pixel. E.g. `wxCLEAR` and `wxSET` completely ignore the source and the destination pixel and always put zeroes or ones in the final surface.

Enumerator

wxCLEAR 0

wxXOR *src* XOR *dst*

wxINVERT NOT *dst*.

wxOR_REVERSE *src* OR (NOT *dst*)

wxAND_REVERSE *src* AND (NOT *dst*)

wxCOPY *src*

wxAND *src* AND *dst*

wxAND_INVERT (NOT *src*) AND *dst*

wxNO_OP *dst*

wxNOR (NOT *src*) AND (NOT *dst*)

wxEQUIV (NOT *src*) XOR *dst*

wxSRC_INVERT (NOT *src*)

wxOR_INVERT (NOT *src*) OR *dst*

wxNAND (NOT *src*) OR (NOT *dst*)

wxOR *src* OR *dst*

wxSET 1

22.111 docs/doxygen/overviews/debugging.h File Reference

22.112 docs/doxygen/overviews/dialog.h File Reference

22.113 interface/wx/dialog.h File Reference

Classes

- class wxDialog

  A dialog box is a window with a title bar and sometimes a system menu, which can be moved around the screen.
- class wxDialogLayoutAdapter

  This abstract class is the base for classes that help wxWidgets perform run-time layout adaptation of dialogs.
- class wxWindowModalDialogEvent

  Event sent by wxDialog::ShowWindowModal() when the dialog closes.

Macros

- #define wxDIALOG_NO_PARENT 0x00000020

  Don't make owned by apps top window.
- #define wxDEFAULT_DIALOG_STYLE (wxCAPTION | wxSYSTEM_MENU | wxCLOSE_BOX)
- #define wxDIALOG_ADAPTATION_NONE 0

  Don't do any layout adaptation.
- #define wxDIALOG_ADAPTATION_STANDARD_SIZER 1

  Only look for wxStdDialogButtonSizer for non-scrolling part.
- #define wxDIALOG_ADAPTATION_ANY_SIZER 2

  Also look for any suitable sizer for non-scrolling part.
- #define wxDIALOG_ADAPTATION_LOOSE_BUTTONS 3

  Also look for 'loose' standard buttons for non-scrolling part.

Enumerations

- enum wxDialogLayoutAdaptationMode {
  wxDIALOG_ADAPTATION_MODE_DEFAULT = 0,
  wxDIALOG_ADAPTATION_MODE_ENABLED = 1,
  wxDIALOG_ADAPTATION_MODE_DISABLED = 2 }

  Modes used for wxDialog::SetLayoutAdaptationMode().

22.113.1 Macro Definition Documentation

#define wxDEFAULT_DIALOG_STYLE (wxCAPTION | wxSYSTEM_MENU | wxCLOSE_BOX)

#define wxDIALOG_ADAPTATION_ANY_SIZER 2

Also look for any suitable sizer for non-scrolling part.

#define wxDIALOG_ADAPTATION_LOOSE_BUTTONS 3

Also look for 'loose' standard buttons for non-scrolling part.

#define wxDIALOG_ADAPTATION_NONE 0

Don't do any layout adaptation.

#define wxDIALOG_ADAPTATION_STANDARD_SIZER 1

Only look for wxStdDialogButtonSizer for non-scrolling part.

#define wxDIALOG_NO_PARENT 0x00000020

Don't make owned by apps top window.

22.113.2 Enumeration Type Documentation

enum wxDialogLayoutAdaptationMode

Modes used for wxDialog::SetLayoutAdaptationMode().

Enumerator

wxDIALOG_ADAPTATION_MODE_DEFAULT Use global adaptation enabled status.

wxDIALOG_ADAPTATION_MODE_ENABLED Enable this dialog overriding global status.

wxDIALOG_ADAPTATION_MODE_DISABLED Disable this dialog overriding global status.

22.114 docs/doxygen/overviews/dnd.h File Reference

22.115 interface/wx/dnd.h File Reference

Classes

- class wxDropTarget
  This class represents a target for a drag and drop operation.
- class wxDropSource
  This class represents a source for a drag and drop operation.
- class wxTextDropTarget
  A predefined drop target for dealing with text data.
- class wxFileDropTarget
  This is a drop target which accepts files (dragged from File Manager or Explorer).

Macros

- #define wxDROP_ICON(name)
  This macro creates either a cursor (MSW) or an icon (elsewhere) with the given name (of type `const char*`).

Enumerations

- enum {
  wxDrag_CopyOnly = 0,
  wxDrag_AllowMove = 1,
  wxDrag_DefaultMove = 3 }

Possible flags for drag and drop operations.

- enum wxDragResult {
  wxDragError,
  wxDragNone,
  wxDragCopy,
  wxDragMove,
  wxDragLink,
  wxDragCancel }

  Result returned from a wxDropSource::DoDragDrop() call.

Functions

- bool wxIsDragResultOk (wxDragResult res)

  Returns true if res indicates that something was done during a DnD operation, i.e.

22.115.1 Enumeration Type Documentation

anonymous enum

Possible flags for drag and drop operations.

Enumerator

wxDrag_CopyOnly Allow only copying.

wxDrag_AllowMove Allow moving too (copying is always allowed).

wxDrag_DefaultMove Allow moving and make it default operation.

enum wxDragResult

Result returned from a wxDropSource::DoDragDrop() call.

Enumerator

wxDragError Error prevented the D&D operation from completing.

wxDragNone Drag target didn't accept the data.

wxDragCopy The data was successfully copied.

wxDragMove The data was successfully moved (MSW only).

wxDragLink Operation is a drag-link.

wxDragCancel The operation was cancelled by user (not an error).

22.116 docs/doxygen/overviews/docview.h File Reference

22.117 interface/wx/docview.h File Reference

Classes

- class wxDocTemplate

  The wxDocTemplate class is used to model the relationship between a document class and a view class.
- class wxDocManager

The wxDocManager class is part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocument and wxDocTemplate classes.

- class wxView

  The view class can be used to model the viewing and editing component of an application's file-based data.
- class wxDocChildFrame

  The wxDocChildFrame class provides a default frame for displaying documents on separate windows.
- class wxDocParentFrame

  The wxDocParentFrame class provides a default top-level frame for applications using the document/view framework.
- class wxDocument

  The document class can be used to model an application's file-based data.

Typedefs

- typedef wxVector< wxDocument * > wxDocVector

  A vector of wxDocument pointers.
- typedef wxVector< wxView * > wxViewVector

  A vector of wxView pointers.
- typedef wxVector< wxDocTemplate * > wxDocTemplateVector

  A vector of wxDocTemplate pointers.

Functions

- bool wxTransferFileToStream (const wxString &filename, ostream &stream)

  Copies the given file to stream.
- bool wxTransferStreamToFile (istream &stream, const wxString &filename)

  Copies the given stream to the file filename.

22.117.1 Typedef Documentation

typedef wxVector<wxDocTemplate*> wxDocTemplateVector

A vector of wxDocTemplate pointers.

Since

2.9.5

typedef wxVector<wxDocument*> wxDocVector

A vector of wxDocument pointers.

Since

2.9.5

typedef wxVector<wxView*> wxViewVector

A vector of wxView pointers.

Since

2.9.5

22.118 docs/doxygen/overviews/envvars.h File Reference

22.119 docs/doxygen/overviews/eventhandling.h File Reference

22.120 docs/doxygen/overviews/exceptions.h File Reference

22.121 docs/doxygen/overviews/file.h File Reference

22.122 interface/wx/file.h File Reference

Classes

- class wxTempFile

  wxTempFile provides a relatively safe way to replace the contents of the existing file.
- class wxFile

  A wxFile performs raw file I/O.

22.123 docs/doxygen/overviews/filesystem.h File Reference

22.124 docs/doxygen/overviews/font.h File Reference

22.125 interface/wx/font.h File Reference

Classes

- class wxFontInfo

  This class is a helper used for wxFont creation using named parameter idiom: it allows to specify various wxFont attributes using the chained calls to its clearly named methods instead of passing them in the fixed order to wxFont constructors.
- class wxFont

  A font is an object which determines the appearance of text.
- class wxFontList

  A font list is a list containing all fonts which have been created.

Enumerations

- enum wxFontFamily {
  wxFONTFAMILY_DEFAULT = wxDEFAULT,
  wxFONTFAMILY_DECORATIVE = wxDECORATIVE,
  wxFONTFAMILY_ROMAN = wxROMAN,
  wxFONTFAMILY_SCRIPT = wxSCRIPT,
  wxFONTFAMILY_SWISS = wxSWISS,
  wxFONTFAMILY_MODERN = wxMODERN,
  wxFONTFAMILY_TELETYPE = wxTELETYPE,
  wxFONTFAMILY_MAX,
  wxFONTFAMILY_UNKNOWN = wxFONTFAMILY_MAX }

  Standard font families: these are used mainly during wxFont creation to specify the generic properties of the font without hardcoding in the sources a specific face name.

- enum wxFontStyle {
  wxFONTSTYLE_NORMAL = wxNORMAL,
  wxFONTSTYLE_ITALIC = wxITALIC,
  wxFONTSTYLE_SLANT = wxSLANT,
  wxFONTSTYLE_MAX }

  Font styles.

- enum wxFontWeight {
  wxFONTWEIGHT_NORMAL = wxNORMAL,
  wxFONTWEIGHT_LIGHT = wxLIGHT,
  wxFONTWEIGHT_BOLD = wxBOLD,
  wxFONTWEIGHT_MAX }

  Font weights.

- enum wxFontSymbolicSize {
  wxFONTSIZE_XX_SMALL = -3,
  wxFONTSIZE_X_SMALL,
  wxFONTSIZE_SMALL,
  wxFONTSIZE_MEDIUM,
  wxFONTSIZE_LARGE,
  wxFONTSIZE_X_LARGE,
  wxFONTSIZE_XX_LARGE }

  Symbolic font sizes.

- enum wxFontFlag {
  wxFONTFLAG_DEFAULT = 0,
  wxFONTFLAG_ITALIC = 1 << 0,
  wxFONTFLAG_SLANT = 1 << 1,
  wxFONTFLAG_LIGHT = 1 << 2,
  wxFONTFLAG_BOLD = 1 << 3,
  wxFONTFLAG_ANTIALIASED = 1 << 4,
  wxFONTFLAG_NOT_ANTIALIASED = 1 << 5,
  wxFONTFLAG_UNDERLINED = 1 << 6,
  wxFONTFLAG_STRIKETHROUGH = 1 << 7,
  wxFONTFLAG_MASK }

  The font flag bits for the new font ctor accepting one combined flags word.

- enum wxFontEncoding {
  wxFONTENCODING_SYSTEM = -1,
  wxFONTENCODING_DEFAULT,
  wxFONTENCODING_ISO8859_1,
  wxFONTENCODING_ISO8859_2,
  wxFONTENCODING_ISO8859_3,
  wxFONTENCODING_ISO8859_4,
  wxFONTENCODING_ISO8859_5,
  wxFONTENCODING_ISO8859_6,
  wxFONTENCODING_ISO8859_7,
  wxFONTENCODING_ISO8859_8,
  wxFONTENCODING_ISO8859_9,
  wxFONTENCODING_ISO8859_10,
  wxFONTENCODING_ISO8859_11,
  wxFONTENCODING_ISO8859_12,
  wxFONTENCODING_ISO8859_13,
  wxFONTENCODING_ISO8859_14,
  wxFONTENCODING_ISO8859_15,
  wxFONTENCODING_ISO8859_MAX,
  wxFONTENCODING_KOI8,
  wxFONTENCODING_KOI8_U,
  wxFONTENCODING_ALTERNATIVE,
  wxFONTENCODING_BULGARIAN,
  wxFONTENCODING_CP437,
  wxFONTENCODING_CP850,
  wxFONTENCODING_CP852,
  wxFONTENCODING_CP855,
  wxFONTENCODING_CP866,
  wxFONTENCODING_CP874,
  wxFONTENCODING_CP932,
  wxFONTENCODING_CP936,
  wxFONTENCODING_CP949,
  wxFONTENCODING_CP950,
  wxFONTENCODING_CP1250,
  wxFONTENCODING_CP1251,
  wxFONTENCODING_CP1252,
  wxFONTENCODING_CP1253,
  wxFONTENCODING_CP1254,
  wxFONTENCODING_CP1255,
  wxFONTENCODING_CP1256,
  wxFONTENCODING_CP1257,
  wxFONTENCODING_CP1258,
  wxFONTENCODING_CP1361,
  wxFONTENCODING_CP12_MAX,
  wxFONTENCODING_UTF7,
  wxFONTENCODING_UTF8,
  wxFONTENCODING_EUC_JP,
  wxFONTENCODING_UTF16BE,
  wxFONTENCODING_UTF16LE,
  wxFONTENCODING_UTF32BE,
  wxFONTENCODING_UTF32LE,
  wxFONTENCODING_MACROMAN,
  wxFONTENCODING_MACJAPANESE,
  wxFONTENCODING_MACCHINESETRAD,
  wxFONTENCODING_MACKOREAN,
  wxFONTENCODING_MACARABIC,
  wxFONTENCODING_MACHEBREW,
  wxFONTENCODING_MACGREEK,
  wxFONTENCODING_MACCYRILLIC,
  wxFONTENCODING_MACDEVANAGARI,
  wxFONTENCODING_MACGURMUKHI,
  wxFONTENCODING_MACGUJARATI,
  wxFONTENCODING_MACORIYA,
  wxFONTENCODING_MACBENGALI,

Font encodings.

Functions

- bool wxFromString (const wxString &string, wxFont *font)

  Converts string to a wxFont best represented by the given string.
- wxString wxToString (const wxFont &font)

  Converts the given wxFont into a string.

Variables

- wxFont wxNullFont

  An empty wxFont.
- wxFont * wxNORMAL_FONT

  Equivalent to wxSystemSettings::GetFont(wxSYS_DEFAULT_GUI_FONT).
- wxFont * wxSMALL_FONT

  A font using the `wxFONTFAMILY_SWISS` family and 2 points smaller than wxNORMAL_FONT.
- wxFont * wxITALIC_FONT

  A font using the `wxFONTFAMILY_ROMAN` family and `wxFONTSTYLE_ITALIC` style and of the same size of wxNORMAL_FONT.
- wxFont * wxSWISS_FONT

  A font identic to wxNORMAL_FONT except for the family used which is `wxFONTFAMILY_SWISS`.
- wxFontList * wxTheFontList

  The global wxFontList instance.

22.125.1 Enumeration Type Documentation

enum wxFontEncoding

Font encodings.

See wxFont::SetEncoding().

Enumerator

wxFONTENCODING_SYSTEM Default system encoding. Default system encoding.

wxFONTENCODING_DEFAULT Default application encoding: this is the encoding set by calls to wxFont::SetDefaultEncoding(). Initially, the default application encoding is the same as default system encoding.

wxFONTENCODING_ISO8859_1 West European (Latin1)

wxFONTENCODING_ISO8859_2 Central and East European (Latin2)

wxFONTENCODING_ISO8859_3 Esperanto (Latin3)

wxFONTENCODING_ISO8859_4 Baltic (old) (Latin4)

wxFONTENCODING_ISO8859_5 Cyrillic.

wxFONTENCODING_ISO8859_6 Arabic.

wxFONTENCODING_ISO8859_7 Greek.

wxFONTENCODING_ISO8859_8 Hebrew.

wxFONTENCODING_ISO8859_9 Turkish (Latin5)

wxFONTENCODING_ISO8859_10 Variation of Latin4 (Latin6)

wxFONTENCODING_ISO8859_11 Thai.

wxFONTENCODING_ISO8859_12 doesn't exist currently, but put it here anyhow to make all ISO8859 consecutive numbers

wxFONTENCODING_ISO8859_13 Baltic (Latin7)

wxFONTENCODING_ISO8859_14 Latin8.

wxFONTENCODING_ISO8859_15 Latin9 (a.k.a. Latin0, includes euro)

wxFONTENCODING_ISO8859_MAX

wxFONTENCODING_KOI8 KOI8 Russian.

wxFONTENCODING_KOI8_U KOI8 Ukrainian.

wxFONTENCODING_ALTERNATIVE same as MS-DOS CP866

wxFONTENCODING_BULGARIAN used under Linux in Bulgaria

wxFONTENCODING_CP437 original MS-DOS codepage

wxFONTENCODING_CP850 CP437 merged with Latin1.

wxFONTENCODING_CP852 CP437 merged with Latin2.

wxFONTENCODING_CP855 another cyrillic encoding

wxFONTENCODING_CP866 and another one

wxFONTENCODING_CP874 WinThai.

wxFONTENCODING_CP932 Japanese (shift-JIS)

wxFONTENCODING_CP936 Chinese simplified (GB)

wxFONTENCODING_CP949 Korean (Hangul charset)

wxFONTENCODING_CP950 Chinese (traditional - Big5)

wxFONTENCODING_CP1250 WinLatin2.

wxFONTENCODING_CP1251 WinCyrillic.

wxFONTENCODING_CP1252 WinLatin1.

wxFONTENCODING_CP1253 WinGreek (8859-7)

wxFONTENCODING_CP1254 WinTurkish.

wxFONTENCODING_CP1255 WinHebrew.

wxFONTENCODING_CP1256 WinArabic.

wxFONTENCODING_CP1257 WinBaltic (same as Latin 7)

wxFONTENCODING_CP1258 WinVietnamese (since 2.9.4)

wxFONTENCODING_CP1361 Johab Korean character set (since 2.9.4)

wxFONTENCODING_CP12_MAX

wxFONTENCODING_UTF7 UTF-7 Unicode encoding.

wxFONTENCODING_UTF8 UTF-8 Unicode encoding.

wxFONTENCODING_EUC_JP Extended Unix Codepage for Japanese.

wxFONTENCODING_UTF16BE UTF-16 Big Endian Unicode encoding.

wxFONTENCODING_UTF16LE UTF-16 Little Endian Unicode encoding.

wxFONTENCODING_UTF32BE UTF-32 Big Endian Unicode encoding.

wxFONTENCODING_UTF32LE

wxFONTENCODING_MACROMAN the standard mac encodings

wxFONTENCODING_MACJAPANESE

wxFONTENCODING_MACCHINESETRAD

wxFONTENCODING_MACKOREAN

wxFONTENCODING_MACARABIC

wxFONTENCODING_MACHEBREW

wxFONTENCODING_MACGREEK

wxFONTENCODING_MACCYRILLIC
wxFONTENCODING_MACDEVANAGARI
wxFONTENCODING_MACGURMUKHI
wxFONTENCODING_MACGUJARATI
wxFONTENCODING_MACORIYA
wxFONTENCODING_MACBENGALI
wxFONTENCODING_MACTAMIL
wxFONTENCODING_MACTELUGU
wxFONTENCODING_MACKANNADA
wxFONTENCODING_MACMALAJALAM
wxFONTENCODING_MACSINHALESE
wxFONTENCODING_MACBURMESE
wxFONTENCODING_MACKHMER
wxFONTENCODING_MACTHAI
wxFONTENCODING_MACLAOTIAN
wxFONTENCODING_MACGEORGIAN
wxFONTENCODING_MACARMENIAN
wxFONTENCODING_MACCHINESESIMP
wxFONTENCODING_MACTIBETAN
wxFONTENCODING_MACMONGOLIAN
wxFONTENCODING_MACETHIOPIC
wxFONTENCODING_MACCENTRALEUR
wxFONTENCODING_MACVIATNAMESE
wxFONTENCODING_MACARABICEXT
wxFONTENCODING_MACSYMBOL
wxFONTENCODING_MACDINGBATS
wxFONTENCODING_MACTURKISH
wxFONTENCODING_MACCROATIAN
wxFONTENCODING_MACICELANDIC
wxFONTENCODING_MACROMANIAN
wxFONTENCODING_MACCELTIC
wxFONTENCODING_MACGAELIC
wxFONTENCODING_MACKEYBOARD
wxFONTENCODING_ISO2022_JP ISO-2022-JP JIS encoding.
wxFONTENCODING_MAX highest enumerated encoding value
wxFONTENCODING_MACMIN
wxFONTENCODING_MACMAX
wxFONTENCODING_UTF16 native UTF-16
wxFONTENCODING_UTF32 native UTF-32
wxFONTENCODING_UNICODE Alias for the native Unicode encoding on this platform (this is used by wxEncodingConverter and wxUTFFile only for now)
wxFONTENCODING_GB2312 Simplified Chinese.
wxFONTENCODING_BIG5 Traditional Chinese.
wxFONTENCODING_SHIFT_JIS Shift JIS.
wxFONTENCODING_EUC_KR Korean.
wxFONTENCODING_JOHAB Korean Johab (since 2.9.4)
wxFONTENCODING_VIETNAMESE Vietnamese (since 2.9.4)

enum wxFontFamily

Standard font families: these are used mainly during wxFont creation to specify the generic properties of the font without hardcoding in the sources a specific face name.

wxFontFamily thus allows to group the font face names of fonts with similar properties. Most wxWidgets ports use lists of fonts for each font family inspired by the data taken from http://www.codestyle.org/css/font-family.

Enumerator

wxFONTFAMILY_DEFAULT Chooses a default font.

wxFONTFAMILY_DECORATIVE A decorative font.

wxFONTFAMILY_ROMAN A formal, serif font.

wxFONTFAMILY_SCRIPT A handwriting font.

wxFONTFAMILY_SWISS A sans-serif font.

wxFONTFAMILY_MODERN A fixed pitch font. Note that wxFont currently does not make distinctions between wxFONTFAMILY_MODERN and wxFONTFAMILY_TELETYPE.

wxFONTFAMILY_TELETYPE A teletype (i.e. monospaced) font. Monospace fonts have a fixed width like typewriters and often have strong angular or block serifs. Monospace font faces are often used code samples and have a simple, functional font style. See also wxFont::IsFixedWidth() for an easy way to test for monospace property.

wxFONTFAMILY_MAX

wxFONTFAMILY_UNKNOWN Invalid font family value, returned by wxFont::GetFamily() when the font is invalid for example.

enum wxFontFlag

The font flag bits for the new font ctor accepting one combined flags word.

Enumerator

wxFONTFLAG_DEFAULT no special flags: font with default weight/slant/anti-aliasing

wxFONTFLAG_ITALIC slant flags (default: no slant)

wxFONTFLAG_SLANT

wxFONTFLAG_LIGHT weight flags (default: medium)

wxFONTFLAG_BOLD

wxFONTFLAG_ANTIALIASED anti-aliasing flag: force on or off (default: the current system default)

wxFONTFLAG_NOT_ANTIALIASED

wxFONTFLAG_UNDERLINED Underlined style (not underlined by default).

wxFONTFLAG_STRIKETHROUGH Strike-through style (implemented in MSW, GTK, and OSX)

wxFONTFLAG_MASK the mask of all currently used flags

enum wxFontStyle

Font styles.

Enumerator

wxFONTSTYLE_NORMAL The font is drawn without slant.

wxFONTSTYLE_ITALIC The font is slanted in an italic style.

wxFONTSTYLE_SLANT The font is slanted, but in a roman style. Note that under wxMSW this style is the same as wxFONTSTYLE_ITALIC.

wxFONTSTYLE_MAX

enum wxFontSymbolicSize

Symbolic font sizes.

The elements of this enum correspond to CSS absolute size specifications, see http://www.w3.org/TR/CSS21/fonts.html#font-size-props

See also

wxFont::SetSymbolicSize()

Since

2.9.2

Enumerator

wxFONTSIZE_XX_SMALL Extra small.
wxFONTSIZE_X_SMALL Very small.
wxFONTSIZE_SMALL Small.
wxFONTSIZE_MEDIUM Normal.
wxFONTSIZE_LARGE Large.
wxFONTSIZE_X_LARGE Very large.
wxFONTSIZE_XX_LARGE Extra large.

enum wxFontWeight

Font weights.

Enumerator

wxFONTWEIGHT_NORMAL Normal font.
wxFONTWEIGHT_LIGHT Light font.
wxFONTWEIGHT_BOLD Bold font.
wxFONTWEIGHT_MAX

22.125.2 Variable Documentation

wxFont* wxITALIC_FONT

A font using the `wxFONTFAMILY_ROMAN` family and `wxFONTSTYLE_ITALIC` style and of the same size of wxNORMAL_FONT.

wxFont* wxNORMAL_FONT

Equivalent to wxSystemSettings::GetFont(wxSYS_DEFAULT_GUI_FONT).

See also

wxSystemSettings

wxFont wxNullFont

An empty wxFont.

wxFont* wxSMALL_FONT

A font using the `wxFONTFAMILY_SWISS` family and 2 points smaller than wxNORMAL_FONT.

wxFont* wxSWISS_FONT

A font identic to wxNORMAL_FONT except for the family used which is `wxFONTFAMILY_SWISS`.

wxFontList* wxTheFontList

The global wxFontList instance.

22.126 docs/doxygen/overviews/fontencoding.h File Reference

22.127 docs/doxygen/overviews/grid.h File Reference

22.128 interface/wx/grid.h File Reference

Classes

- class wxGridCellRenderer

  This class is responsible for actually drawing the cell in the grid.
- class wxGridCellAutoWrapStringRenderer

  This class may be used to format string data in a cell.
- class wxGridCellBoolRenderer

  This class may be used to format boolean data in a cell.
- class wxGridCellDateTimeRenderer

  This class may be used to format a date/time data in a cell.
- class wxGridCellEnumRenderer

  This class may be used to render in a cell a number as a textual equivalent.
- class wxGridCellFloatRenderer

  This class may be used to format floating point data in a cell.
- class wxGridCellNumberRenderer

  This class may be used to format integer data in a cell.
- class wxGridCellStringRenderer

  This class may be used to format string data in a cell; it is the default for string cells.
- class wxGridCellEditor

  This class is responsible for providing and manipulating the in-place edit controls for the grid.
- class wxGridCellAutoWrapStringEditor

  Grid cell editor for wrappable string/text data.
- class wxGridCellBoolEditor

  Grid cell editor for boolean data.
- class wxGridCellChoiceEditor

  Grid cell editor for string data providing the user a choice from a list of strings.
- class wxGridCellEnumEditor

  Grid cell editor which displays an enum number as a textual equivalent (eg.
- class wxGridCellTextEditor

  Grid cell editor for string/text data.
- class wxGridCellFloatEditor

The editor for floating point numbers data.

- class wxGridCellNumberEditor

  Grid cell editor for numeric integer data.
- class wxGridCellAttr

  This class can be used to alter the cells' appearance in the grid by changing their attributes from the defaults.
- class wxGridCornerHeaderRenderer

  Base class for corner window renderer.
- class wxGridHeaderLabelsRenderer

  Common base class for row and column headers renderers.
- class wxGridRowHeaderRenderer

  Base class for row headers renderer.
- class wxGridColumnHeaderRenderer

  Base class for column headers renderer.
- class wxGridRowHeaderRendererDefault

  Default row header renderer.
- class wxGridColumnHeaderRendererDefault

  Default column header renderer.
- class wxGridCornerHeaderRendererDefault

  Default corner window renderer.
- class wxGridCellAttrProvider

  Class providing attributes to be used for the grid cells.
- class wxGridCellCoords

  Represents coordinates of a grid cell.
- class wxGridTableBase

  The almost abstract base class for grid tables.
- class wxGridTableMessage

  A simple class used to pass messages from the table to the grid.
- class wxGridStringTable

  Simplest type of data table for a grid for small tables of strings that are stored in memory.
- class wxGridSizesInfo

  wxGridSizesInfo stores information about sizes of all wxGrid rows or columns.
- class wxGrid

  wxGrid and its related classes are used for displaying and editing tabular data.
- class wxGridUpdateLocker

  This small class can be used to prevent wxGrid from redrawing during its lifetime by calling wxGrid::BeginBatch() in its constructor and wxGrid::EndBatch() in its destructor.
- class wxGridEvent

  This event class contains information about various grid events.
- class wxGridSizeEvent

  This event class contains information about a row/column resize event.
- class wxGridRangeSelectEvent
- class wxGridEditorCreatedEvent

Enumerations

- enum wxGridCellFloatFormat {
  wxGRID_FLOAT_FORMAT_FIXED = 0x0010,
  wxGRID_FLOAT_FORMAT_SCIENTIFIC = 0x0020,
  wxGRID_FLOAT_FORMAT_COMPACT = 0x0040,
  wxGRID_FLOAT_FORMAT_UPPER = 0x0080,
  wxGRID_FLOAT_FORMAT_DEFAULT = wxGRID_FLOAT_FORMAT_FIXED }

Specifier used to format the data to string for the numbers handled by wxGridCellFloatRenderer and wxGridCellFloatEditor.

- enum wxGridTableRequest {
  wxGRIDTABLE_REQUEST_VIEW_GET_VALUES = 2000,
  wxGRIDTABLE_REQUEST_VIEW_SEND_VALUES,
  wxGRIDTABLE_NOTIFY_ROWS_INSERTED,
  wxGRIDTABLE_NOTIFY_ROWS_APPENDED,
  wxGRIDTABLE_NOTIFY_ROWS_DELETED,
  wxGRIDTABLE_NOTIFY_COLS_INSERTED,
  wxGRIDTABLE_NOTIFY_COLS_APPENDED,
  wxGRIDTABLE_NOTIFY_COLS_DELETED }
- enum wxGridRenderStyle {
  wxGRID_DRAW_ROWS_HEADER = 0x001,
  wxGRID_DRAW_COLS_HEADER = 0x002,
  wxGRID_DRAW_CELL_LINES = 0x004,
  wxGRID_DRAW_BOX_RECT = 0x008,
  wxGRID_DRAW_SELECTION = 0x010,
  wxGRID_DRAW_DEFAULT }

  Rendering styles supported by wxGrid::Render() method.

Variables

- wxEventType wxEVT_GRID_CELL_LEFT_CLICK
- wxEventType wxEVT_GRID_CELL_RIGHT_CLICK
- wxEventType wxEVT_GRID_CELL_LEFT_DCLICK
- wxEventType wxEVT_GRID_CELL_RIGHT_DCLICK
- wxEventType wxEVT_GRID_LABEL_LEFT_CLICK
- wxEventType wxEVT_GRID_LABEL_RIGHT_CLICK
- wxEventType wxEVT_GRID_LABEL_LEFT_DCLICK
- wxEventType wxEVT_GRID_LABEL_RIGHT_DCLICK
- wxEventType wxEVT_GRID_ROW_SIZE
- wxEventType wxEVT_GRID_COL_SIZE
- wxEventType wxEVT_GRID_COL_AUTO_SIZE
- wxEventType wxEVT_GRID_RANGE_SELECT
- wxEventType wxEVT_GRID_CELL_CHANGING
- wxEventType wxEVT_GRID_CELL_CHANGED
- wxEventType wxEVT_GRID_SELECT_CELL
- wxEventType wxEVT_GRID_EDITOR_SHOWN
- wxEventType wxEVT_GRID_EDITOR_HIDDEN
- wxEventType wxEVT_GRID_EDITOR_CREATED
- wxEventType wxEVT_GRID_CELL_BEGIN_DRAG
- wxEventType wxEVT_GRID_COL_MOVE
- wxEventType wxEVT_GRID_COL_SORT
- wxEventType wxEVT_GRID_TABBING

22.128.1 Enumeration Type Documentation

enum wxGridCellFloatFormat

Specifier used to format the data to string for the numbers handled by wxGridCellFloatRenderer and wxGridCellFloatEditor.

Since

2.9.3

Enumerator

wxGRID_FLOAT_FORMAT_FIXED Decimal floating point (f).

wxGRID_FLOAT_FORMAT_SCIENTIFIC Scientific notation (mantissa/exponent) using e character (e).

wxGRID_FLOAT_FORMAT_COMPACT Use the shorter of e or f (g).

wxGRID_FLOAT_FORMAT_UPPER To use in combination with one of the above formats for the upper case version (F/E/G)

wxGRID_FLOAT_FORMAT_DEFAULT The format used by default (wxGRID_FLOAT_FORMAT_FIXED).

enum wxGridRenderStyle

Rendering styles supported by wxGrid::Render() method.

Since

2.9.4

Enumerator

wxGRID_DRAW_ROWS_HEADER Draw grid row header labels.

wxGRID_DRAW_COLS_HEADER Draw grid column header labels.

wxGRID_DRAW_CELL_LINES Draw grid cell border lines.

wxGRID_DRAW_BOX_RECT Draw a bounding rectangle around the rendered cell area. Useful where row or column headers are not drawn or where there is multi row or column cell clipping and therefore no cell border at the rendered outer boundary.

wxGRID_DRAW_SELECTION Draw the grid cell selection highlight if a selection is present. At present the highlight colour drawn depends on whether the grid window loses focus before drawing begins.

wxGRID_DRAW_DEFAULT The default render style. Includes all except wxGRID_DRAW_SELECTION.

enum wxGridTableRequest

Enumerator

wxGRIDTABLE_REQUEST_VIEW_GET_VALUES

wxGRIDTABLE_REQUEST_VIEW_SEND_VALUES

wxGRIDTABLE_NOTIFY_ROWS_INSERTED

wxGRIDTABLE_NOTIFY_ROWS_APPENDED

wxGRIDTABLE_NOTIFY_ROWS_DELETED

wxGRIDTABLE_NOTIFY_COLS_INSERTED

wxGRIDTABLE_NOTIFY_COLS_APPENDED

wxGRIDTABLE_NOTIFY_COLS_DELETED

22.128.2 Variable Documentation

wxEventType wxEVT_GRID_CELL_BEGIN_DRAG

wxEventType wxEVT_GRID_CELL_CHANGED

wxEventType wxEVT_GRID_CELL_CHANGING

wxEventType wxEVT_GRID_CELL_LEFT_CLICK

wxEventType wxEVT_GRID_CELL_LEFT_DCLICK

wxEventType wxEVT_GRID_CELL_RIGHT_CLICK

wxEventType wxEVT_GRID_CELL_RIGHT_DCLICK

wxEventType wxEVT_GRID_COL_AUTO_SIZE

wxEventType wxEVT_GRID_COL_MOVE

wxEventType wxEVT_GRID_COL_SIZE

wxEventType wxEVT_GRID_COL_SORT

wxEventType wxEVT_GRID_EDITOR_CREATED

wxEventType wxEVT_GRID_EDITOR_HIDDEN

wxEventType wxEVT_GRID_EDITOR_SHOWN

wxEventType wxEVT_GRID_LABEL_LEFT_CLICK

wxEventType wxEVT_GRID_LABEL_LEFT_DCLICK

wxEventType wxEVT_GRID_LABEL_RIGHT_CLICK

wxEventType wxEVT_GRID_LABEL_RIGHT_DCLICK

wxEventType wxEVT_GRID_RANGE_SELECT

wxEventType wxEVT_GRID_ROW_SIZE

wxEventType wxEVT_GRID_SELECT_CELL

wxEventType wxEVT_GRID_TABBING

22.129 docs/doxygen/overviews/helloworld.h File Reference

22.130 docs/doxygen/overviews/html.h File Reference

22.131 docs/doxygen/overviews/internationalization.h File Reference

22.132 docs/doxygen/overviews/ipc.h File Reference

22.133 interface/wx/ipc.h File Reference

Classes

- class wxConnection

A wxConnection object represents the connection between a client and a server.

- class wxClient

  A wxClient object represents the client part of a client-server DDE-like (Dynamic Data Exchange) conversation.

- class wxServer

  A wxServer object represents the server part of a client-server DDE-like (Dynamic Data Exchange) conversation.

22.134 docs/doxygen/overviews/listctrl.h File Reference

22.135 interface/wx/listctrl.h File Reference

Classes

- class wxListCtrl

  A list control presents lists in a number of formats: list view, report view, icon view and small icon view.

- class wxListEvent

  A list event holds information about events associated with wxListCtrl objects.

- class wxListItemAttr

  Represents the attributes (color, font, ...) of a wxListCtrl's wxListItem.

- class wxListView

  This class currently simply presents a simpler to use interface for the wxListCtrl – it can be thought of as a façade for that complicated class.

- class wxListItem

  This class stores information about a wxListCtrl item or column.

Macros

- #define wxLC_VRULES 0x0001

  style flags

- #define wxLC_HRULES 0x0002
- #define wxLC_ICON 0x0004
- #define wxLC_SMALL_ICON 0x0008
- #define wxLC_LIST 0x0010
- #define wxLC_REPORT 0x0020
- #define wxLC_ALIGN_TOP 0x0040
- #define wxLC_ALIGN_LEFT 0x0080
- #define wxLC_AUTOARRANGE 0x0100
- #define wxLC_VIRTUAL 0x0200
- #define wxLC_EDIT_LABELS 0x0400
- #define wxLC_NO_HEADER 0x0800
- #define wxLC_NO_SORT_HEADER 0x1000
- #define wxLC_SINGLE_SEL 0x2000
- #define wxLC_SORT_ASCENDING 0x4000
- #define wxLC_SORT_DESCENDING 0x8000
- #define wxLC_MASK_TYPE (wxLC_ICON | wxLC_SMALL_ICON | wxLC_LIST | wxLC_REPORT)
- #define wxLC_MASK_ALIGN (wxLC_ALIGN_TOP | wxLC_ALIGN_LEFT)
- #define wxLC_MASK_SORT (wxLC_SORT_ASCENDING | wxLC_SORT_DESCENDING)
- #define wxLIST_MASK_STATE 0x0001

  Mask flags to tell app/GUI what fields of wxListItem are valid.

- #define wxLIST_MASK_TEXT 0x0002
- #define wxLIST_MASK_IMAGE 0x0004
- #define wxLIST_MASK_DATA 0x0008

- #define wxLIST_SET_ITEM 0x0010
- #define wxLIST_MASK_WIDTH 0x0020
- #define wxLIST_MASK_FORMAT 0x0040
- #define wxLIST_STATE_DONTCARE 0x0000

  State flags for indicating the state of an item.
- #define wxLIST_STATE_DROPHILITED 0x0001
- #define wxLIST_STATE_FOCUSED 0x0002
- #define wxLIST_STATE_SELECTED 0x0004
- #define wxLIST_STATE_CUT 0x0008
- #define wxLIST_HITTEST_ABOVE 0x0001

  Hit test flags, used in HitTest.
- #define wxLIST_HITTEST_BELOW 0x0002
- #define wxLIST_HITTEST_NOWHERE 0x0004
- #define wxLIST_HITTEST_ONITEMICON 0x0020
- #define wxLIST_HITTEST_ONITEMLABEL 0x0080
- #define wxLIST_HITTEST_ONITEMRIGHT 0x0100
- #define wxLIST_HITTEST_ONITEMSTATEICON 0x0200
- #define wxLIST_HITTEST_TOLEFT 0x0400
- #define wxLIST_HITTEST_TORIGHT 0x0800
- #define wxLIST_HITTEST_ONITEM (wxLIST_HITTEST_ONITEMICON | wxLIST_HITTEST_ONITEMLAB↩EL | wxLIST_HITTEST_ONITEMSTATEICON)
- #define wxLIST_GETSUBITEMRECT_WHOLEITEM -1l

  GetSubItemRect constants.

Enumerations

- enum {
  wxLIST_NEXT_ABOVE,
  wxLIST_NEXT_ALL,
  wxLIST_NEXT_BELOW,
  wxLIST_NEXT_LEFT,
  wxLIST_NEXT_RIGHT }

  Flags for GetNextItem (MSW only except wxLIST_NEXT_ALL)
- enum {
  wxLIST_ALIGN_DEFAULT,
  wxLIST_ALIGN_LEFT,
  wxLIST_ALIGN_TOP,
  wxLIST_ALIGN_SNAP_TO_GRID }

  Alignment flags for Arrange (MSW only except wxLIST_ALIGN_LEFT)
- enum wxListColumnFormat {
  wxLIST_FORMAT_LEFT,
  wxLIST_FORMAT_RIGHT,
  wxLIST_FORMAT_CENTRE,
  wxLIST_FORMAT_CENTER = wxLIST_FORMAT_CENTRE }

  Column format (MSW only except wxLIST_FORMAT_LEFT)
- enum {
  wxLIST_AUTOSIZE = -1,
  wxLIST_AUTOSIZE_USEHEADER = -2 }

  Autosize values for SetColumnWidth.
- enum {
  wxLIST_RECT_BOUNDS,
  wxLIST_RECT_ICON,
  wxLIST_RECT_LABEL }

  Flag values for GetItemRect.

- enum {
  wxLIST_FIND_UP,
  wxLIST_FIND_DOWN,
  wxLIST_FIND_LEFT,
  wxLIST_FIND_RIGHT }

  Flag values for FindItem (MSW only)

Variables

- wxEventType wxEVT_LIST_BEGIN_DRAG
- wxEventType wxEVT_LIST_BEGIN_RDRAG
- wxEventType wxEVT_LIST_BEGIN_LABEL_EDIT
- wxEventType wxEVT_LIST_END_LABEL_EDIT
- wxEventType wxEVT_LIST_DELETE_ITEM
- wxEventType wxEVT_LIST_DELETE_ALL_ITEMS
- wxEventType wxEVT_LIST_ITEM_SELECTED
- wxEventType wxEVT_LIST_ITEM_DESELECTED
- wxEventType wxEVT_LIST_KEY_DOWN
- wxEventType wxEVT_LIST_INSERT_ITEM
- wxEventType wxEVT_LIST_COL_CLICK
- wxEventType wxEVT_LIST_ITEM_RIGHT_CLICK
- wxEventType wxEVT_LIST_ITEM_MIDDLE_CLICK
- wxEventType wxEVT_LIST_ITEM_ACTIVATED
- wxEventType wxEVT_LIST_CACHE_HINT
- wxEventType wxEVT_LIST_COL_RIGHT_CLICK
- wxEventType wxEVT_LIST_COL_BEGIN_DRAG
- wxEventType wxEVT_LIST_COL_DRAGGING
- wxEventType wxEVT_LIST_COL_END_DRAG
- wxEventType wxEVT_LIST_ITEM_FOCUSED

22.135.1 Macro Definition Documentation

#define wxLC_ALIGN_LEFT 0x0080

#define wxLC_ALIGN_TOP 0x0040

#define wxLC_AUTOARRANGE 0x0100

#define wxLC_EDIT_LABELS 0x0400

#define wxLC_HRULES 0x0002

#define wxLC_ICON 0x0004

#define wxLC_LIST 0x0010

#define wxLC_MASK_ALIGN (wxLC_ALIGN_TOP | wxLC_ALIGN_LEFT)

#define wxLC_MASK_SORT (wxLC_SORT_ASCENDING | wxLC_SORT_DESCENDING)

#define wxLC_MASK_TYPE (wxLC_ICON | wxLC_SMALL_ICON | wxLC_LIST | wxLC_REPORT)

#define wxLC_NO_HEADER 0x0800

#define wxLC_NO_SORT_HEADER 0x1000

#define wxLC_REPORT 0x0020

#define wxLC_SINGLE_SEL 0x2000

#define wxLC_SMALL_ICON 0x0008

#define wxLC_SORT_ASCENDING 0x4000

#define wxLC_SORT_DESCENDING 0x8000

#define wxLC_VIRTUAL 0x0200

#define wxLC_VRULES 0x0001

style flags

#define wxLIST_GETSUBITEMRECT_WHOLEITEM -1l

GetSubItemRect constants.

#define wxLIST_HITTEST_ABOVE 0x0001

Hit test flags, used in HitTest.

#define wxLIST_HITTEST_BELOW 0x0002

#define wxLIST_HITTEST_NOWHERE 0x0004

#define wxLIST_HITTEST_ONITEM (**wxLIST_HITTEST_ONITEMICON** | **wxLIST_HITTEST_ONITEMLABEL** | **wxLIST_HITTEST_ONITEMSTATEICON**)

#define wxLIST_HITTEST_ONITEMICON 0x0020

#define wxLIST_HITTEST_ONITEMLABEL 0x0080

#define wxLIST_HITTEST_ONITEMRIGHT 0x0100

#define wxLIST_HITTEST_ONITEMSTATEICON 0x0200

#define wxLIST_HITTEST_TOLEFT 0x0400

#define wxLIST_HITTEST_TORIGHT 0x0800

#define wxLIST_MASK_DATA 0x0008

#define wxLIST_MASK_FORMAT 0x0040

#define wxLIST_MASK_IMAGE 0x0004

#define wxLIST_MASK_STATE 0x0001

Mask flags to tell app/GUI what fields of wxListItem are valid.

#define wxLIST_MASK_TEXT 0x0002

#define wxLIST_MASK_WIDTH 0x0020

#define wxLIST_SET_ITEM 0x0010

#define wxLIST_STATE_CUT 0x0008

#define wxLIST_STATE_DONTCARE 0x0000

State flags for indicating the state of an item.

#define wxLIST_STATE_DROPHILITED 0x0001

#define wxLIST_STATE_FOCUSED 0x0002

#define wxLIST_STATE_SELECTED 0x0004

22.135.2 Enumeration Type Documentation

anonymous enum

Flags for GetNextItem (MSW only except wxLIST_NEXT_ALL)

Enumerator

wxLIST_NEXT_ABOVE

wxLIST_NEXT_ALL

wxLIST_NEXT_BELOW

wxLIST_NEXT_LEFT

wxLIST_NEXT_RIGHT

anonymous enum

Alignment flags for Arrange (MSW only except wxLIST_ALIGN_LEFT)

Enumerator

wxLIST_ALIGN_DEFAULT

wxLIST_ALIGN_LEFT

wxLIST_ALIGN_TOP

wxLIST_ALIGN_SNAP_TO_GRID

anonymous enum

Autosize values for SetColumnWidth.

Enumerator

wxLIST_AUTOSIZE

wxLIST_AUTOSIZE_USEHEADER

anonymous enum

Flag values for GetItemRect.

Enumerator

wxLIST_RECT_BOUNDS

wxLIST_RECT_ICON

wxLIST_RECT_LABEL

anonymous enum

Flag values for FindItem (MSW only)

Enumerator

wxLIST_FIND_UP

wxLIST_FIND_DOWN

wxLIST_FIND_LEFT

wxLIST_FIND_RIGHT

enum wxListColumnFormat

Column format (MSW only except wxLIST_FORMAT_LEFT)

Enumerator

wxLIST_FORMAT_LEFT

wxLIST_FORMAT_RIGHT

wxLIST_FORMAT_CENTRE

wxLIST_FORMAT_CENTER

22.135.3 Variable Documentation

wxEventType wxEVT_LIST_BEGIN_DRAG

wxEventType wxEVT_LIST_BEGIN_LABEL_EDIT

wxEventType wxEVT_LIST_BEGIN_RDRAG

wxEventType wxEVT_LIST_CACHE_HINT

wxEventType wxEVT_LIST_COL_BEGIN_DRAG

wxEventType wxEVT_LIST_COL_CLICK

wxEventType wxEVT_LIST_COL_DRAGGING

wxEventType wxEVT_LIST_COL_END_DRAG

wxEventType wxEVT_LIST_COL_RIGHT_CLICK

wxEventType wxEVT_LIST_DELETE_ALL_ITEMS

wxEventType wxEVT_LIST_DELETE_ITEM

wxEventType wxEVT_LIST_END_LABEL_EDIT

wxEventType wxEVT_LIST_INSERT_ITEM

wxEventType wxEVT_LIST_ITEM_ACTIVATED

wxEventType wxEVT_LIST_ITEM_DESELECTED

wxEventType wxEVT_LIST_ITEM_FOCUSED

wxEventType wxEVT_LIST_ITEM_MIDDLE_CLICK

wxEventType wxEVT_LIST_ITEM_RIGHT_CLICK

wxEventType wxEVT_LIST_ITEM_SELECTED

wxEventType wxEVT_LIST_KEY_DOWN

22.136 docs/doxygen/overviews/log.h File Reference

22.137 interface/wx/log.h File Reference

Classes

- class wxLogRecordInfo

  Information about a log record (unit of the log output).
- class wxLogFormatter

  wxLogFormatter class is used to format the log messages.
- class wxLog

  wxLog class defines the interface for the log targets used by wxWidgets logging functions as explained in the Logging Overview.
- class wxLogChain

  This simple class allows you to chain log sinks, that is to install a new sink but keep passing log messages to the old one instead of replacing it completely as wxLog::SetActiveTarget does.
- class wxLogInterposer

  A special version of wxLogChain which uses itself as the new log target.
- class wxLogInterposerTemp

  A special version of wxLogChain which uses itself as the new log target.
- class wxLogStream

  This class can be used to redirect the log messages to a C++ stream.
- class wxLogStderr

  This class can be used to redirect the log messages to a C file stream (not to be confused with C++ streams).
- class wxLogBuffer

  wxLogBuffer is a very simple implementation of log sink which simply collects all the logged messages in a string (except the debug messages which are output in the usual way immediately as we're presumably not interested in collecting them for later).
- class wxLogNull

  This class allows you to temporarily suspend logging.
- class wxLogWindow

  This class represents a background log window: to be precise, it collects all log messages in the log frame which it manages but also passes them on to the log target which was active at the moment of its creation.

- class wxLogGui

  This is the default log target for the GUI wxWidgets applications.
- class wxLogTextCtrl

  Using these target all the log messages can be redirected to a text control.

Macros

- #define wxDISABLE_DEBUG_LOGGING_IN_RELEASE_BUILD()

  Use this macro to disable logging at debug and trace levels in release build when not using wxIMPLEMENT_APP().

Typedefs

- typedef unsigned long wxLogLevel

  The type used to specify a log level.

Enumerations

- enum wxLogLevelValues {
  wxLOG_FatalError,
  wxLOG_Error,
  wxLOG_Warning,
  wxLOG_Message,
  wxLOG_Status,
  wxLOG_Info,
  wxLOG_Debug,
  wxLOG_Trace,
  wxLOG_Progress,
  wxLOG_User = 100,
  wxLOG_Max = 10000 }

  Different standard log levels (you may also define your own) used with by standard wxLog functions wxLogGeneric(), wxLogError(), wxLogWarning(), etc...

Functions

- void wxSafeShowMessage (const wxString &title, const wxString &text)

  This function shows a message to the user in a safe way and should be safe to call even before the application has been initialized or if it is currently in some other strange state (for example, about to crash).
- unsigned long wxSysErrorCode ()

  Returns the error code from the last system call.
- const wxChar * wxSysErrorMsg (unsigned long errCode=0)

  Returns the error message corresponding to the given system error code.
- void wxLogGeneric (wxLogLevel level, const char *formatString,...)

  Logs a message with the given wxLogLevel.
- void wxVLogGeneric (wxLogLevel level, const char *formatString, va_list argPtr)
- void wxLogMessage (const char *formatString,...)

  For all normal, informational messages.
- void wxVLogMessage (const char *formatString, va_list argPtr)
- void wxLogVerbose (const char *formatString,...)

  For verbose output.
- void wxVLogVerbose (const char *formatString, va_list argPtr)
- void wxLogWarning (const char *formatString,...)

  For warnings - they are also normally shown to the user, but don't interrupt the program work.

- void wxVLogWarning (const char *formatString, va_list argPtr)
- void wxLogFatalError (const char *formatString,...)

  Like wxLogError(), but also terminates the program with the exit code 3.
- void wxVLogFatalError (const char *formatString, va_list argPtr)
- void wxLogError (const char *formatString,...)

  The functions to use for error messages, i.e.
- void wxVLogError (const char *formatString, va_list argPtr)
- void wxLogTrace (const char *mask, const char *formatString,...)

  Log a message at `wxLOG_Trace` log level (see wxLogLevelValues enum).
- void wxVLogTrace (const char *mask, const char *formatString, va_list argPtr)
- void wxLogTrace (wxTraceMask mask, const char *formatString,...)

  Like wxLogDebug(), trace functions only do something in debug builds and expand to nothing in the release one.
- void wxVLogTrace (wxTraceMask mask, const char *formatString, va_list argPtr)
- void wxLogDebug (const char *formatString,...)

  The right functions for debug output.
- void wxVLogDebug (const char *formatString, va_list argPtr)
- void wxLogStatus (wxFrame *frame, const char *formatString,...)

  Messages logged by this function will appear in the statusbar of the frame or of the top level application window by default (i.e.
- void wxVLogStatus (wxFrame *frame, const char *formatString, va_list argPtr)
- void wxLogStatus (const char *formatString,...)
- void wxVLogStatus (const char *formatString, va_list argPtr)
- void wxLogSysError (const char *formatString,...)

  Mostly used by wxWidgets itself, but might be handy for logging errors after system call (API function) failure.
- void wxVLogSysError (const char *formatString, va_list argPtr)

22.137.1 Typedef Documentation

typedef unsigned long wxLogLevel

The type used to specify a log level.

Default values of wxLogLevel used by wxWidgets are contained in the wxLogLevelValues enumeration.

22.137.2 Enumeration Type Documentation

enum wxLogLevelValues

Different standard log levels (you may also define your own) used with by standard wxLog functions wxLogGeneric(), wxLogError(), wxLogWarning(), etc...

Enumerator

wxLOG_FatalError program can't continue, abort immediately

wxLOG_Error a serious error, user must be informed about it

wxLOG_Warning user is normally informed about it but may be ignored

wxLOG_Message normal message (i.e. normal output of a non GUI app)

wxLOG_Status informational: might go to the status line of GUI app

wxLOG_Info informational message (a.k.a. 'Verbose')

wxLOG_Debug never shown to the user, disabled in release mode

wxLOG_Trace trace messages are also only enabled in debug mode

wxLOG_Progress used for progress indicator (not yet)

wxLOG_User user defined levels start here

wxLOG_Max

22.138 interface/wx/protocol/log.h File Reference

Classes

- class wxProtocolLog

  Class allowing to log network operations performed by wxProtocol.

22.139 docs/doxygen/overviews/mbconvclasses.h File Reference

22.140 docs/doxygen/overviews/nonenglish.h File Reference

22.141 docs/doxygen/overviews/persistence.h File Reference

22.142 docs/doxygen/overviews/printing.h File Reference

22.143 docs/doxygen/overviews/propgrid.h File Reference

22.144 interface/wx/propgrid/propgrid.h File Reference

- #define wxPG_DEFAULT_STYLE (0)

  Combines various styles.
- #define wxPGMAN_DEFAULT_STYLE (0)

  Combines various styles.
- enum wxPG_WINDOW_STYLES {
  wxPG_AUTO_SORT = 0x00000010,
  wxPG_HIDE_CATEGORIES = 0x00000020,
  wxPG_ALPHABETIC_MODE = (wxPG_HIDE_CATEGORIES|wxPG_AUTO_SORT),
  wxPG_BOLD_MODIFIED = 0x00000040,
  wxPG_SPLITTER_AUTO_CENTER = 0x00000080,
  wxPG_TOOLTIPS = 0x00000100,
  wxPG_HIDE_MARGIN = 0x00000200,
  wxPG_STATIC_SPLITTER = 0x00000400,
  wxPG_STATIC_LAYOUT = (wxPG_HIDE_MARGIN|wxPG_STATIC_SPLITTER),
  wxPG_LIMITED_EDITING = 0x00000800,
  wxPG_TOOLBAR = 0x00001000,
  wxPG_DESCRIPTION = 0x00002000,
  wxPG_NO_INTERNAL_BORDER = 0x00004000 }
- enum wxPG_EX_WINDOW_STYLES {
  wxPG_EX_INIT_NOCAT = 0x00001000,
  wxPG_EX_NO_FLAT_TOOLBAR = 0x00002000,
  wxPG_EX_MODE_BUTTONS = 0x00008000,
  wxPG_EX_HELP_AS_TOOLTIPS = 0x00010000,
  wxPG_EX_NATIVE_DOUBLE_BUFFERING = 0x00080000,
  wxPG_EX_AUTO_UNSPECIFIED_VALUES = 0x00200000,
  wxPG_EX_WRITEONLY_BUILTIN_ATTRIBUTES = 0x00400000,
  wxPG_EX_HIDE_PAGE_BUTTONS = 0x01000000,
  wxPG_EX_MULTIPLE_SELECTION = 0x02000000,
  wxPG_EX_ENABLE_TLP_TRACKING = 0x04000000,
  wxPG_EX_NO_TOOLBAR_DIVIDER = 0x04000000,
  wxPG_EX_TOOLBAR_SEPARATOR = 0x08000000 }

  Combines various styles.

Classes

- class wxPGValidationInfo

  wxPGValidationInfo
- class wxPropertyGrid

  wxPropertyGrid is a specialized grid for editing properties - in other words name = value pairs.
- class wxPropertyGridEvent

  A property grid event holds information about events associated with wxPropertyGrid objects.

Typedefs

- typedef wxByte wxPGVFBFlags
- typedef int(∗ wxPGSortCallback)(wxPropertyGrid ∗propGrid, wxPGProperty ∗p1, wxPGProperty ∗p2)

  This callback function is used for sorting properties.

Enumerations

- enum wxPG_VALIDATION_FAILURE_BEHAVIOR_FLAGS {
  wxPG_VFB_STAY_IN_PROPERTY = 0x01,
  wxPG_VFB_BEEP = 0x02,
  wxPG_VFB_MARK_CELL = 0x04,
  wxPG_VFB_SHOW_MESSAGE = 0x08,
  wxPG_VFB_SHOW_MESSAGEBOX = 0x10,
  wxPG_VFB_SHOW_MESSAGE_ON_STATUSBAR = 0x20,
  wxPG_VFB_DEFAULT }

- enum wxPG_KEYBOARD_ACTIONS {
  wxPG_ACTION_INVALID = 0,
  wxPG_ACTION_NEXT_PROPERTY,
  wxPG_ACTION_PREV_PROPERTY,
  wxPG_ACTION_EXPAND_PROPERTY,
  wxPG_ACTION_COLLAPSE_PROPERTY,
  wxPG_ACTION_CANCEL_EDIT,
  wxPG_ACTION_EDIT,
  wxPG_ACTION_PRESS_BUTTON,
  wxPG_ACTION_MAX }

22.144.1 Macro Definition Documentation

#define wxPG_DEFAULT_STYLE (0)

Combines various styles.

#define wxPGMAN_DEFAULT_STYLE (0)

Combines various styles.

22.144.2 Typedef Documentation

typedef int(∗ wxPGSortCallback)(wxPropertyGrid ∗propGrid, wxPGProperty ∗p1, wxPGProperty ∗p2)

This callback function is used for sorting properties.

Call wxPropertyGrid::SetSortFunction() to set it.

Sort function should return a value greater than 0 if position of p1 is after p2. So, for instance, when comparing property names, you can use following implementation:

```
1 int MyPropertySortFunction(wxPropertyGrid* propGrid,
2                            wxPGProperty* p1,
3                            wxPGProperty* p2)
4 {
5     return p1->GetBaseName().compare( p2->GetBaseName() );
6 }
```

typedef wxByte wxPGVFBFlags

22.144.3 Enumeration Type Documentation

enum wxPG_EX_WINDOW_STYLES

Combines various styles.

Enumerator

wxPG_EX_INIT_NOCAT NOTE: wxPG_EX_xxx are extra window styles and must be set using SetExtra↩ Style() member function. Speeds up switching to wxPG_HIDE_CATEGORIES mode. Initially, if wxPG_↩ HIDE_CATEGORIES is not defined, the non-categorized data storage is not activated, and switching the mode first time becomes somewhat slower. wxPG_EX_INIT_NOCAT activates the non-categorized data storage right away.

NOTE: If you do plan not switching to non-categoric mode, or if you don't plan to use categories at all, then using this style will result in waste of resources.

wxPG_EX_NO_FLAT_TOOLBAR Extended window style that sets wxPropertyGridManager tool bar to not use flat style.

wxPG_EX_MODE_BUTTONS Shows alphabetic/categoric mode buttons from tool bar.

wxPG_EX_HELP_AS_TOOLTIPS Show property help strings as tool tips instead as text on the status bar. You can set the help strings using SetPropertyHelpString member function.

wxPG_EX_NATIVE_DOUBLE_BUFFERING Allows relying on native double-buffering.

wxPG_EX_AUTO_UNSPECIFIED_VALUES Set this style to let user have ability to set values of properties to unspecified state. Same as setting wxPG_PROP_AUTO_UNSPECIFIED for all properties.

wxPG_EX_WRITEONLY_BUILTIN_ATTRIBUTES If this style is used, built-in attributes (such as wxPG_↩ FLOAT_PRECISION and wxPG_STRING_PASSWORD) are not stored into property's attribute storage (thus they are not readable). Note that this option is global, and applies to all wxPG property containers.

wxPG_EX_HIDE_PAGE_BUTTONS Hides page selection buttons from tool bar.

wxPG_EX_MULTIPLE_SELECTION Allows multiple properties to be selected by user (by pressing SHI↩ FT when clicking on a property, or by dragging with left mouse button down). You can get array of selected properties with wxPropertyGridInterface::GetSelectedProperties(). In multiple selection mode wxPropertyGridInterface::GetSelection() returns property which has editor active (usually the first one selected). Other useful member functions are ClearSelection(), AddToSelection() and RemoveFrom↩ Selection().

wxPG_EX_ENABLE_TLP_TRACKING This enables top-level window tracking which allows wxPropertyGrid to notify the application of last-minute property value changes by user. This style is not enabled by default because it may cause crashes when wxPropertyGrid is used in with wxAUI or similar system.

Remarks

If you are not in fact using any system that may change wxPropertyGrid's top-level parent window on its own, then you are recommended to enable this style.

wxPG_EX_NO_TOOLBAR_DIVIDER Don't show divider above toolbar, on Windows.

wxPG_EX_TOOLBAR_SEPARATOR Show a separator below the toolbar.

enum wxPG_KEYBOARD_ACTIONS

22.144.4 wxPropertyGrid Action Identifiers

These are used with wxPropertyGrid::AddActionTrigger() and wxPropertyGrid::ClearActionTriggers().

Enumerator

wxPG_ACTION_INVALID

wxPG_ACTION_NEXT_PROPERTY Select the next property.

wxPG_ACTION_PREV_PROPERTY Select the previous property.

wxPG_ACTION_EXPAND_PROPERTY Expand the selected property, if it has child items.

wxPG_ACTION_COLLAPSE_PROPERTY Collapse the selected property, if it has child items.

wxPG_ACTION_CANCEL_EDIT Cancel and undo any editing done in the currently active property editor.

wxPG_ACTION_EDIT Move focus to the editor control of the currently selected property.

wxPG_ACTION_PRESS_BUTTON Causes editor's button (if any) to be pressed.

wxPG_ACTION_MAX

enum wxPG_VALIDATION_FAILURE_BEHAVIOR_FLAGS

22.144.5 wxPropertyGrid Validation Failure behaviour Flags

Enumerator

wxPG_VFB_STAY_IN_PROPERTY Prevents user from leaving property unless value is valid. If this behaviour flag is not used, then value change is instead cancelled.

wxPG_VFB_BEEP Calls wxBell() on validation failure.

wxPG_VFB_MARK_CELL Cell with invalid value will be marked (with red colour).

wxPG_VFB_SHOW_MESSAGE Display a text message explaining the situation. To customize the way the message is displayed, you need to reimplement wxPropertyGrid::DoShowPropertyError() in a derived class. Default behaviour is to display the text on the top-level frame's status bar, if present, and otherwise using wxMessageBox.

wxPG_VFB_SHOW_MESSAGEBOX Similar to wxPG_VFB_SHOW_MESSAGE, except always displays the message using wxMessageBox.

wxPG_VFB_SHOW_MESSAGE_ON_STATUSBAR Similar to wxPG_VFB_SHOW_MESSAGE, except always displays the message on the status bar (when present - you can reimplement wxPropertyGrid::GetStatusBar() in a derived class to specify this yourself).

wxPG_VFB_DEFAULT Defaults.

enum wxPG_WINDOW_STYLES

22.144.6 wxPropertyGrid Window Styles

SetWindowStyleFlag method can be used to modify some of these at run-time.

Enumerator

wxPG_AUTO_SORT This will cause Sort() automatically after an item is added. When inserting a lot of items in this mode, it may make sense to use Freeze() before operations and Thaw() afterwards to increase performance.

wxPG_HIDE_CATEGORIES Categories are not initially shown (even if added). IMPORTANT NOTE: If you do not plan to use categories, then this style will waste resources. This flag can also be changed using wxPropertyGrid::EnableCategories method.

wxPG_ALPHABETIC_MODE This style combines non-categoric mode and automatic sorting.

wxPG_BOLD_MODIFIED Modified values are shown in bold font. Changing this requires Refresh() to show changes.

wxPG_SPLITTER_AUTO_CENTER When wxPropertyGrid is resized, splitter moves to the center. This behaviour stops once the user manually moves the splitter.

wxPG_TOOLTIPS Display tool tips for cell text that cannot be shown completely. If wxUSE_TOOLTIPS is 0, then this doesn't have any effect.

wxPG_HIDE_MARGIN Disables margin and hides all expand/collapse buttons that would appear outside the margin (for sub-properties). Toggling this style automatically expands all collapsed items.

wxPG_STATIC_SPLITTER This style prevents user from moving the splitter.

wxPG_STATIC_LAYOUT Combination of other styles that make it impossible for user to modify the layout.

wxPG_LIMITED_EDITING Disables wxTextCtrl based editors for properties which can be edited in another way. Equals calling wxPropertyGrid::LimitPropertyEditing() for all added properties.

wxPG_TOOLBAR wxPropertyGridManager only: Show tool bar for mode and page selection.

wxPG_DESCRIPTION wxPropertyGridManager only: Show adjustable text box showing description or help text, if available, for currently selected property.

wxPG_NO_INTERNAL_BORDER wxPropertyGridManager only: don't show an internal border around the property grid. Recommended if you use a header.

22.145 docs/doxygen/overviews/python.h File Reference

22.146 docs/doxygen/overviews/refcount.h File Reference

22.147 docs/doxygen/overviews/referencenotes.h File Reference

22.148 docs/doxygen/overviews/resyntax.h File Reference

22.149 docs/doxygen/overviews/richtextctrl.h File Reference

22.150 interface/wx/richtext/richtextctrl.h File Reference

Classes

- class wxRichTextContextMenuPropertiesInfo

  wxRichTextContextMenuPropertiesInfo keeps track of objects that appear in the context menu, whose properties are available to be edited.
- class wxRichTextCtrl

  wxRichTextCtrl provides a generic, ground-up implementation of a text control capable of showing multiple styles and images.
- class wxRichTextEvent

  This is the event class for wxRichTextCtrl notifications.

Macros

- #define wxRE_READONLY 0x0010

  Styles.
- #define wxRE_MULTILINE 0x0020

- #define wxRE_CENTRE_CARET 0x8000
- #define wxRE_CENTER_CARET wxRE_CENTRE_CARET
- #define wxRICHTEXT_SHIFT_DOWN 0x01

  Flags.
- #define wxRICHTEXT_CTRL_DOWN 0x02
- #define wxRICHTEXT_ALT_DOWN 0x04
- #define wxRICHTEXT_EX_NO_GUIDELINES 0x00000100

  Extra flags.
- #define wxRICHTEXT_DEFAULT_OVERALL_SIZE wxSize(-1, -1)
- #define wxRICHTEXT_DEFAULT_IMAGE_SIZE wxSize(80, 80)
- #define wxRICHTEXT_DEFAULT_SPACING 3
- #define wxRICHTEXT_DEFAULT_MARGIN 3
- #define wxRICHTEXT_DEFAULT_UNFOCUSSED_BACKGROUND wxColour(175, 175, 175)
- #define wxRICHTEXT_DEFAULT_FOCUSSED_BACKGROUND wxColour(140, 140, 140)
- #define wxRICHTEXT_DEFAULT_UNSELECTED_BACKGROUND wxSystemSettings::GetColour(wxSYS↩_COLOUR_3DFACE)
- #define wxRICHTEXT_DEFAULT_TYPE_COLOUR wxColour(0, 0, 200)
- #define wxRICHTEXT_DEFAULT_FOCUS_RECT_COLOUR wxColour(100, 80, 80)
- #define wxRICHTEXT_DEFAULT_CARET_WIDTH 2
- #define wxRICHTEXT_DEFAULT_DELAYED_LAYOUT_THRESHOLD 20000
- #define wxRICHTEXT_DEFAULT_LAYOUT_INTERVAL 50
- #define wxRICHTEXT_DEFAULT_DELAYED_IMAGE_PROCESSING_INTERVAL 200
- #define wxID_RICHTEXT_PROPERTIES1 (wxID_HIGHEST + 1)
- #define wxID_RICHTEXT_PROPERTIES2 (wxID_HIGHEST + 2)
- #define wxID_RICHTEXT_PROPERTIES3 (wxID_HIGHEST + 3)

Enumerations

- enum wxRichTextCtrlSelectionState {
  wxRichTextCtrlSelectionState_Normal,
  wxRichTextCtrlSelectionState_CommonAncestor }

Variables

- wxEventType wxEVT_RICHTEXT_LEFT_CLICK
- wxEventType wxEVT_RICHTEXT_RIGHT_CLICK
- wxEventType wxEVT_RICHTEXT_MIDDLE_CLICK
- wxEventType wxEVT_RICHTEXT_LEFT_DCLICK
- wxEventType wxEVT_RICHTEXT_RETURN
- wxEventType wxEVT_RICHTEXT_CHARACTER
- wxEventType wxEVT_RICHTEXT_CONSUMING_CHARACTER
- wxEventType wxEVT_RICHTEXT_DELETE
- wxEventType wxEVT_RICHTEXT_STYLESHEET_CHANGING
- wxEventType wxEVT_RICHTEXT_STYLESHEET_CHANGED
- wxEventType wxEVT_RICHTEXT_STYLESHEET_REPLACING
- wxEventType wxEVT_RICHTEXT_STYLESHEET_REPLACED
- wxEventType wxEVT_RICHTEXT_CONTENT_INSERTED
- wxEventType wxEVT_RICHTEXT_CONTENT_DELETED
- wxEventType wxEVT_RICHTEXT_STYLE_CHANGED
- wxEventType wxEVT_RICHTEXT_PROPERTIES_CHANGED
- wxEventType wxEVT_RICHTEXT_SELECTION_CHANGED
- wxEventType wxEVT_RICHTEXT_BUFFER_RESET
- wxEventType wxEVT_RICHTEXT_FOCUS_OBJECT_CHANGED

22.150.1 Macro Definition Documentation

#define wxID_RICHTEXT_PROPERTIES1 (wxID_HIGHEST + 1)

#define wxID_RICHTEXT_PROPERTIES2 (wxID_HIGHEST + 2)

#define wxID_RICHTEXT_PROPERTIES3 (wxID_HIGHEST + 3)

#define wxRE_CENTER_CARET wxRE_CENTRE_CARET

#define wxRE_CENTRE_CARET 0x8000

#define wxRE_MULTILINE 0x0020

#define wxRE_READONLY 0x0010

Styles.

#define wxRICHTEXT_ALT_DOWN 0x04

#define wxRICHTEXT_CTRL_DOWN 0x02

#define wxRICHTEXT_DEFAULT_CARET_WIDTH 2

#define wxRICHTEXT_DEFAULT_DELAYED_IMAGE_PROCESSING_INTERVAL 200

#define wxRICHTEXT_DEFAULT_DELAYED_LAYOUT_THRESHOLD 20000

#define wxRICHTEXT_DEFAULT_FOCUS_RECT_COLOUR wxColour(100, 80, 80)

#define wxRICHTEXT_DEFAULT_FOCUSSED_BACKGROUND wxColour(140, 140, 140)

#define wxRICHTEXT_DEFAULT_IMAGE_SIZE wxSize(80, 80)

#define wxRICHTEXT_DEFAULT_LAYOUT_INTERVAL 50

#define wxRICHTEXT_DEFAULT_MARGIN 3

#define wxRICHTEXT_DEFAULT_OVERALL_SIZE wxSize(-1, -1)

#define wxRICHTEXT_DEFAULT_SPACING 3

#define wxRICHTEXT_DEFAULT_TYPE_COLOUR wxColour(0, 0, 200)

#define wxRICHTEXT_DEFAULT_UNFOCUSSED_BACKGROUND wxColour(175, 175, 175)

#define wxRICHTEXT_DEFAULT_UNSELECTED_BACKGROUND wxSystemSettings::GetColour(wxSYS_COLOUR_3↩
DFACE)

#define wxRICHTEXT_EX_NO_GUIDELINES 0x00000100

Extra flags.

#define wxRICHTEXT_SHIFT_DOWN 0x01

Flags.

22.150.2 Enumeration Type Documentation

enum wxRichTextCtrlSelectionState

Enumerator

wxRichTextCtrlSelectionState_Normal

wxRichTextCtrlSelectionState_CommonAncestor

22.150.3 Variable Documentation

wxEventType wxEVT_RICHTEXT_BUFFER_RESET

wxEventType wxEVT_RICHTEXT_CHARACTER

wxEventType wxEVT_RICHTEXT_CONSUMING_CHARACTER

wxEventType wxEVT_RICHTEXT_CONTENT_DELETED

wxEventType wxEVT_RICHTEXT_CONTENT_INSERTED

wxEventType wxEVT_RICHTEXT_DELETE

wxEventType wxEVT_RICHTEXT_FOCUS_OBJECT_CHANGED

wxEventType wxEVT_RICHTEXT_LEFT_CLICK

wxEventType wxEVT_RICHTEXT_LEFT_DCLICK

wxEventType wxEVT_RICHTEXT_MIDDLE_CLICK

wxEventType wxEVT_RICHTEXT_PROPERTIES_CHANGED

wxEventType wxEVT_RICHTEXT_RETURN

wxEventType wxEVT_RICHTEXT_RIGHT_CLICK

wxEventType wxEVT_RICHTEXT_SELECTION_CHANGED

wxEventType wxEVT_RICHTEXT_STYLE_CHANGED

wxEventType wxEVT_RICHTEXT_STYLESHEET_CHANGED

wxEventType wxEVT_RICHTEXT_STYLESHEET_CHANGING

wxEventType wxEVT_RICHTEXT_STYLESHEET_REPLACED

wxEventType wxEVT_RICHTEXT_STYLESHEET_REPLACING

22.151 docs/doxygen/overviews/roughguide.h File Reference

22.152 docs/doxygen/overviews/runtimeclass.h File Reference

22.153 docs/doxygen/overviews/scrolling.h File Reference

22.154 docs/doxygen/overviews/sizer.h File Reference

22.155 interface/wx/sizer.h File Reference

Classes

- class wxSizer

  wxSizer is the abstract base class used for laying out subwindows in a window.
- class wxStdDialogButtonSizer

  This class creates button layouts which conform to the standard button spacing and ordering defined by the platform or toolkit's user interface guidelines (if such things exist).
- class wxSizerItem

  The wxSizerItem class is used to track the position, size and other attributes of each item managed by a wxSizer.
- class wxSizerFlags

  Container for sizer items flags providing readable names for them.
- class wxFlexGridSizer

  A flex grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the wxGridSizer.
- class wxGridSizer

  A grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields having the same size, i.e.
- class wxStaticBoxSizer

  wxStaticBoxSizer is a sizer derived from wxBoxSizer but adds a static box around the sizer.
- class wxBoxSizer

  The basic idea behind a box sizer is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.

Enumerations

- enum wxFlexSizerGrowMode {
  wxFLEX_GROWMODE_NONE,
  wxFLEX_GROWMODE_SPECIFIED,
  wxFLEX_GROWMODE_ALL }

  Values which define the behaviour for resizing wxFlexGridSizer cells in the "non-flexible" direction.

22.155.1 Enumeration Type Documentation

enum wxFlexSizerGrowMode

Values which define the behaviour for resizing wxFlexGridSizer cells in the "non-flexible" direction.

Enumerator

wxFLEX_GROWMODE_NONE Don't resize the cells in non-flexible direction at all.

wxFLEX_GROWMODE_SPECIFIED Uniformly resize only the specified ones (default).

wxFLEX_GROWMODE_ALL Uniformly resize all cells.

22.156 docs/doxygen/overviews/splitterwindow.h File Reference

22.157 docs/doxygen/overviews/stream.h File Reference

22.158 interface/wx/stream.h File Reference

Classes

- class wxStreamBase

  This class is the base class of most stream related classes in wxWidgets.
- class wxStreamBuffer

  wxStreamBuffer is a cache manager for wxStreamBase: it manages a stream buffer linked to a stream.
- class wxOutputStream

  wxOutputStream is an abstract base class which may not be used directly.
- class wxInputStream

  wxInputStream is an abstract base class which may not be used directly.
- class wxCountingOutputStream

  wxCountingOutputStream is a specialized output stream which does not write any data anywhere, instead it counts how many bytes would get written if this were a normal stream.
- class wxBufferedInputStream

  This stream acts as a cache.
- class wxFilterClassFactory

  Allows the creation of filter streams to handle compression formats such as gzip and bzip2.
- class wxFilterOutputStream

  A filter stream has the capability of a normal stream but it can be placed on top of another stream.
- class wxFilterInputStream

  A filter stream has the capability of a normal stream but it can be placed on top of another stream.
- class wxBufferedOutputStream

  This stream acts as a cache.
- class wxWrapperInputStream

  A wrapper input stream is a kind of filter stream which forwards all the operations to its base stream.

Enumerations

- enum wxStreamError {
  wxSTREAM_NO_ERROR = 0,
  wxSTREAM_EOF,
  wxSTREAM_WRITE_ERROR,
  wxSTREAM_READ_ERROR }

  These enumeration values are returned by various functions in the context of wxStream classes.
- enum wxStreamProtocolType {
  wxSTREAM_PROTOCOL,
  wxSTREAM_MIMETYPE,
  wxSTREAM_ENCODING,
  wxSTREAM_FILEEXT }

  Enumeration values used by wxFilterClassFactory.

22.158.1 Enumeration Type Documentation

enum wxStreamError

These enumeration values are returned by various functions in the context of wxStream classes.

Enumerator

wxSTREAM_NO_ERROR No error occurred.

wxSTREAM_EOF EOF reached in Read() or similar.

wxSTREAM_WRITE_ERROR generic write error on the last write call.

wxSTREAM_READ_ERROR generic read error on the last read call.

enum wxStreamProtocolType

Enumeration values used by wxFilterClassFactory.

Enumerator

wxSTREAM_PROTOCOL wxFileSystem protocol (should be only one).

wxSTREAM_MIMETYPE MIME types the stream handles.

wxSTREAM_ENCODING The HTTP Content-Encodings the stream handles.

wxSTREAM_FILEEXT File extensions the stream handles.

22.159 docs/doxygen/overviews/string.h File Reference

22.160 interface/wx/string.h File Reference

Classes

- class wxString

  String class for passing textual data to or receiving it from wxWidgets.
- class wxStringBufferLength

  This tiny class allows you to conveniently access the wxString internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later, and allows the user to set the internal length of the string.
- class wxStringBuffer

  This tiny class allows you to conveniently access the wxString internal buffer as a writable pointer without any risk of forgetting to restore the string to the usable state later.

Functions

- template<bool(T)(const wxUniChar &c) >
  bool wxStringCheck (const wxString &val)

  Allows to extend a function with the signature:

- bool operator== (const wxString &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator!= (const wxString &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator< (const wxString &s1, const wxString &s2)

Comparison operator for string types.

- bool operator> (const wxString &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator<= (const wxString &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator>= (const wxString &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator== (const wxString &s1, const wxCStrData &s2)

  Comparison operator for string types.
- bool operator== (const wxCStrData &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator!= (const wxString &s1, const wxCStrData &s2)

  Comparison operator for string types.
- bool operator!= (const wxCStrData &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator== (const wxString &s1, const wxWCharBuffer &s2)

  Comparison operator for string types.
- bool operator== (const wxWCharBuffer &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator!= (const wxString &s1, const wxWCharBuffer &s2)

  Comparison operator for string types.
- bool operator!= (const wxWCharBuffer &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator== (const wxString &s1, const wxCharBuffer &s2)

  Comparison operator for string types.
- bool operator== (const wxCharBuffer &s1, const wxString &s2)

  Comparison operator for string types.
- bool operator!= (const wxString &s1, const wxCharBuffer &s2)

  Comparison operator for string types.
- bool operator!= (const wxCharBuffer &s1, const wxString &s2)

  Comparison operator for string types.

- bool operator== (const wxUniChar &c, const wxString &s)

  Comparison operators char types.
- bool operator== (const wxUniCharRef &c, const wxString &s)

  Comparison operators char types.
- bool operator== (char c, const wxString &s)

  Comparison operators char types.
- bool operator== (wchar_t c, const wxString &s)

  Comparison operators char types.
- bool operator== (int c, const wxString &s)

  Comparison operators char types.
- bool operator== (const wxString &s, const wxUniChar &c)

  Comparison operators char types.
- bool operator== (const wxString &s, const wxUniCharRef &c)

  Comparison operators char types.
- bool operator== (const wxString &s, char c)

  Comparison operators char types.
- bool operator== (const wxString &s, wchar_t c)

  Comparison operators char types.
- bool operator!= (const wxUniChar &c, const wxString &s)

Comparison operators char types.

- bool operator!= (const wxUniCharRef &c, const wxString &s)

  Comparison operators char types.
- bool operator!= (char c, const wxString &s)

  Comparison operators char types.
- bool operator!= (wchar_t c, const wxString &s)

  Comparison operators char types.
- bool operator!= (int c, const wxString &s)

  Comparison operators char types.
- bool operator!= (const wxString &s, const wxUniChar &c)

  Comparison operators char types.
- bool operator!= (const wxString &s, const wxUniCharRef &c)

  Comparison operators char types.
- bool operator!= (const wxString &s, char c)

  Comparison operators char types.
- bool operator!= (const wxString &s, wchar_t c)

  Comparison operators char types.

Variables

- wxString wxEmptyString

  The global wxString instance of an empty string.

22.160.1 Function Documentation

bool operator!= (const wxString & *s1*, const wxString & *s2*) `[inline]`

Comparison operator for string types.

bool operator!= (const wxString & *s1*, const wxCStrData & *s2*) `[inline]`

Comparison operator for string types.

bool operator!= (const wxCStrData & *s1*, const wxString & *s2*) `[inline]`

Comparison operator for string types.

bool operator!= (const wxString & *s1*, const wxWCharBuffer & *s2*) `[inline]`

Comparison operator for string types.

bool operator!= (const wxWCharBuffer & *s1*, const wxString & *s2*) `[inline]`

Comparison operator for string types.

bool operator!= (const wxString & *s1*, const wxCharBuffer & *s2*) `[inline]`

Comparison operator for string types.

bool operator!= (const wxCharBuffer & *s1*, const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator!= (const wxUniChar & *c*, const wxString & *s*) [inline]

Comparison operators char types.

bool operator!= (const wxUniCharRef & *c*, const wxString & *s*) [inline]

Comparison operators char types.

bool operator!= (char *c*, const wxString & *s*) [inline]

Comparison operators char types.

bool operator!= (wchar_t *c*, const wxString & *s*) [inline]

Comparison operators char types.

bool operator!= (int *c*, const wxString & *s*) [inline]

Comparison operators char types.

bool operator!= (const wxString & *s*, const wxUniChar & *c*) [inline]

Comparison operators char types.

bool operator!= (const wxString & *s*, const wxUniCharRef & *c*) [inline]

Comparison operators char types.

bool operator!= (const wxString & *s*, char *c*) [inline]

Comparison operators char types.

bool operator!= (const wxString & *s*, wchar_t *c*) [inline]

Comparison operators char types.

bool operator< (const wxString & *s1*, const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator<= (const wxString & *s1*, const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxString & *s1,* const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxString & *s1,* const wxCStrData & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxCStrData & *s1,* const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxString & *s1,* const wxWCharBuffer & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxWCharBuffer & *s1,* const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxString & *s1,* const wxCharBuffer & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxCharBuffer & *s1,* const wxString & *s2*) [inline]

Comparison operator for string types.

bool operator== (const wxUniChar & *c,* const wxString & *s*) [inline]

Comparison operators char types.

bool operator== (const wxUniCharRef & *c,* const wxString & *s*) [inline]

Comparison operators char types.

bool operator== (char *c,* const wxString & *s*) [inline]

Comparison operators char types.

bool operator== (wchar_t *c,* const wxString & *s*) [inline]

Comparison operators char types.

bool operator== (int *c,* const wxString & *s*) [inline]

Comparison operators char types.

bool operator== (const wxString & *s*, const wxUniChar & *c*) `[inline]`

Comparison operators char types.

bool operator== (const wxString & *s*, const wxUniCharRef & *c*) `[inline]`

Comparison operators char types.

bool operator== (const wxString & *s*, char *c*) `[inline]`

Comparison operators char types.

bool operator== (const wxString & *s*, wchar_t *c*) `[inline]`

Comparison operators char types.

bool operator> (const wxString & *s1*, const wxString & *s2*) `[inline]`

Comparison operator for string types.

bool operator>= (const wxString & *s1*, const wxString & *s2*) `[inline]`

Comparison operator for string types.

22.160.2 Variable Documentation

wxString wxEmptyString

The global wxString instance of an empty string.

Used extensively in the entire wxWidgets API.

22.161 docs/doxygen/overviews/thread.h File Reference

22.162 interface/wx/thread.h File Reference

Classes

- class wxCondition

  wxCondition variables correspond to pthread conditions or to Win32 event objects.
- class wxCriticalSectionLocker

  This is a small helper class to be used with wxCriticalSection objects.
- class wxThreadHelper

  The wxThreadHelper class is a mix-in class that manages a single background thread, either detached or joinable (see wxThread for the differences).
- class wxCriticalSection

  A critical section object is used for exactly the same purpose as a wxMutex.
- class wxThread

  A thread is basically a path of execution through a program.
- class wxSemaphore

wxSemaphore is a counter limiting the number of threads concurrently accessing a shared resource.

- class wxMutexLocker

  This is a small helper class to be used with wxMutex objects.

- class wxMutex

  A mutex object is a synchronization object whose state is set to signaled when it is not owned by any thread, and nonsignaled when it is owned.

Macros

- #define wxCRIT_SECT_DECLARE(cs)

  This macro declares a (static) critical section object named cs if `wxUSE_THREADS` is 1 and does nothing if it is 0.

- #define wxCRIT_SECT_DECLARE_MEMBER(cs)

  This macro declares a critical section object named cs if `wxUSE_THREADS` is 1 and does nothing if it is 0.

- #define wxCRIT_SECT_LOCKER(name, cs)

  This macro creates a wxCriticalSectionLocker named name and associated with the critical section cs if `wxUSE_THREADS` is 1 and does nothing if it is 0.

- #define wxCRITICAL_SECTION(name)

  This macro combines wxCRIT_SECT_DECLARE() and wxCRIT_SECT_LOCKER(): it creates a static critical section object and also the lock object associated with it.

- #define wxLEAVE_CRIT_SECT(critical_section)

  This macro is equivalent to critical_section.Leave() if `wxUSE_THREADS` is 1 and does nothing if it is 0.

- #define wxENTER_CRIT_SECT(critical_section)

  This macro is equivalent to critical_section.Enter() if `wxUSE_THREADS` is 1 and does nothing if it is 0.

Enumerations

- enum wxCondError {
  wxCOND_NO_ERROR = 0,
  wxCOND_INVALID,
  wxCOND_TIMEOUT,
  wxCOND_MISC_ERROR }

  See wxCondition.

- enum wxCriticalSectionType {
  wxCRITSEC_DEFAULT,
  wxCRITSEC_NON_RECURSIVE }

  Possible critical section types.

- enum wxThreadWait {
  wxTHREAD_WAIT_BLOCK,
  wxTHREAD_WAIT_YIELD,
  wxTHREAD_WAIT_DEFAULT = wxTHREAD_WAIT_YIELD }

  The possible thread wait types.

- enum wxThreadKind {
  wxTHREAD_DETACHED,
  wxTHREAD_JOINABLE }

  The possible thread kinds.

- enum wxThreadError {
  wxTHREAD_NO_ERROR = 0,
  wxTHREAD_NO_RESOURCE,
  wxTHREAD_RUNNING,
  wxTHREAD_NOT_RUNNING,
  wxTHREAD_KILLED,
  wxTHREAD_MISC_ERROR }

  The possible thread errors.

- enum wxSemaError {
  wxSEMA_NO_ERROR = 0,
  wxSEMA_INVALID,
  wxSEMA_BUSY,
  wxSEMA_TIMEOUT,
  wxSEMA_OVERFLOW,
  wxSEMA_MISC_ERROR }

  See wxSemaphore.
- enum wxMutexType {
  wxMUTEX_DEFAULT,
  wxMUTEX_RECURSIVE }

  The possible wxMutex kinds.
- enum wxMutexError {
  wxMUTEX_NO_ERROR = 0,
  wxMUTEX_INVALID,
  wxMUTEX_DEAD_LOCK,
  wxMUTEX_BUSY,
  wxMUTEX_UNLOCKED,
  wxMUTEX_TIMEOUT,
  wxMUTEX_MISC_ERROR }

  The possible wxMutex errors.

Functions

- bool wxIsMainThread ()

  Returns true if this thread is the main one.
- void wxMutexGuiEnter ()

  This function must be called when any thread other than the main GUI thread wants to get access to the GUI library.
- void wxMutexGuiLeave ()

  This function is only defined on platforms which support preemptive threads.

22.162.1 Enumeration Type Documentation

enum wxCondError

See wxCondition.

Enumerator

wxCOND_NO_ERROR

wxCOND_INVALID

wxCOND_TIMEOUT WaitTimeout() has timed out.

wxCOND_MISC_ERROR

enum wxCriticalSectionType

Possible critical section types.

Enumerator

wxCRITSEC_DEFAULT

wxCRITSEC_NON_RECURSIVE Recursive critical section under both Windows and Unix. Non-recursive critical section under Unix, recursive under Windows

enum wxMutexError

The possible wxMutex errors.

Enumerator

wxMUTEX_NO_ERROR The operation completed successfully.

wxMUTEX_INVALID The mutex hasn't been initialized.

wxMUTEX_DEAD_LOCK The mutex is already locked by the calling thread.

wxMUTEX_BUSY The mutex is already locked by another thread.

wxMUTEX_UNLOCKED An attempt to unlock a mutex which is not locked.

wxMUTEX_TIMEOUT wxMutex::LockTimeout() has timed out.

wxMUTEX_MISC_ERROR Any other error.

enum wxMutexType

The possible wxMutex kinds.

Enumerator

wxMUTEX_DEFAULT Normal non-recursive mutex: try to always use this one.

wxMUTEX_RECURSIVE Recursive mutex: don't use these ones with wxCondition.

enum wxSemaError

See wxSemaphore.

Enumerator

wxSEMA_NO_ERROR

wxSEMA_INVALID semaphore hasn't been initialized successfully

wxSEMA_BUSY returned by TryWait() if Wait() would block

wxSEMA_TIMEOUT returned by WaitTimeout()

wxSEMA_OVERFLOW Post() would increase counter past the max.

wxSEMA_MISC_ERROR

enum wxThreadError

The possible thread errors.

Enumerator

wxTHREAD_NO_ERROR No error.

wxTHREAD_NO_RESOURCE No resource left to create a new thread.

wxTHREAD_RUNNING The thread is already running.

wxTHREAD_NOT_RUNNING The thread isn't running.

wxTHREAD_KILLED Thread we waited for had to be killed.

wxTHREAD_MISC_ERROR Some other error.

enum wxThreadKind

The possible thread kinds.

Enumerator

wxTHREAD_DETACHED Detached thread.

wxTHREAD_JOINABLE Joinable thread.

enum wxThreadWait

The possible thread wait types.

Since

2.9.2

Enumerator

wxTHREAD_WAIT_BLOCK No events are processed while waiting. This is the default under all platforms except for wxMSW.

wxTHREAD_WAIT_YIELD Yield for event dispatching while waiting. This flag is dangerous as it exposes the program using it to unexpected reentrancies in the same way as calling wxYield() function does so you are strongly advised to avoid its use and not wait for the thread termination from the main (GUI) thread at all to avoid making your application unresponsive.

Also notice that this flag is not portable as it is only implemented in wxMSW and simply ignored under the other platforms.

wxTHREAD_WAIT_DEFAULT Default wait mode for wxThread::Wait() and wxThread::Delete(). For compatibility reasons, the default wait mode is currently wxTHREAD_WAIT_YIELD if WXWIN_COMPATIBILIT↩ Y_2_8 is defined (and it is by default). However, as mentioned above, you're strongly encouraged to not use wxTHREAD_WAIT_YIELD and pass wxTHREAD_WAIT_BLOCK to wxThread method explicitly.

22.163 docs/doxygen/overviews/tips.h File Reference

22.164 docs/doxygen/overviews/toolbar.h File Reference

22.165 interface/wx/ribbon/toolbar.h File Reference

Classes

- class wxRibbonToolBar

  A ribbon tool bar is similar to a traditional toolbar which has no labels.

22.166 interface/wx/toolbar.h File Reference

Classes

- class wxToolBarToolBase

  A toolbar tool represents one item on the toolbar.
- class wxToolBar

  A toolbar is a bar of buttons and/or other controls usually placed below the menu bar in a wxFrame.

Enumerations

- enum wxToolBarToolStyle {
  wxTOOL_STYLE_BUTTON = 1,
  wxTOOL_STYLE_SEPARATOR = 2,
  wxTOOL_STYLE_CONTROL }
- enum {
  wxTB_HORIZONTAL = wxHORIZONTAL,
  wxTB_TOP = wxTB_HORIZONTAL,
  wxTB_VERTICAL = wxVERTICAL,
  wxTB_LEFT = wxTB_VERTICAL,
  wxTB_3DBUTTONS,
  wxTB_FLAT,
  wxTB_DOCKABLE,
  wxTB_NOICONS,
  wxTB_TEXT,
  wxTB_NODIVIDER,
  wxTB_NOALIGN,
  wxTB_HORZ_LAYOUT,
  wxTB_HORZ_TEXT = wxTB_HORZ_LAYOUT | wxTB_TEXT,
  wxTB_NO_TOOLTIPS,
  wxTB_BOTTOM,
  wxTB_RIGHT,
  wxTB_DEFAULT_STYLE = wxTB_HORIZONTAL }

  wxToolBar style flags

22.166.1 Enumeration Type Documentation

anonymous enum

wxToolBar style flags

Enumerator

wxTB_HORIZONTAL lay out the toolbar horizontally

wxTB_TOP

wxTB_VERTICAL lay out the toolbar vertically

wxTB_LEFT

wxTB_3DBUTTONS show 3D buttons (wxToolBarSimple only)

wxTB_FLAT "flat" buttons (Win32/GTK only)

wxTB_DOCKABLE dockable toolbar (GTK only)

wxTB_NOICONS don't show the icons (they're shown by default)

wxTB_TEXT show the text (not shown by default)

wxTB_NODIVIDER don't show the divider between toolbar and the window (Win32 only)

wxTB_NOALIGN no automatic alignment (Win32 only, useless)

wxTB_HORZ_LAYOUT show the text and the icons alongside, not vertically stacked (Win32/GTK)

wxTB_HORZ_TEXT

wxTB_NO_TOOLTIPS don't show the toolbar short help tooltips

wxTB_BOTTOM lay out toolbar at the bottom of the window

wxTB_RIGHT lay out toolbar at the right edge of the window

wxTB_DEFAULT_STYLE flags that are closest to the native look

enum wxToolBarToolStyle

Enumerator

wxTOOL_STYLE_BUTTON

wxTOOL_STYLE_SEPARATOR

wxTOOL_STYLE_CONTROL

22.167 docs/doxygen/overviews/treectrl.h File Reference

22.168 interface/wx/treectrl.h File Reference

Classes

- class wxTreeCtrl

  A tree control presents information as a hierarchy, with items that may be expanded to show further items.
- class wxTreeEvent

  A tree event holds information about events associated with wxTreeCtrl objects.

Variables

- wxEventType wxEVT_TREE_BEGIN_DRAG
- wxEventType wxEVT_TREE_BEGIN_RDRAG
- wxEventType wxEVT_TREE_BEGIN_LABEL_EDIT
- wxEventType wxEVT_TREE_END_LABEL_EDIT
- wxEventType wxEVT_TREE_DELETE_ITEM
- wxEventType wxEVT_TREE_GET_INFO
- wxEventType wxEVT_TREE_SET_INFO
- wxEventType wxEVT_TREE_ITEM_EXPANDED
- wxEventType wxEVT_TREE_ITEM_EXPANDING
- wxEventType wxEVT_TREE_ITEM_COLLAPSED
- wxEventType wxEVT_TREE_ITEM_COLLAPSING
- wxEventType wxEVT_TREE_SEL_CHANGED
- wxEventType wxEVT_TREE_SEL_CHANGING
- wxEventType wxEVT_TREE_KEY_DOWN
- wxEventType wxEVT_TREE_ITEM_ACTIVATED
- wxEventType wxEVT_TREE_ITEM_RIGHT_CLICK
- wxEventType wxEVT_TREE_ITEM_MIDDLE_CLICK
- wxEventType wxEVT_TREE_END_DRAG
- wxEventType wxEVT_TREE_STATE_IMAGE_CLICK
- wxEventType wxEVT_TREE_ITEM_GETTOOLTIP
- wxEventType wxEVT_TREE_ITEM_MENU

22.168.1 Variable Documentation

wxEventType wxEVT_TREE_BEGIN_DRAG

wxEventType wxEVT_TREE_BEGIN_LABEL_EDIT

wxEventType wxEVT_TREE_BEGIN_RDRAG

wxEventType wxEVT_TREE_DELETE_ITEM

wxEventType wxEVT_TREE_END_DRAG

wxEventType wxEVT_TREE_END_LABEL_EDIT

wxEventType wxEVT_TREE_GET_INFO

wxEventType wxEVT_TREE_ITEM_ACTIVATED

wxEventType wxEVT_TREE_ITEM_COLLAPSED

wxEventType wxEVT_TREE_ITEM_COLLAPSING

wxEventType wxEVT_TREE_ITEM_EXPANDED

wxEventType wxEVT_TREE_ITEM_EXPANDING

wxEventType wxEVT_TREE_ITEM_GETTOOLTIP

wxEventType wxEVT_TREE_ITEM_MENU

wxEventType wxEVT_TREE_ITEM_MIDDLE_CLICK

wxEventType wxEVT_TREE_ITEM_RIGHT_CLICK

wxEventType wxEVT_TREE_KEY_DOWN

wxEventType wxEVT_TREE_SEL_CHANGED

wxEventType wxEVT_TREE_SEL_CHANGING

wxEventType wxEVT_TREE_SET_INFO

wxEventType wxEVT_TREE_STATE_IMAGE_CLICK

22.169 docs/doxygen/overviews/unicode.h File Reference

22.170 docs/doxygen/overviews/unixprinting.h File Reference

22.171 docs/doxygen/overviews/validator.h File Reference

22.172 docs/doxygen/overviews/windowdeletion.h File Reference

22.173 docs/doxygen/overviews/windowids.h File Reference

22.174 docs/doxygen/overviews/windowsizing.h File Reference

22.175 docs/doxygen/overviews/windowstyles.h File Reference

22.176 docs/doxygen/overviews/xrc.h File Reference

22.177 docs/doxygen/overviews/xrc_format.h File Reference

22.178 interface/wx/aboutdlg.h File Reference

Classes

- class wxAboutDialogInfo

  wxAboutDialogInfo contains information shown in the standard About dialog displayed by the wxAboutBox() function.

Functions

- void wxAboutBox (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  This function shows the standard about dialog containing the information specified in info.
- void wxGenericAboutBox (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  This function does the same thing as wxAboutBox() except that it always uses the generic wxWidgets version of the dialog instead of the native one.

22.179 interface/wx/accel.h File Reference

Classes

- class wxAcceleratorEntry

  An object used by an application wishing to create an accelerator table (see wxAcceleratorTable).
- class wxAcceleratorTable

  An accelerator table allows the application to specify a table of keyboard shortcuts for menu or button commands.

Enumerations

- enum wxAcceleratorEntryFlags {
  wxACCEL_NORMAL,
  wxACCEL_ALT,
  wxACCEL_CTRL,
  wxACCEL_SHIFT,
  wxACCEL_RAW_CTRL,
  wxACCEL_CMD }

  wxAcceleratorEntry flags

Variables

- wxAcceleratorTable wxNullAcceleratorTable

  An empty accelerator table.

22.179.1 Enumeration Type Documentation

enum wxAcceleratorEntryFlags

wxAcceleratorEntry flags

Enumerator

wxACCEL_NORMAL no modifiers

wxACCEL_ALT hold Alt key down

wxACCEL_CTRL hold Ctrl key down, corresponds to Command key on OS X

wxACCEL_SHIFT hold Shift key down

wxACCEL_RAW_CTRL corresponds to real Ctrl key on OS X, identic to `wxACCEL_CTRL` on other platforms

wxACCEL_CMD deprecated, identic to `wxACCEL_CTRL` on all platforms.

22.179.2 Variable Documentation

wxAcceleratorTable wxNullAcceleratorTable

An empty accelerator table.

22.180 interface/wx/access.h File Reference

Classes

- class wxAccessible

  The wxAccessible class allows wxWidgets applications, and wxWidgets itself, to return extended information about user interface elements to client applications such as screen readers.

Macros

- #define wxACC_STATE_SYSTEM_ALERT_HIGH 0x00000001

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_ALERT_MEDIUM 0x00000002

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_ALERT_LOW 0x00000004

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_ANIMATED 0x00000008

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_BUSY 0x00000010

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_CHECKED 0x00000020

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_COLLAPSED 0x00000040

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_DEFAULT 0x00000080

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_EXPANDED 0x00000100

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_EXTSELECTABLE 0x00000200

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_FLOATING 0x00000400

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_FOCUSABLE 0x00000800

  Represents a status of the system.

- #define wxACC_STATE_SYSTEM_FOCUSED 0x00001000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_HOTTRACKED 0x00002000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_INVISIBLE 0x00004000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_MARQUEED 0x00008000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_MIXED 0x00010000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_MULTISELECTABLE 0x00020000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_OFFSCREEN 0x00040000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_PRESSED 0x00080000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_PROTECTED 0x00100000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_READONLY 0x00200000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_SELECTABLE 0x00400000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_SELECTED 0x00800000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_SELFVOICING 0x01000000

  Represents a status of the system.
- #define wxACC_STATE_SYSTEM_UNAVAILABLE 0x02000000

  Represents a status of the system.

- #define wxACC_EVENT_SYSTEM_SOUND 0x0001

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_ALERT 0x0002

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_FOREGROUND 0x0003

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MENUSTART 0x0004

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MENUEND 0x0005

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MENUPOPUPSTART 0x0006

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MENUPOPUPEND 0x0007

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_CAPTURESTART 0x0008

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_CAPTUREEND 0x0009

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MOVESIZESTART 0x000A

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MOVESIZEEND 0x000B

  An event identifier that can be sent via wxAccessible::NotifyEvent.

- #define wxACC_EVENT_SYSTEM_CONTEXTHELPSTART 0x000C

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_CONTEXTHELPEND 0x000D

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_DRAGDROPSTART 0x000E

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_DRAGDROPEND 0x000F

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_DIALOGSTART 0x0010

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_DIALOGEND 0x0011

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_SCROLLINGSTART 0x0012

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_SCROLLINGEND 0x0013

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_SWITCHSTART 0x0014

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_SWITCHEND 0x0015

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MINIMIZESTART 0x0016

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_SYSTEM_MINIMIZEEND 0x0017

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_CREATE 0x8000

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_DESTROY 0x8001

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_SHOW 0x8002

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_HIDE 0x8003

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_REORDER 0x8004

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_FOCUS 0x8005

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_SELECTION 0x8006

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_SELECTIONADD 0x8007

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_SELECTIONREMOVE 0x8008

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_SELECTIONWITHIN 0x8009

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_STATECHANGE 0x800A

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_LOCATIONCHANGE 0x800B

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_NAMECHANGE 0x800C

  An event identifier that can be sent via wxAccessible::NotifyEvent.
- #define wxACC_EVENT_OBJECT_DESCRIPTIONCHANGE 0x800D

An event identifier that can be sent via wxAccessible::NotifyEvent.

- #define wxACC_EVENT_OBJECT_VALUECHANGE 0x800E

  An event identifier that can be sent via wxAccessible::NotifyEvent.

- #define wxACC_EVENT_OBJECT_PARENTCHANGE 0x800F

  An event identifier that can be sent via wxAccessible::NotifyEvent.

- #define wxACC_EVENT_OBJECT_HELPCHANGE 0x8010

  An event identifier that can be sent via wxAccessible::NotifyEvent.

- #define wxACC_EVENT_OBJECT_DEFACTIONCHANGE 0x8011

  An event identifier that can be sent via wxAccessible::NotifyEvent.

- #define wxACC_EVENT_OBJECT_ACCELERATORCHANGE 0x8012

  An event identifier that can be sent via wxAccessible::NotifyEvent.

Enumerations

- enum wxAccStatus {
  wxACC_FAIL,
  wxACC_FALSE,
  wxACC_OK,
  wxACC_NOT_IMPLEMENTED,
  wxACC_NOT_SUPPORTED }

  wxAccessible functions return a wxAccStatus error code, which may be one of this enum's values.

- enum wxNavDir {
  wxNAVDIR_DOWN,
  wxNAVDIR_FIRSTCHILD,
  wxNAVDIR_LASTCHILD,
  wxNAVDIR_LEFT,
  wxNAVDIR_NEXT,
  wxNAVDIR_PREVIOUS,
  wxNAVDIR_RIGHT,
  wxNAVDIR_UP }

  Directions of navigation are represented by this enum.

- enum wxAccRole {
  wxROLE_NONE,
  wxROLE_SYSTEM_ALERT,
  wxROLE_SYSTEM_ANIMATION,
  wxROLE_SYSTEM_APPLICATION,
  wxROLE_SYSTEM_BORDER,
  wxROLE_SYSTEM_BUTTONDROPDOWN,
  wxROLE_SYSTEM_BUTTONDROPDOWNGRID,
  wxROLE_SYSTEM_BUTTONMENU,
  wxROLE_SYSTEM_CARET,
  wxROLE_SYSTEM_CELL,
  wxROLE_SYSTEM_CHARACTER,
  wxROLE_SYSTEM_CHART,
  wxROLE_SYSTEM_CHECKBUTTON,
  wxROLE_SYSTEM_CLIENT,
  wxROLE_SYSTEM_CLOCK,
  wxROLE_SYSTEM_COLUMN,
  wxROLE_SYSTEM_COLUMNHEADER,
  wxROLE_SYSTEM_COMBOBOX,
  wxROLE_SYSTEM_CURSOR,
  wxROLE_SYSTEM_DIAGRAM,
  wxROLE_SYSTEM_DIAL,
  wxROLE_SYSTEM_DIALOG,
  wxROLE_SYSTEM_DOCUMENT,
  wxROLE_SYSTEM_DROPLIST,
  wxROLE_SYSTEM_EQUATION,
  wxROLE_SYSTEM_GRAPHIC,
  wxROLE_SYSTEM_GRIP,
  wxROLE_SYSTEM_GROUPING,
  wxROLE_SYSTEM_HELPBALLOON,
  wxROLE_SYSTEM_HOTKEYFIELD,
  wxROLE_SYSTEM_INDICATOR,
  wxROLE_SYSTEM_LINK,
  wxROLE_SYSTEM_LIST,
  wxROLE_SYSTEM_LISTITEM,
  wxROLE_SYSTEM_MENUBAR,
  wxROLE_SYSTEM_MENUITEM,
  wxROLE_SYSTEM_MENUPOPUP,
  wxROLE_SYSTEM_OUTLINE,
  wxROLE_SYSTEM_OUTLINEITEM,
  wxROLE_SYSTEM_PAGETAB,
  wxROLE_SYSTEM_PAGETABLIST,
  wxROLE_SYSTEM_PANE,
  wxROLE_SYSTEM_PROGRESSBAR,
  wxROLE_SYSTEM_PROPERTYPAGE,
  wxROLE_SYSTEM_PUSHBUTTON,
  wxROLE_SYSTEM_RADIOBUTTON,
  wxROLE_SYSTEM_ROW,
  wxROLE_SYSTEM_ROWHEADER,
  wxROLE_SYSTEM_SCROLLBAR,
  wxROLE_SYSTEM_SEPARATOR,
  wxROLE_SYSTEM_SLIDER,
  wxROLE_SYSTEM_SOUND,
  wxROLE_SYSTEM_SPINBUTTON,
  wxROLE_SYSTEM_STATICTEXT,
  wxROLE_SYSTEM_STATUSBAR,
  wxROLE_SYSTEM_TABLE,
  wxROLE_SYSTEM_TEXT,
  wxROLE_SYSTEM_TITLEBAR,
  wxROLE_SYSTEM_TOOLBAR,
  wxROLE_SYSTEM_TOOLTIP,
  wxROLE_SYSTEM_WHITESPACE,
  wxROLE_SYSTEM_WINDOW }

The role of a user interface element is represented by the values of this enum.

- enum wxAccObject {
  wxOBJID_WINDOW = 0x00000000,
  wxOBJID_SYSMENU = 0xFFFFFFFF,
  wxOBJID_TITLEBAR = 0xFFFFFFFE,
  wxOBJID_MENU = 0xFFFFFFFD,
  wxOBJID_CLIENT = 0xFFFFFFFC,
  wxOBJID_VSCROLL = 0xFFFFFFFB,
  wxOBJID_HSCROLL = 0xFFFFFFFA,
  wxOBJID_SIZEGRIP = 0xFFFFFFF9,
  wxOBJID_CARET = 0xFFFFFFF8,
  wxOBJID_CURSOR = 0xFFFFFFF7,
  wxOBJID_ALERT = 0xFFFFFFF6,
  wxOBJID_SOUND = 0xFFFFFFF5 }

  Objects are represented by a wxAccObject enum value.
- enum wxAccSelectionFlags {
  wxACC_SEL_NONE = 0,
  wxACC_SEL_TAKEFOCUS = 1,
  wxACC_SEL_TAKESELECTION = 2,
  wxACC_SEL_EXTENDSELECTION = 4,
  wxACC_SEL_ADDSELECTION = 8,
  wxACC_SEL_REMOVESELECTION = 16 }

  Selection actions are identified by the wxAccSelectionFlags values.

22.180.1 Macro Definition Documentation

#define wxACC_EVENT_OBJECT_ACCELERATORCHANGE 0x8012

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_CREATE 0x8000

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_DEFACTIONCHANGE 0x8011

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_DESCRIPTIONCHANGE 0x800D

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_DESTROY 0x8001

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_FOCUS 0x8005

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_HELPCHANGE 0x8010

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_HIDE 0x8003

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_LOCATIONCHANGE 0x800B

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_NAMECHANGE 0x800C

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_PARENTCHANGE 0x800F

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_REORDER 0x8004

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_SELECTION 0x8006

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_SELECTIONADD 0x8007

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_SELECTIONREMOVE 0x8008

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_SELECTIONWITHIN 0x8009

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_SHOW 0x8002

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_STATECHANGE 0x800A

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_OBJECT_VALUECHANGE 0x800E

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_ALERT 0x0002

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_CAPTUREEND 0x0009

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_CAPTURESTART 0x0008

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_CONTEXTHELPEND 0x000D

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_CONTEXTHELPSTART 0x000C

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_DIALOGEND 0x0011

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_DIALOGSTART 0x0010

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_DRAGDROPEND 0x000F

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_DRAGDROPSTART 0x000E

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_FOREGROUND 0x0003

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MENUEND 0x0005

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MENUPOPUPEND 0x0007

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MENUPOPUPSTART 0x0006

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MENUSTART 0x0004

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MINIMIZEEND 0x0017

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MINIMIZESTART 0x0016

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MOVESIZEEND 0x000B

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_MOVESIZESTART 0x000A

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_SCROLLINGEND 0x0013

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_SCROLLINGSTART 0x0012

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_SOUND 0x0001

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_SWITCHEND 0x0015

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_EVENT_SYSTEM_SWITCHSTART 0x0014

An event identifier that can be sent via wxAccessible::NotifyEvent.

#define wxACC_STATE_SYSTEM_ALERT_HIGH 0x00000001

Represents a status of the system.

#define wxACC_STATE_SYSTEM_ALERT_LOW 0x00000004

Represents a status of the system.

#define wxACC_STATE_SYSTEM_ALERT_MEDIUM 0x00000002

Represents a status of the system.

#define wxACC_STATE_SYSTEM_ANIMATED 0x00000008

Represents a status of the system.

#define wxACC_STATE_SYSTEM_BUSY 0x00000010

Represents a status of the system.

#define wxACC_STATE_SYSTEM_CHECKED 0x00000020

Represents a status of the system.

#define wxACC_STATE_SYSTEM_COLLAPSED 0x00000040

Represents a status of the system.

#define wxACC_STATE_SYSTEM_DEFAULT 0x00000080

Represents a status of the system.

#define wxACC_STATE_SYSTEM_EXPANDED 0x00000100

Represents a status of the system.

#define wxACC_STATE_SYSTEM_EXTSELECTABLE 0x00000200

Represents a status of the system.

#define wxACC_STATE_SYSTEM_FLOATING 0x00000400

Represents a status of the system.

#define wxACC_STATE_SYSTEM_FOCUSABLE 0x00000800

Represents a status of the system.

#define wxACC_STATE_SYSTEM_FOCUSED 0x00001000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_HOTTRACKED 0x00002000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_INVISIBLE 0x00004000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_MARQUEED 0x00008000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_MIXED 0x00010000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_MULTISELECTABLE 0x00020000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_OFFSCREEN 0x00040000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_PRESSED 0x00080000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_PROTECTED 0x00100000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_READONLY 0x00200000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_SELECTABLE 0x00400000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_SELECTED 0x00800000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_SELFVOICING 0x01000000

Represents a status of the system.

#define wxACC_STATE_SYSTEM_UNAVAILABLE 0x02000000

Represents a status of the system.

22.180.2 Enumeration Type Documentation

enum wxAccObject

Objects are represented by a wxAccObject enum value.

Enumerator

wxOBJID_WINDOW
wxOBJID_SYSMENU
wxOBJID_TITLEBAR
wxOBJID_MENU
wxOBJID_CLIENT
wxOBJID_VSCROLL
wxOBJID_HSCROLL
wxOBJID_SIZEGRIP
wxOBJID_CARET
wxOBJID_CURSOR
wxOBJID_ALERT
wxOBJID_SOUND

enum wxAccRole

The role of a user interface element is represented by the values of this enum.

Enumerator

wxROLE_NONE
wxROLE_SYSTEM_ALERT
wxROLE_SYSTEM_ANIMATION
wxROLE_SYSTEM_APPLICATION
wxROLE_SYSTEM_BORDER
wxROLE_SYSTEM_BUTTONDROPDOWN
wxROLE_SYSTEM_BUTTONDROPDOWNGRID
wxROLE_SYSTEM_BUTTONMENU
wxROLE_SYSTEM_CARET
wxROLE_SYSTEM_CELL
wxROLE_SYSTEM_CHARACTER
wxROLE_SYSTEM_CHART
wxROLE_SYSTEM_CHECKBUTTON
wxROLE_SYSTEM_CLIENT
wxROLE_SYSTEM_CLOCK
wxROLE_SYSTEM_COLUMN
wxROLE_SYSTEM_COLUMNHEADER

wxROLE_SYSTEM_COMBOBOX
wxROLE_SYSTEM_CURSOR
wxROLE_SYSTEM_DIAGRAM
wxROLE_SYSTEM_DIAL
wxROLE_SYSTEM_DIALOG
wxROLE_SYSTEM_DOCUMENT
wxROLE_SYSTEM_DROPLIST
wxROLE_SYSTEM_EQUATION
wxROLE_SYSTEM_GRAPHIC
wxROLE_SYSTEM_GRIP
wxROLE_SYSTEM_GROUPING
wxROLE_SYSTEM_HELPBALLOON
wxROLE_SYSTEM_HOTKEYFIELD
wxROLE_SYSTEM_INDICATOR
wxROLE_SYSTEM_LINK
wxROLE_SYSTEM_LIST
wxROLE_SYSTEM_LISTITEM
wxROLE_SYSTEM_MENUBAR
wxROLE_SYSTEM_MENUITEM
wxROLE_SYSTEM_MENUPOPUP
wxROLE_SYSTEM_OUTLINE
wxROLE_SYSTEM_OUTLINEITEM
wxROLE_SYSTEM_PAGETAB
wxROLE_SYSTEM_PAGETABLIST
wxROLE_SYSTEM_PANE
wxROLE_SYSTEM_PROGRESSBAR
wxROLE_SYSTEM_PROPERTYPAGE
wxROLE_SYSTEM_PUSHBUTTON
wxROLE_SYSTEM_RADIOBUTTON
wxROLE_SYSTEM_ROW
wxROLE_SYSTEM_ROWHEADER
wxROLE_SYSTEM_SCROLLBAR
wxROLE_SYSTEM_SEPARATOR
wxROLE_SYSTEM_SLIDER
wxROLE_SYSTEM_SOUND
wxROLE_SYSTEM_SPINBUTTON
wxROLE_SYSTEM_STATICTEXT
wxROLE_SYSTEM_STATUSBAR
wxROLE_SYSTEM_TABLE
wxROLE_SYSTEM_TEXT
wxROLE_SYSTEM_TITLEBAR
wxROLE_SYSTEM_TOOLBAR
wxROLE_SYSTEM_TOOLTIP
wxROLE_SYSTEM_WHITESPACE
wxROLE_SYSTEM_WINDOW

enum wxAccSelectionFlags

Selection actions are identified by the wxAccSelectionFlags values.

Enumerator

wxACC_SEL_NONE

wxACC_SEL_TAKEFOCUS

wxACC_SEL_TAKESELECTION

wxACC_SEL_EXTENDSELECTION

wxACC_SEL_ADDSELECTION

wxACC_SEL_REMOVESELECTION

enum wxAccStatus

wxAccessible functions return a wxAccStatus error code, which may be one of this enum's values.

Enumerator

wxACC_FAIL The function failed.

wxACC_FALSE The function returned false.

wxACC_OK The function completed successfully.

wxACC_NOT_IMPLEMENTED The function is not implemented.

wxACC_NOT_SUPPORTED The function is not supported.

enum wxNavDir

Directions of navigation are represented by this enum.

Enumerator

wxNAVDIR_DOWN

wxNAVDIR_FIRSTCHILD

wxNAVDIR_LASTCHILD

wxNAVDIR_LEFT

wxNAVDIR_NEXT

wxNAVDIR_PREVIOUS

wxNAVDIR_RIGHT

wxNAVDIR_UP

22.181 interface/wx/affinematrix2d.h File Reference

Classes

- class wxAffineMatrix2D

  A 3x2 matrix representing an affine 2D transformation.

22.182 interface/wx/affinematrix2dbase.h File Reference

Classes

- class wxMatrix2D

  A simple container for 2x2 matrix.
- class wxAffineMatrix2DBase

  A 2x3 matrix representing an affine 2D transformation.

22.183 interface/wx/animate.h File Reference

Classes

- class wxAnimationCtrl

  This is a static control which displays an animation.
- class wxAnimation

  This class encapsulates the concept of a platform-dependent animation.

Macros

- #define wxAC_NO_AUTORESIZE (0x0010)
- #define wxAC_DEFAULT_STYLE (wxBORDER_NONE)

Enumerations

- enum wxAnimationType {
  wxANIMATION_TYPE_INVALID,
  wxANIMATION_TYPE_GIF,
  wxANIMATION_TYPE_ANI,
  wxANIMATION_TYPE_ANY }

  Supported animation types.

Variables

- wxAnimation wxNullAnimation

  An empty animation object.

22.183.1 Macro Definition Documentation

#define wxAC_DEFAULT_STYLE (wxBORDER_NONE)

#define wxAC_NO_AUTORESIZE (0x0010)

22.183.2 Enumeration Type Documentation

enum wxAnimationType

Supported animation types.

Enumerator

wxANIMATION_TYPE_INVALID

wxANIMATION_TYPE_GIF represents an animated GIF file.

wxANIMATION_TYPE_ANI represents an ANI file.

wxANIMATION_TYPE_ANY autodetect the filetype.

22.183.3 Variable Documentation

wxAnimation wxNullAnimation

An empty animation object.

22.184 interface/wx/any.h File Reference

Classes

- class wxAny

  The wxAny class represents a container for any type.
- union wxAnyValueBuffer

  Type for buffer within wxAny for holding data.
- class wxAnyValueType

  wxAnyValueType is base class for value type functionality for C++ data types used with wxAny.

Enumerations

- enum { WX_ANY_VALUE_BUFFER_SIZE = 16 }

  Size of the wxAny value buffer.

22.184.1 Enumeration Type Documentation

anonymous enum

Size of the wxAny value buffer.

Enumerator

WX_ANY_VALUE_BUFFER_SIZE

22.185 interface/wx/anybutton.h File Reference

Classes

- class wxAnyButton

  A class for common button functionality used as the base for the various button classes.

Macros

- #define wxBU_LEFT 0x0040
- #define wxBU_TOP 0x0080
- #define wxBU_RIGHT 0x0100
- #define wxBU_BOTTOM 0x0200

- #define wxBU_ALIGN_MASK (wxBU_LEFT | wxBU_TOP | wxBU_RIGHT | wxBU_BOTTOM)
- #define wxBU_EXACTFIT 0x0001
- #define wxBU_NOTEXT 0x0002
- #define wxBU_AUTODRAW 0x0004

22.185.1 Macro Definition Documentation

#define wxBU_ALIGN_MASK (wxBU_LEFT | wxBU_TOP | wxBU_RIGHT | wxBU_BOTTOM)

#define wxBU_AUTODRAW 0x0004

#define wxBU_BOTTOM 0x0200

#define wxBU_EXACTFIT 0x0001

#define wxBU_LEFT 0x0040

#define wxBU_NOTEXT 0x0002

#define wxBU_RIGHT 0x0100

#define wxBU_TOP 0x0080

22.186 interface/wx/appprogress.h File Reference

Classes

- class wxAppProgressIndicator

  A helper class that can be used to update the progress bar in the taskbar button.

22.187 interface/wx/apptrait.h File Reference

Classes

- class wxAppTraits

  The wxAppTraits class defines various configurable aspects of a wxApp.

22.188 interface/wx/arrstr.h File Reference

Classes

- class wxArrayString

  wxArrayString is an efficient container for storing wxString objects.
- class wxSortedArrayString

  wxSortedArrayString is an efficient container for storing wxString objects which always keeps the string in alphabetical order.

Functions

- int wxStringSortAscending (const wxString &s1, const wxString &s2)

  Comparison function comparing strings in alphabetical order.
- int wxStringSortDescending (const wxString &s1, const wxString &s2)

  Comparison function comparing strings in reverse alphabetical order.
- int wxDictionaryStringSortAscending (const wxString &s1, const wxString &s2)

  Comparison function comparing strings in dictionary order.
- wxArrayString wxSplit (const wxString &str, const wxChar sep, const wxChar escape= '\\')

  Splits the given wxString object using the separator sep and returns the result as a wxArrayString.
- wxString wxJoin (const wxArrayString &arr, const wxChar sep, const wxChar escape= '\\')

  Concatenate all lines of the given wxArrayString object using the separator sep and returns the result as a wxString.

22.188.1 Function Documentation

int wxDictionaryStringSortAscending (const wxString & *s1*, const wxString & *s2*)

Comparison function comparing strings in dictionary order.

Comparison function comparing strings in reverse dictionary order.

The "dictionary order" differs from the alphabetical order in that the strings differing not only in case are compared case-insensitively to ensure that "Aa" comes before "AB" in the sorted array, unlike with wxStringSortAscending().

This function can be used with wxSortedArrayString::Sort() or passed as an argument to wxSortedArrayString constructor.

See also

wxStringSortAscending(), wxDictionaryStringSortDescending()

Since

3.1.0

See wxDictionaryStringSortAscending() for the dictionary sort description.

See also

wxStringSortDescending()

Since

3.1.0

int wxStringSortAscending (const wxString & *s1*, const wxString & *s2*)

Comparison function comparing strings in alphabetical order.

This function can be used with wxSortedArrayString::Sort() or passed as an argument to wxSortedArrayString constructor.

See also

wxStringSortDescending(), wxDictionaryStringSortAscending()

Since

3.1.0

int wxStringSortDescending (const wxString & *s1*, const wxString & *s2*)

Comparison function comparing strings in reverse alphabetical order.

This function can be used with wxSortedArrayString::Sort() or passed as an argument to wxSortedArrayString constructor.

See also

wxStringSortAscending(), wxDictionaryStringSortAscending()

Since

3.1.0

22.189 interface/wx/artprov.h File Reference

Classes

- class wxArtProvider

  wxArtProvider class is used to customize the look of wxWidgets application.

Typedefs

- typedef wxString wxArtClient

  This type identifies the client of the art objects requested to wxArtProvider.
- typedef wxString wxArtID

  This type identifies a specific art object which can be requested to wxArtProvider.

Variables

- wxArtClient wxART_TOOLBAR
- wxArtClient wxART_MENU
- wxArtClient wxART_FRAME_ICON
- wxArtClient wxART_CMN_DIALOG
- wxArtClient wxART_HELP_BROWSER
- wxArtClient wxART_MESSAGE_BOX
- wxArtClient wxART_BUTTON
- wxArtClient wxART_LIST
- wxArtClient wxART_OTHER
- wxArtID wxART_ADD_BOOKMARK
- wxArtID wxART_DEL_BOOKMARK
- wxArtID wxART_HELP_SIDE_PANEL
- wxArtID wxART_HELP_SETTINGS
- wxArtID wxART_HELP_BOOK
- wxArtID wxART_HELP_FOLDER
- wxArtID wxART_HELP_PAGE
- wxArtID wxART_GO_BACK
- wxArtID wxART_GO_FORWARD
- wxArtID wxART_GO_UP
- wxArtID wxART_GO_DOWN
- wxArtID wxART_GO_TO_PARENT
- wxArtID wxART_GO_HOME

- wxArtID wxART_GOTO_FIRST
- wxArtID wxART_GOTO_LAST
- wxArtID wxART_FILE_OPEN
- wxArtID wxART_FILE_SAVE
- wxArtID wxART_FILE_SAVE_AS
- wxArtID wxART_PRINT
- wxArtID wxART_HELP
- wxArtID wxART_TIP
- wxArtID wxART_REPORT_VIEW
- wxArtID wxART_LIST_VIEW
- wxArtID wxART_NEW_DIR
- wxArtID wxART_HARDDISK
- wxArtID wxART_FLOPPY
- wxArtID wxART_CDROM
- wxArtID wxART_REMOVABLE
- wxArtID wxART_FOLDER
- wxArtID wxART_FOLDER_OPEN
- wxArtID wxART_GO_DIR_UP
- wxArtID wxART_EXECUTABLE_FILE
- wxArtID wxART_NORMAL_FILE
- wxArtID wxART_TICK_MARK
- wxArtID wxART_CROSS_MARK
- wxArtID wxART_ERROR
- wxArtID wxART_QUESTION
- wxArtID wxART_WARNING
- wxArtID wxART_INFORMATION
- wxArtID wxART_MISSING_IMAGE
- wxArtID wxART_COPY
- wxArtID wxART_CUT
- wxArtID wxART_PASTE
- wxArtID wxART_DELETE
- wxArtID wxART_NEW
- wxArtID wxART_UNDO
- wxArtID wxART_REDO
- wxArtID wxART_PLUS
- wxArtID wxART_MINUS
- wxArtID wxART_CLOSE
- wxArtID wxART_QUIT
- wxArtID wxART_FIND
- wxArtID wxART_FIND_AND_REPLACE
- wxArtID wxART_FULL_SCREEN

22.189.1 Typedef Documentation

typedef wxString wxArtClient

This type identifies the client of the art objects requested to wxArtProvider.

typedef wxString wxArtID

This type identifies a specific art object which can be requested to wxArtProvider.

22.189.2 Variable Documentation

wxArtID wxART_ADD_BOOKMARK

wxArtClient wxART_BUTTON

wxArtID wxART_CDROM

wxArtID wxART_CLOSE

wxArtClient wxART_CMN_DIALOG

wxArtID wxART_COPY

wxArtID wxART_CROSS_MARK

wxArtID wxART_CUT

wxArtID wxART_DEL_BOOKMARK

wxArtID wxART_DELETE

wxArtID wxART_ERROR

wxArtID wxART_EXECUTABLE_FILE

wxArtID wxART_FILE_OPEN

wxArtID wxART_FILE_SAVE

wxArtID wxART_FILE_SAVE_AS

wxArtID wxART_FIND

wxArtID wxART_FIND_AND_REPLACE

wxArtID wxART_FLOPPY

wxArtID wxART_FOLDER

wxArtID wxART_FOLDER_OPEN

wxArtClient wxART_FRAME_ICON

wxArtID wxART_FULL_SCREEN

wxArtID wxART_GO_BACK

wxArtID wxART_GO_DIR_UP

wxArtID wxART_GO_DOWN

wxArtID wxART_GO_FORWARD

wxArtID wxART_GO_HOME

wxArtID wxART_GO_TO_PARENT

wxArtID wxART_GO_UP

wxArtID wxART_GOTO_FIRST

wxArtID wxART_GOTO_LAST

wxArtID wxART_HARDDISK

wxArtID wxART_HELP

wxArtID wxART_HELP_BOOK

wxArtClient wxART_HELP_BROWSER

wxArtID wxART_HELP_FOLDER

wxArtID wxART_HELP_PAGE

wxArtID wxART_HELP_SETTINGS

wxArtID wxART_HELP_SIDE_PANEL

wxArtID wxART_INFORMATION

wxArtClient wxART_LIST

wxArtID wxART_LIST_VIEW

wxArtClient wxART_MENU

wxArtClient wxART_MESSAGE_BOX

wxArtID wxART_MINUS

wxArtID wxART_MISSING_IMAGE

wxArtID wxART_NEW

wxArtID wxART_NEW_DIR

wxArtID wxART_NORMAL_FILE

wxArtClient wxART_OTHER

wxArtID wxART_PASTE

wxArtID wxART_PLUS

wxArtID wxART_PRINT

wxArtID wxART_QUESTION

wxArtID wxART_QUIT

wxArtID wxART_REDO

wxArtID wxART_REMOVABLE

wxArtID wxART_REPORT_VIEW

wxArtID wxART_TICK_MARK

wxArtID wxART_TIP

wxArtClient wxART_TOOLBAR

wxArtID wxART_UNDO

wxArtID wxART_WARNING

22.190 interface/wx/atomic.h File Reference

Functions

- void wxAtomicInc (wxAtomicInt &value)

  This function increments value in an atomic manner.

- wxInt32 wxAtomicDec (wxAtomicInt &value)

  This function decrements value in an atomic manner.

22.191 interface/wx/aui/auibar.h File Reference

Classes

- class wxAuiToolBarEvent

  wxAuiToolBarEvent is used for the events generated by wxAuiToolBar.

- class wxAuiToolBarItem

  wxAuiToolBarItem is part of the wxAUI class framework, representing a toolbar element.

- class wxAuiToolBarArt

  wxAuiToolBarArt is part of the wxAUI class framework.

- class wxAuiDefaultToolBarArt

  wxAuiDefaultToolBarArt is part of the wxAUI class framework.

- class wxAuiToolBar

  wxAuiToolBar is a dockable toolbar, part of the wxAUI class framework.

Enumerations

- enum wxAuiToolBarStyle {
  wxAUI_TB_TEXT = 1 << 0,
  wxAUI_TB_NO_TOOLTIPS = 1 << 1,
  wxAUI_TB_NO_AUTORESIZE = 1 << 2,
  wxAUI_TB_GRIPPER = 1 << 3,
  wxAUI_TB_OVERFLOW = 1 << 4,
  wxAUI_TB_VERTICAL = 1 << 5,
  wxAUI_TB_HORZ_LAYOUT = 1 << 6,
  wxAUI_TB_HORIZONTAL = 1 << 7,
  wxAUI_TB_PLAIN_BACKGROUND = 1 << 8,
  wxAUI_TB_HORZ_TEXT = (wxAUI_TB_HORZ_LAYOUT | wxAUI_TB_TEXT),
  wxAUI_ORIENTATION_MASK = (wxAUI_TB_VERTICAL | wxAUI_TB_HORIZONTAL),
  wxAUI_TB_DEFAULT_STYLE = 0 }

  wxAuiToolBarStyle is part of the wxAUI class framework, used to define the appearance of a wxAuiToolBar.
- enum wxAuiToolBarArtSetting {
  wxAUI_TBART_SEPARATOR_SIZE = 0,
  wxAUI_TBART_GRIPPER_SIZE = 1,
  wxAUI_TBART_OVERFLOW_SIZE = 2 }

  wxAuiToolBarArtSetting
- enum wxAuiToolBarToolTextOrientation {
  wxAUI_TBTOOL_TEXT_LEFT = 0,
  wxAUI_TBTOOL_TEXT_RIGHT = 1,
  wxAUI_TBTOOL_TEXT_TOP = 2,
  wxAUI_TBTOOL_TEXT_BOTTOM = 3 }

  wxAuiToolBarToolTextOrientation

22.192 interface/wx/aui/auibook.h File Reference

Classes

- class wxAuiNotebook

  wxAuiNotebook is part of the wxAUI class framework, which represents a notebook control, managing multiple windows with associated tabs.
- class wxAuiTabContainerButton

  A simple class which holds information about wxAuiNotebook tab buttons and their state.
- class wxAuiTabContainer

  wxAuiTabContainer is a class which contains information about each tab.
- class wxAuiTabArt

  Tab art provider defines all the drawing functions used by wxAuiNotebook.
- class wxAuiNotebookEvent

  This class is used by the events generated by wxAuiNotebook.
- class wxAuiDefaultTabArt

  Default art provider for wxAuiNotebook.
- class wxAuiSimpleTabArt

  Another standard tab art provider for wxAuiNotebook.

Variables

- wxEventType wxEVT_AUINOTEBOOK_PAGE_CLOSE
- wxEventType wxEVT_AUINOTEBOOK_PAGE_CHANGED
- wxEventType wxEVT_AUINOTEBOOK_PAGE_CHANGING

- wxEventType wxEVT_AUINOTEBOOK_PAGE_CLOSED
- wxEventType wxEVT_AUINOTEBOOK_BUTTON
- wxEventType wxEVT_AUINOTEBOOK_BEGIN_DRAG
- wxEventType wxEVT_AUINOTEBOOK_END_DRAG
- wxEventType wxEVT_AUINOTEBOOK_DRAG_MOTION
- wxEventType wxEVT_AUINOTEBOOK_ALLOW_DND
- wxEventType wxEVT_AUINOTEBOOK_TAB_MIDDLE_DOWN
- wxEventType wxEVT_AUINOTEBOOK_TAB_MIDDLE_UP
- wxEventType wxEVT_AUINOTEBOOK_TAB_RIGHT_DOWN
- wxEventType wxEVT_AUINOTEBOOK_TAB_RIGHT_UP
- wxEventType wxEVT_AUINOTEBOOK_DRAG_DONE
- wxEventType wxEVT_AUINOTEBOOK_BG_DCLICK

22.192.1 Variable Documentation

wxEventType wxEVT_AUINOTEBOOK_ALLOW_DND

wxEventType wxEVT_AUINOTEBOOK_BEGIN_DRAG

wxEventType wxEVT_AUINOTEBOOK_BG_DCLICK

wxEventType wxEVT_AUINOTEBOOK_BUTTON

wxEventType wxEVT_AUINOTEBOOK_DRAG_DONE

wxEventType wxEVT_AUINOTEBOOK_DRAG_MOTION

wxEventType wxEVT_AUINOTEBOOK_END_DRAG

wxEventType wxEVT_AUINOTEBOOK_PAGE_CHANGED

wxEventType wxEVT_AUINOTEBOOK_PAGE_CHANGING

wxEventType wxEVT_AUINOTEBOOK_PAGE_CLOSE

wxEventType wxEVT_AUINOTEBOOK_PAGE_CLOSED

wxEventType wxEVT_AUINOTEBOOK_TAB_MIDDLE_DOWN

wxEventType wxEVT_AUINOTEBOOK_TAB_MIDDLE_UP

wxEventType wxEVT_AUINOTEBOOK_TAB_RIGHT_DOWN

wxEventType wxEVT_AUINOTEBOOK_TAB_RIGHT_UP

22.193 interface/wx/aui/dockart.h File Reference

Classes

- class wxAuiDockArt

  wxAuiDockArt is part of the wxAUI class framework.

Enumerations

- enum wxAuiPaneDockArtSetting {
  wxAUI_DOCKART_SASH_SIZE = 0,
  wxAUI_DOCKART_CAPTION_SIZE = 1,
  wxAUI_DOCKART_GRIPPER_SIZE = 2,
  wxAUI_DOCKART_PANE_BORDER_SIZE = 3,
  wxAUI_DOCKART_PANE_BUTTON_SIZE = 4,
  wxAUI_DOCKART_BACKGROUND_COLOUR = 5,
  wxAUI_DOCKART_SASH_COLOUR = 6,
  wxAUI_DOCKART_ACTIVE_CAPTION_COLOUR = 7,
  wxAUI_DOCKART_ACTIVE_CAPTION_GRADIENT_COLOUR = 8,
  wxAUI_DOCKART_INACTIVE_CAPTION_COLOUR = 9,
  wxAUI_DOCKART_INACTIVE_CAPTION_GRADIENT_COLOUR = 10,
  wxAUI_DOCKART_ACTIVE_CAPTION_TEXT_COLOUR = 11,
  wxAUI_DOCKART_INACTIVE_CAPTION_TEXT_COLOUR = 12,
  wxAUI_DOCKART_BORDER_COLOUR = 13,
  wxAUI_DOCKART_GRIPPER_COLOUR = 14,
  wxAUI_DOCKART_CAPTION_FONT = 15,
  wxAUI_DOCKART_GRADIENT_TYPE = 16 }

  These are the possible pane dock art settings for wxAuiDefaultDockArt.
- enum wxAuiPaneDockArtGradients {
  wxAUI_GRADIENT_NONE = 0,
  wxAUI_GRADIENT_VERTICAL = 1,
  wxAUI_GRADIENT_HORIZONTAL = 2 }

  These are the possible gradient dock art settings for wxAuiDefaultDockArt.
- enum wxAuiPaneButtonState {
  wxAUI_BUTTON_STATE_NORMAL = 0,
  wxAUI_BUTTON_STATE_HOVER = 1 << 1,
  wxAUI_BUTTON_STATE_PRESSED = 1 << 2,
  wxAUI_BUTTON_STATE_DISABLED = 1 << 3,
  wxAUI_BUTTON_STATE_HIDDEN = 1 << 4,
  wxAUI_BUTTON_STATE_CHECKED = 1 << 5 }

  These are the possible pane button / wxAuiNotebook button / wxAuiToolBar button states.
- enum wxAuiButtonId {
  wxAUI_BUTTON_CLOSE = 101,
  wxAUI_BUTTON_MAXIMIZE_RESTORE = 102,
  wxAUI_BUTTON_MINIMIZE = 103,
  wxAUI_BUTTON_PIN = 104,
  wxAUI_BUTTON_OPTIONS = 105,
  wxAUI_BUTTON_WINDOWLIST = 106,
  wxAUI_BUTTON_LEFT = 107,
  wxAUI_BUTTON_RIGHT = 108,
  wxAUI_BUTTON_UP = 109,
  wxAUI_BUTTON_DOWN = 110,
  wxAUI_BUTTON_CUSTOM1 = 201,
  wxAUI_BUTTON_CUSTOM2 = 202,
  wxAUI_BUTTON_CUSTOM3 = 203 }

  These are the possible pane button / wxAuiNotebook button / wxAuiToolBar button identifiers.

22.193.1 Enumeration Type Documentation

enum wxAuiButtonId

These are the possible pane button / wxAuiNotebook button / wxAuiToolBar button identifiers.

Enumerator

wxAUI_BUTTON_CLOSE Shows a close button on the pane.

wxAUI_BUTTON_MAXIMIZE_RESTORE Shows a maximize/restore button on the pane.

wxAUI_BUTTON_MINIMIZE Shows a minimize button on the pane.

wxAUI_BUTTON_PIN Shows a pin button on the pane.

wxAUI_BUTTON_OPTIONS Shows an option button on the pane (not implemented)

wxAUI_BUTTON_WINDOWLIST Shows a window list button on the pane (for wxAuiNotebook)

wxAUI_BUTTON_LEFT Shows a left button on the pane (for wxAuiNotebook)

wxAUI_BUTTON_RIGHT Shows a right button on the pane (for wxAuiNotebook)

wxAUI_BUTTON_UP Shows an up button on the pane (not implemented)

wxAUI_BUTTON_DOWN Shows a down button on the pane (not implemented)

wxAUI_BUTTON_CUSTOM1 Shows one of three possible custom buttons on the pane (not implemented)

wxAUI_BUTTON_CUSTOM2 Shows one of three possible custom buttons on the pane (not implemented)

wxAUI_BUTTON_CUSTOM3 Shows one of three possible custom buttons on the pane (not implemented)

enum wxAuiPaneButtonState

These are the possible pane button / wxAuiNotebook button / wxAuiToolBar button states.

Enumerator

wxAUI_BUTTON_STATE_NORMAL Normal button state.

wxAUI_BUTTON_STATE_HOVER Hovered button state.

wxAUI_BUTTON_STATE_PRESSED Pressed button state.

wxAUI_BUTTON_STATE_DISABLED Disabled button state.

wxAUI_BUTTON_STATE_HIDDEN Hidden button state.

wxAUI_BUTTON_STATE_CHECKED Checked button state.

enum wxAuiPaneDockArtGradients

These are the possible gradient dock art settings for wxAuiDefaultDockArt.

Enumerator

wxAUI_GRADIENT_NONE No gradient on the captions, in other words a solid colour.

wxAUI_GRADIENT_VERTICAL Vertical gradient on the captions, in other words a gradal change in colours from top to bottom.

wxAUI_GRADIENT_HORIZONTAL Horizontal gradient on the captions, in other words a gradual change in colours from left to right.

22.194 interface/wx/aui/framemanager.h File Reference

Classes

- class wxAuiManager

  wxAuiManager is the central class of the wxAUI class framework.
- class wxAuiPaneInfo

  wxAuiPaneInfo is part of the wxAUI class framework.
- class wxAuiManagerEvent

  Event used to indicate various actions taken with wxAuiManager.

Enumerations

- enum wxAuiManagerDock {
  wxAUI_DOCK_NONE = 0,
  wxAUI_DOCK_TOP = 1,
  wxAUI_DOCK_RIGHT = 2,
  wxAUI_DOCK_BOTTOM = 3,
  wxAUI_DOCK_LEFT = 4,
  wxAUI_DOCK_CENTER = 5,
  wxAUI_DOCK_CENTRE = wxAUI_DOCK_CENTER }
- enum wxAuiManagerOption {
  wxAUI_MGR_ALLOW_FLOATING = 1 << 0,
  wxAUI_MGR_ALLOW_ACTIVE_PANE = 1 << 1,
  wxAUI_MGR_TRANSPARENT_DRAG = 1 << 2,
  wxAUI_MGR_TRANSPARENT_HINT = 1 << 3,
  wxAUI_MGR_VENETIAN_BLINDS_HINT = 1 << 4,
  wxAUI_MGR_RECTANGLE_HINT = 1 << 5,
  wxAUI_MGR_HINT_FADE = 1 << 6,
  wxAUI_MGR_NO_VENETIAN_BLINDS_FADE = 1 << 7,
  wxAUI_MGR_LIVE_RESIZE = 1 << 8,
  wxAUI_MGR_DEFAULT }

  wxAuiManager behaviour and visual effects style flags.

22.194.1 Enumeration Type Documentation

enum wxAuiManagerDock

Todo wxAuiPaneInfo dock direction types used with wxAuiManager.

Enumerator

wxAUI_DOCK_NONE

wxAUI_DOCK_TOP

wxAUI_DOCK_RIGHT

wxAUI_DOCK_BOTTOM

wxAUI_DOCK_LEFT

wxAUI_DOCK_CENTER

wxAUI_DOCK_CENTRE

enum wxAuiManagerOption

wxAuiManager behaviour and visual effects style flags.

Enumerator

wxAUI_MGR_ALLOW_FLOATING Allow a pane to be undocked to take the form of a wxMiniFrame.

wxAUI_MGR_ALLOW_ACTIVE_PANE Change the color of the title bar of the pane when it is activated.

wxAUI_MGR_TRANSPARENT_DRAG Make the pane transparent during its movement.

wxAUI_MGR_TRANSPARENT_HINT The possible location for docking is indicated by a translucent area.

wxAUI_MGR_VENETIAN_BLINDS_HINT The possible location for docking is indicated by a gradually appearing partially transparent area.

wxAUI_MGR_RECTANGLE_HINT The possible location for docking is indicated by a rectangular outline.

wxAUI_MGR_HINT_FADE The translucent area where the pane could be docked appears gradually.

wxAUI_MGR_NO_VENETIAN_BLINDS_FADE Used in complement of wxAUI_MGR_VENETIAN_BLINDS_HINT to show the hint immediately.

wxAUI_MGR_LIVE_RESIZE When a docked pane is resized, its content is refreshed in live (instead of moving the border alone and refreshing the content at the end).

wxAUI_MGR_DEFAULT Default behavior.

22.195 interface/wx/bannerwindow.h File Reference

Classes

- class wxBannerWindow

  A simple banner window showing either a bitmap or text.

22.196 interface/wx/base64.h File Reference

Enumerations

- enum wxBase64DecodeMode {
  wxBase64DecodeMode_Strict,
  wxBase64DecodeMode_SkipWS,
  wxBase64DecodeMode_Relaxed }

  Elements of this enum specify the possible behaviours of wxBase64Decode when an invalid character is encountered.

Functions

- size_t wxBase64Encode (char *dst, size_t dstLen, const void *src, size_t srcLen)

  This function encodes the given data using base64.
- wxString wxBase64Encode (const void *src, size_t srcLen)

  This function encodes the given data using base64 and returns the output as a wxString.
- wxString wxBase64Encode (const wxMemoryBuffer &buf)

  This function encodes the given data using base64 and returns the output as a wxString.
- size_t wxBase64DecodedSize (size_t srcLen)

  Returns the size of the buffer necessary to contain the data encoded in a base64 string of length srcLen.
- size_t wxBase64EncodedSize (size_t len)

  Returns the length of the string with base64 representation of a buffer of specified size len.
- size_t wxBase64Decode (void *dst, size_t dstLen, const char *src, size_t srcLen=wxNO_LEN, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  This function decodes a Base64-encoded string.
- size_t wxBase64Decode (void *dst, size_t dstLen, const wxString &str, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  Decode a Base64-encoded wxString.
- wxMemoryBuffer wxBase64Decode (const char *src, size_t srcLen=wxNO_LEN, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  Decode a Base64-encoded string and return decoded contents in a buffer.
- wxMemoryBuffer wxBase64Decode (const wxString &src, wxBase64DecodeMode mode=wxBase64DecodeMode_Strict, size_t *posErr=NULL)

  Decode a Base64-encoded wxString and return decoded contents in a buffer.

22.197 interface/wx/bmpbuttn.h File Reference

Classes

- class wxBitmapButton

  A bitmap button is a control that contains a bitmap.

22.198 interface/wx/bmpcbox.h File Reference

Classes

- class wxBitmapComboBox

  A combobox that displays bitmap in front of the list items.

22.199 interface/wx/brush.h File Reference

Classes

- class wxBrush

  A brush is a drawing tool for filling in areas.
- class wxBrushList

  A brush list is a list containing all brushes which have been created.

Enumerations

- enum wxBrushStyle {
  wxBRUSHSTYLE_INVALID = -1,
  wxBRUSHSTYLE_SOLID = wxSOLID,
  wxBRUSHSTYLE_TRANSPARENT = wxTRANSPARENT,
  wxBRUSHSTYLE_STIPPLE_MASK_OPAQUE = wxSTIPPLE_MASK_OPAQUE,
  wxBRUSHSTYLE_STIPPLE_MASK = wxSTIPPLE_MASK,
  wxBRUSHSTYLE_STIPPLE = wxSTIPPLE,
  wxBRUSHSTYLE_BDIAGONAL_HATCH,
  wxBRUSHSTYLE_CROSSDIAG_HATCH,
  wxBRUSHSTYLE_FDIAGONAL_HATCH,
  wxBRUSHSTYLE_CROSS_HATCH,
  wxBRUSHSTYLE_HORIZONTAL_HATCH,
  wxBRUSHSTYLE_VERTICAL_HATCH,
  wxBRUSHSTYLE_FIRST_HATCH,
  wxBRUSHSTYLE_LAST_HATCH }

  The possible brush styles.

Variables

- wxBrush wxNullBrush

  An empty brush.
- wxBrush * wxBLUE_BRUSH

  Blue brush.
- wxBrush * wxGREEN_BRUSH

  Green brush.

- wxBrush * wxYELLOW_BRUSH
  Yellow brush.
- wxBrush * wxWHITE_BRUSH
  White brush.
- wxBrush * wxBLACK_BRUSH
  Black brush.
- wxBrush * wxGREY_BRUSH
  Grey brush.
- wxBrush * wxMEDIUM_GREY_BRUSH
  Medium grey brush.
- wxBrush * wxLIGHT_GREY_BRUSH
  Light grey brush.
- wxBrush * wxTRANSPARENT_BRUSH
  Transparent brush.
- wxBrush * wxCYAN_BRUSH
  Cyan brush.
- wxBrush * wxRED_BRUSH
  Red brush.
- wxBrushList * wxTheBrushList
  The global wxBrushList instance.

22.199.1 Enumeration Type Documentation

enum wxBrushStyle

The possible brush styles.

Enumerator

wxBRUSHSTYLE_INVALID

wxBRUSHSTYLE_SOLID Solid.

wxBRUSHSTYLE_TRANSPARENT Transparent (no fill).

wxBRUSHSTYLE_STIPPLE_MASK_OPAQUE Uses a bitmap as a stipple; the mask is used for blitting monochrome using text foreground and background colors.

wxBRUSHSTYLE_STIPPLE_MASK Uses a bitmap as a stipple; mask is used for masking areas in the stipple bitmap.

wxBRUSHSTYLE_STIPPLE Uses a bitmap as a stipple.

wxBRUSHSTYLE_BDIAGONAL_HATCH Backward diagonal hatch.

wxBRUSHSTYLE_CROSSDIAG_HATCH Cross-diagonal hatch.

wxBRUSHSTYLE_FDIAGONAL_HATCH Forward diagonal hatch.

wxBRUSHSTYLE_CROSS_HATCH Cross hatch.

wxBRUSHSTYLE_HORIZONTAL_HATCH Horizontal hatch.

wxBRUSHSTYLE_VERTICAL_HATCH Vertical hatch.

wxBRUSHSTYLE_FIRST_HATCH First of the hatch styles (inclusive).

wxBRUSHSTYLE_LAST_HATCH Last of the hatch styles (inclusive).

22.199.2 Variable Documentation

wxBrush* wxBLACK_BRUSH

Black brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxBLUE_BRUSH

Blue brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxCYAN_BRUSH

Cyan brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxGREEN_BRUSH

Green brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxGREY_BRUSH

Grey brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxLIGHT_GREY_BRUSH

Light grey brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxMEDIUM_GREY_BRUSH

Medium grey brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush wxNullBrush

An empty brush.

wxBrush::IsOk() always returns false for this object.

wxBrush* wxRED_BRUSH

Red brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrushList* wxTheBrushList

The global wxBrushList instance.

wxBrush* wxTRANSPARENT_BRUSH

Transparent brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxWHITE_BRUSH

White brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

wxBrush* wxYELLOW_BRUSH

Yellow brush.

Except for the color it has all standard attributes (`wxBRUSHSTYLE_SOLID`, no stipple bitmap, etc...).

22.200 interface/wx/buffer.h File Reference

Classes

- class wxScopedCharTypeBuffer< T >

  wxScopedCharTypeBuffer<T> is a template class for storing characters.
- class wxCharTypeBuffer< T >

  wxCharTypeBuffer<T> is a template class for storing characters.
- class wxCharBuffer

  This is a specialization of wxCharTypeBuffer<T> for `char` type.
- class wxWCharBuffer

  This is a specialization of wxCharTypeBuffer<T> for `wchar_t` type.
- class wxMemoryBuffer

  *A **wxMemoryBuffer** is a useful data structure for storing arbitrary sized blocks of memory.*

Typedefs

- typedef wxScopedCharTypeBuffer < char > wxScopedCharBuffer

  Scoped char buffer.
- typedef wxScopedCharTypeBuffer < wchar_t > wxScopedWCharBuffer

  Scoped wchar_t buffer.

22.200.1 Typedef Documentation

typedef wxScopedCharTypeBuffer<char> wxScopedCharBuffer

Scoped char buffer.

typedef wxScopedCharTypeBuffer<wchar_t> wxScopedWCharBuffer

Scoped wchar_t buffer.

22.201 interface/wx/busyinfo.h File Reference

Classes

- class wxBusyInfo

  This class makes it easy to tell your user that the program is temporarily busy.
- class wxBusyInfoFlags

  Parameters for wxBusyInfo.

22.202 interface/wx/button.h File Reference

Classes

- class wxButton

  A button is a control that contains a text string, and is one of the most common elements of a GUI.

22.203 interface/wx/calctrl.h File Reference

Classes

- class wxCalendarEvent

  The wxCalendarEvent class is used together with wxCalendarCtrl.
- class wxCalendarDateAttr

  wxCalendarDateAttr is a custom attributes for a calendar date.
- class wxCalendarCtrl

  The calendar control allows the user to pick a date.

Enumerations

- enum {
  wxCAL_SUNDAY_FIRST = 0x0000,
  wxCAL_MONDAY_FIRST = 0x0001,
  wxCAL_SHOW_HOLIDAYS = 0x0002,
  wxCAL_NO_YEAR_CHANGE = 0x0004,
  wxCAL_NO_MONTH_CHANGE = 0x000c,
  wxCAL_SEQUENTIAL_MONTH_SELECTION = 0x0010,
  wxCAL_SHOW_SURROUNDING_WEEKS = 0x0020,
  wxCAL_SHOW_WEEK_NUMBERS = 0x0040 }
- enum wxCalendarDateBorder {
  wxCAL_BORDER_NONE,
  wxCAL_BORDER_SQUARE,
  wxCAL_BORDER_ROUND }

  Possible kinds of borders which may be used to decorate a date using wxCalendarDateAttr.
- enum wxCalendarHitTestResult {
  wxCAL_HITTEST_NOWHERE,
  wxCAL_HITTEST_HEADER,
  wxCAL_HITTEST_DAY,
  wxCAL_HITTEST_INCMONTH,
  wxCAL_HITTEST_DECMONTH,
  wxCAL_HITTEST_SURROUNDING_WEEK,
  wxCAL_HITTEST_WEEK }

  Possible return values from wxCalendarCtrl::HitTest().

Variables

- wxEventType wxEVT_CALENDAR_SEL_CHANGED
- wxEventType wxEVT_CALENDAR_PAGE_CHANGED
- wxEventType wxEVT_CALENDAR_DOUBLECLICKED
- wxEventType wxEVT_CALENDAR_WEEKDAY_CLICKED
- wxEventType wxEVT_CALENDAR_WEEK_CLICKED

22.203.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxCAL_SUNDAY_FIRST

wxCAL_MONDAY_FIRST

wxCAL_SHOW_HOLIDAYS

wxCAL_NO_YEAR_CHANGE

wxCAL_NO_MONTH_CHANGE

wxCAL_SEQUENTIAL_MONTH_SELECTION

wxCAL_SHOW_SURROUNDING_WEEKS

wxCAL_SHOW_WEEK_NUMBERS

enum wxCalendarDateBorder

Possible kinds of borders which may be used to decorate a date using wxCalendarDateAttr.

Enumerator

wxCAL_BORDER_NONE No Border (Default)

wxCAL_BORDER_SQUARE Rectangular Border.

wxCAL_BORDER_ROUND Round Border.

enum wxCalendarHitTestResult

Possible return values from wxCalendarCtrl::HitTest().

Enumerator

wxCAL_HITTEST_NOWHERE Hit outside of anything.

wxCAL_HITTEST_HEADER Hit on the header (weekdays).

wxCAL_HITTEST_DAY Hit on a day in the calendar.

wxCAL_HITTEST_INCMONTH Hit on next month arrow (in alternate month selector mode).

wxCAL_HITTEST_DECMONTH Hit on previous month arrow (in alternate month selector mode).

wxCAL_HITTEST_SURROUNDING_WEEK Hit on surrounding week of previous/next month (if shown).

wxCAL_HITTEST_WEEK Hit on week of the year number (if shown).

22.203.2 Variable Documentation

wxEventType wxEVT_CALENDAR_DOUBLECLICKED

wxEventType wxEVT_CALENDAR_PAGE_CHANGED

wxEventType wxEVT_CALENDAR_SEL_CHANGED

wxEventType wxEVT_CALENDAR_WEEK_CLICKED

wxEventType wxEVT_CALENDAR_WEEKDAY_CLICKED

22.204 interface/wx/caret.h File Reference

Classes

- class wxCaret

  A caret is a blinking cursor showing the position where the typed text will appear.

22.205 interface/wx/chartype.h File Reference

Macros

- #define wxT(string)

  This macro can be used with character and string literals (in other words, `'x'` or `"foo"`) to automatically convert them to wide strings in Unicode builds of wxWidgets.
- #define wxT_2(string)

  Compatibility macro which expands to wxT() in wxWidgets 2 only.
- #define wxS(string)

  wxS is a macro which can be used with character and string literals (in other words, `'x'` or `"foo"`) to convert them either to wide characters or wide strings in `wchar_t`-based (UTF-16) builds, or to keep them unchanged in `char`-based (UTF-8) builds.
- #define _T(string)

  This macro is exactly the same as wxT() and is defined in wxWidgets simply because it may be more intuitive for Windows programmers as the standard Win32 headers also define it (as well as yet another name for the same macro which is _TEXT()).

Typedefs

- typedef wxUSE_UNICODE_dependent wxChar

  wxChar is defined to be
  - `char` *when* `wxUSE_UNICODE==0`
  - `wchar_t` *when* `wxUSE_UNICODE==1` *(the default).*
- typedef wxUSE_UNICODE_dependent wxSChar

  wxSChar is defined to be
  - `signed char` *when* `wxUSE_UNICODE==0`
  - `wchar_t` *when* `wxUSE_UNICODE==1` *(the default).*
- typedef wxUSE_UNICODE_dependent wxUChar

  wxUChar is defined to be
  - `unsigned char` *when* `wxUSE_UNICODE==0`
  - `wchar_t` *when* `wxUSE_UNICODE==1` *(the default).*

- typedef
  wxUSE_UNICODE_WCHAR_dependent wxStringCharType

  wxStringCharType is defined to be:
  - `char` *when* `wxUSE_UNICODE==0`
  - `char` *when* `wxUSE_UNICODE_WCHAR==0` *and* `wxUSE_UNICODE==1`
  - `wchar_t` *when* `wxUSE_UNICODE_WCHAR==1` *and* `wxUSE_UNICODE==1`

22.206 interface/wx/checkbox.h File Reference

Classes

- class wxCheckBox

  A checkbox is a labelled box which by default is either on (checkmark is visible) or off (no checkmark).

Macros

- #define wxCHK_2STATE 0x4000
- #define wxCHK_3STATE 0x1000
- #define wxCHK_ALLOW_3RD_STATE_FOR_USER 0x2000

Enumerations

- enum wxCheckBoxState {
  wxCHK_UNCHECKED,
  wxCHK_CHECKED,
  wxCHK_UNDETERMINED }

  The possible states of a 3-state wxCheckBox (Compatible with the 2-state wxCheckBox).

22.206.1 Macro Definition Documentation

#define wxCHK_2STATE 0x4000

#define wxCHK_3STATE 0x1000

#define wxCHK_ALLOW_3RD_STATE_FOR_USER 0x2000

22.206.2 Enumeration Type Documentation

enum wxCheckBoxState

The possible states of a 3-state wxCheckBox (Compatible with the 2-state wxCheckBox).

Enumerator

wxCHK_UNCHECKED

wxCHK_CHECKED

wxCHK_UNDETERMINED 3-state checkbox only

22.207 interface/wx/checklst.h File Reference

Classes

- class wxCheckListBox

  A wxCheckListBox is like a wxListBox, but allows items to be checked or unchecked.

22.208 interface/wx/choicdlg.h File Reference

Classes

- class wxMultiChoiceDialog

  This class represents a dialog that shows a list of strings, and allows the user to select one or more.
- class wxSingleChoiceDialog

  This class represents a dialog that shows a list of strings, and allows the user to select one.

Macros

- #define wxCHOICE_WIDTH 150

  Default width of the choice dialog.
- #define wxCHOICE_HEIGHT 200

  Default height of the choice dialog.
- #define wxCHOICEDLG_STYLE (wxDEFAULT_DIALOG_STYLE | wxOK | wxCANCEL | wxCENTRE | wx↩ RESIZE_BORDER)

  Default style of the choice dialog.

Functions

- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, const wxArrayString &a↩ Choices, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)

  Same as wxGetSingleChoice() but returns the index representing the selected string.
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, int n, const wxString &choices[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, const wxArrayString &choices, int initialSelection, wxWindow *parent=NULL)
- int wxGetSingleChoiceIndex (const wxString &message, const wxString &caption, int n, const wxString *choices, int initialSelection, wxWindow *parent=NULL)
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, const wxArrayString &aChoices, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)

  Pops up a dialog box containing a message, OK/Cancel buttons and a single-selection listbox.
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, int n, const wxString &choices[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, const wxArrayString &choices, int initialSelection, wxWindow *parent=NULL)
- wxString wxGetSingleChoice (const wxString &message, const wxString &caption, int n, const wxString *choices, int initialSelection, wxWindow *parent=NULL)

- wxString wxGetSingleChoiceData (const wxString &message, const wxString &caption, const wxArrayString &aChoices, const wxString &client_data[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)

  Same as wxGetSingleChoice but takes an array of client data pointers corresponding to the strings, and returns one of these pointers or NULL if Cancel was pressed.
- wxString wxGetSingleChoiceData (const wxString &message, const wxString &caption, int n, const wxString &choices[], const wxString &client_data[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT, int initialSelection=0)
- void * wxGetSingleChoiceData (const wxString &message, const wxString &caption, const wxArrayString &choices, void **client_data, int initialSelection, wxWindow *parent=NULL)
- void * wxGetSingleChoiceData (const wxString &message, const wxString &caption, int n, const wxString *choices, void **client_data, int initialSelection, wxWindow *parent=NULL)
- int wxGetSelectedChoices (wxArrayInt &selections, const wxString &message, const wxString &caption, const wxArrayString &aChoices, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT)

  Pops up a dialog box containing a message, OK/Cancel buttons and a multiple-selection listbox.
- int wxGetSelectedChoices (wxArrayInt &selections, const wxString &message, const wxString &caption, int n, const wxString &choices[], wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true, int width=wxCHOICE_WIDTH, int height=wxCHOICE_HEIGHT)

22.208.1 Macro Definition Documentation

#define wxCHOICE_HEIGHT 200

Default height of the choice dialog.

#define wxCHOICE_WIDTH 150

Default width of the choice dialog.

#define wxCHOICEDLG_STYLE (wxDEFAULT_DIALOG_STYLE | wxOK | wxCANCEL | wxCENTRE | wxRESIZE_BORDER)

Default style of the choice dialog.

Remarks

wxRESIZE_BORDER is not used under WinCE.

22.209 interface/wx/choice.h File Reference

Classes

- class wxChoice

  A choice item is used to select one of a list of strings.

22.210 interface/wx/choicebk.h File Reference

Classes

- class wxChoicebook

wxChoicebook is a class similar to wxNotebook, but uses a wxChoice control to show the labels instead of the tabs.

Macros

- #define wxCHB_DEFAULT wxBK_DEFAULT
- #define wxCHB_TOP wxBK_TOP
- #define wxCHB_BOTTOM wxBK_BOTTOM
- #define wxCHB_LEFT wxBK_LEFT
- #define wxCHB_RIGHT wxBK_RIGHT
- #define wxCHB_ALIGN_MASK wxBK_ALIGN_MASK

Variables

- wxEventType wxEVT_CHOICEBOOK_PAGE_CHANGED
- wxEventType wxEVT_CHOICEBOOK_PAGE_CHANGING

22.210.1 Macro Definition Documentation

#define wxCHB_ALIGN_MASK wxBK_ALIGN_MASK

#define wxCHB_BOTTOM wxBK_BOTTOM

#define wxCHB_DEFAULT wxBK_DEFAULT

#define wxCHB_LEFT wxBK_LEFT

#define wxCHB_RIGHT wxBK_RIGHT

#define wxCHB_TOP wxBK_TOP

22.210.2 Variable Documentation

wxEventType wxEVT_CHOICEBOOK_PAGE_CHANGED

wxEventType wxEVT_CHOICEBOOK_PAGE_CHANGING

22.211 interface/wx/clipbrd.h File Reference

Classes

- class wxClipboard

  A class for manipulating the clipboard.

Macros

- #define wxTheClipboard

  The backwards compatible access macro that returns the global clipboard object pointer.

22.211.1 Macro Definition Documentation

#define wxTheClipboard

The backwards compatible access macro that returns the global clipboard object pointer.

22.212 interface/wx/clntdata.h File Reference

Classes

- class wxClientDataContainer

  This class is a mixin that provides storage and management of "client data".
- class wxClientData

  All classes deriving from wxEvtHandler (such as all controls and wxApp) can hold arbitrary data which is here referred to as "client data".
- class wxStringClientData

  Predefined client data class for holding a string.

22.213 interface/wx/clrpicker.h File Reference

Classes

- class wxColourPickerCtrl

  This control allows the user to select a colour.
- class wxColourPickerEvent

  This event class is used for the events generated by wxColourPickerCtrl.

Macros

- #define wxCLRP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)
- #define wxCLRP_DEFAULT_STYLE 0
- #define wxCLRP_SHOW_LABEL 0x0008

Variables

- wxEventType wxEVT_COLOURPICKER_CHANGED

22.213.1 Macro Definition Documentation

#define wxCLRP_DEFAULT_STYLE 0

#define wxCLRP_SHOW_LABEL 0x0008

#define wxCLRP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)

22.213.2 Variable Documentation

wxEventType wxEVT_COLOURPICKER_CHANGED

22.214 interface/wx/cmdline.h File Reference

Classes

- struct wxCmdLineEntryDesc

  The structure wxCmdLineEntryDesc is used to describe a command line switch, option or parameter.
- class wxCmdLineArg

  The interface wxCmdLineArg provides information for an instance of argument passed on command line.
- class wxCmdLineArgs

  An ordered collection of wxCmdLineArg providing an iterator to enumerate the arguments passed on command line.
- class wxCmdLineParser

  wxCmdLineParser is a class for parsing the command line.

Enumerations

- enum wxCmdLineEntryFlags {
  wxCMD_LINE_OPTION_MANDATORY = 0x01,
  wxCMD_LINE_PARAM_OPTIONAL = 0x02,
  wxCMD_LINE_PARAM_MULTIPLE = 0x04,
  wxCMD_LINE_OPTION_HELP = 0x08,
  wxCMD_LINE_NEEDS_SEPARATOR = 0x10,
  wxCMD_LINE_SWITCH_NEGATABLE = 0x20 }

  wxCmdLineEntryDesc::flags field is a combination of these bit masks.
- enum wxCmdLineParamType {
  wxCMD_LINE_VAL_STRING,
  wxCMD_LINE_VAL_NUMBER,
  wxCMD_LINE_VAL_DATE,
  wxCMD_LINE_VAL_DOUBLE,
  wxCMD_LINE_VAL_NONE }

  The possible values of wxCmdLineEntryDesc::type which specify the type of the value accepted by an option.
- enum wxCmdLineEntryType {
  wxCMD_LINE_SWITCH,
  wxCMD_LINE_OPTION,
  wxCMD_LINE_PARAM,
  wxCMD_LINE_USAGE_TEXT,
  wxCMD_LINE_NONE }

  The type of a command line entity used for wxCmdLineEntryDesc::kind.
- enum wxCmdLineSwitchState {
  wxCMD_SWITCH_OFF,
  wxCMD_SWITCH_ON }

  The state of a switch as returned by wxCmdLineParser::FoundSwitch().
- enum wxCmdLineSplitType {
  wxCMD_LINE_SPLIT_DOS,
  wxCMD_LINE_SPLIT_UNIX }

  Flags determining wxCmdLineParser::ConvertStringToArgs() behaviour.

22.214.1 Enumeration Type Documentation

enum wxCmdLineEntryFlags

wxCmdLineEntryDesc::flags field is a combination of these bit masks.

Notice that by default (i.e. if flags are just 0), options are optional (sic) and each call to wxCmdLineParser::AddParam() allows one more parameter - this may be changed by giving non-default flags to it, i.e. use `wxCMD_LINE_OPTION_MANDATORY` to require that the option is given and `wxCMD_LINE_PARAM_OPTIONAL` to make a parameter optional.

Also, `wxCMD_LINE_PARAM_MULTIPLE` may be specified if the programs accepts a variable number of parameters - but it only can be given for the last parameter in the command line description. If you use this flag, you will probably need to use wxCmdLineEntryDesc::GetParamCount() to retrieve the number of parameters effectively specified after calling wxCmdLineEntryDesc::Parse().

`wxCMD_LINE_NEEDS_SEPARATOR` can be specified to require a separator (either a colon, an equal sign or white space) between the option name and its value. By default, no separator is required.

`wxCMD_LINE_SWITCH_NEGATABLE` can be specified if you want to allow the user to specify the switch in both normal form and in negated one (e.g. /R-). You will need to use wxCmdLineParser::FoundSwitch() to distinguish between the normal and negated forms of the switch. This flag is new since wxWidgets 2.9.2.

Enumerator

wxCMD_LINE_OPTION_MANDATORY This option must be given.

wxCMD_LINE_PARAM_OPTIONAL The parameter may be omitted.

wxCMD_LINE_PARAM_MULTIPLE The parameter may be repeated.

wxCMD_LINE_OPTION_HELP This option is a help request.

wxCMD_LINE_NEEDS_SEPARATOR Must have a separator before the value.

wxCMD_LINE_SWITCH_NEGATABLE This switch can be negated (e.g. /S-)

enum wxCmdLineEntryType

The type of a command line entity used for wxCmdLineEntryDesc::kind.

Enumerator

wxCMD_LINE_SWITCH A boolean argument of the program; e.g. `-v` to enable verbose mode.

wxCMD_LINE_OPTION An argument with an associated value; e.g. `"-o filename"` to specify an optional output filename.

wxCMD_LINE_PARAM A parameter: a required program argument.

wxCMD_LINE_USAGE_TEXT Additional usage text. See wxCmdLineParser::AddUsageText.

wxCMD_LINE_NONE Use this to terminate the list.

enum wxCmdLineParamType

The possible values of wxCmdLineEntryDesc::type which specify the type of the value accepted by an option.

Enumerator

wxCMD_LINE_VAL_STRING

wxCMD_LINE_VAL_NUMBER

wxCMD_LINE_VAL_DATE

wxCMD_LINE_VAL_DOUBLE

wxCMD_LINE_VAL_NONE

enum wxCmdLineSplitType

Flags determining wxCmdLineParser::ConvertStringToArgs() behaviour.

Enumerator

wxCMD_LINE_SPLIT_DOS

wxCMD_LINE_SPLIT_UNIX

enum wxCmdLineSwitchState

The state of a switch as returned by wxCmdLineParser::FoundSwitch().

Since

2.9.2

Enumerator

wxCMD_SWITCH_OFF The switch was found in negated form, i.e. followed by a '-'.

wxCMD_SWITCH_ON The switch was not found at all on the command line. The switch was found (and was not negated)

22.215 interface/wx/cmdproc.h File Reference

Classes

- class wxCommand

  wxCommand is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other means provided by the application to change the data or view.

- class wxCommandProcessor

  wxCommandProcessor is a class that maintains a history of wxCommands, with undo/redo functionality built-in.

22.216 interface/wx/cmndata.h File Reference

Classes

- class wxPageSetupDialogData

  This class holds a variety of information related to wxPageSetupDialog.

- class wxPrintData

  This class holds a variety of information related to printers and printer device contexts.

- class wxPrintDialogData

  This class holds information related to the visual characteristics of wxPrintDialog.

Enumerations

- enum wxPrintBin {
  wxPRINTBIN_DEFAULT,
  wxPRINTBIN_ONLYONE,
  wxPRINTBIN_LOWER,
  wxPRINTBIN_MIDDLE,
  wxPRINTBIN_MANUAL,
  wxPRINTBIN_ENVELOPE,
  wxPRINTBIN_ENVMANUAL,
  wxPRINTBIN_AUTO,
  wxPRINTBIN_TRACTOR,
  wxPRINTBIN_SMALLFMT,
  wxPRINTBIN_LARGEFMT,
  wxPRINTBIN_LARGECAPACITY,
  wxPRINTBIN_CASSETTE,
  wxPRINTBIN_FORMSOURCE,
  wxPRINTBIN_USER }

  Enumeration of various printer bin sources.

22.216.1 Enumeration Type Documentation

enum wxPrintBin

Enumeration of various printer bin sources.

See also

wxPrintData::SetBin()

Enumerator

wxPRINTBIN_DEFAULT

wxPRINTBIN_ONLYONE

wxPRINTBIN_LOWER

wxPRINTBIN_MIDDLE

wxPRINTBIN_MANUAL

wxPRINTBIN_ENVELOPE

wxPRINTBIN_ENVMANUAL

wxPRINTBIN_AUTO

wxPRINTBIN_TRACTOR

wxPRINTBIN_SMALLFMT

wxPRINTBIN_LARGEFMT

wxPRINTBIN_LARGECAPACITY

wxPRINTBIN_CASSETTE

wxPRINTBIN_FORMSOURCE

wxPRINTBIN_USER

22.217 interface/wx/collpane.h File Reference

Classes

- class wxCollapsiblePaneEvent

This event class is used for the events generated by wxCollapsiblePane.

- class wxCollapsiblePane

  A collapsible pane is a container with an embedded button-like control which can be used by the user to collapse or expand the pane's contents.

Macros

- #define wxCP_DEFAULT_STYLE (wxTAB_TRAVERSAL | wxNO_BORDER)
- #define wxCP_NO_TLW_RESIZE (0x0002)

Variables

- wxEventType wxEVT_COLLAPSIBLEPANE_CHANGED

22.217.1 Macro Definition Documentation

#define wxCP_DEFAULT_STYLE (wxTAB_TRAVERSAL | wxNO_BORDER)

#define wxCP_NO_TLW_RESIZE (0x0002)

22.217.2 Variable Documentation

wxEventType wxEVT_COLLAPSIBLEPANE_CHANGED

22.218 interface/wx/colordlg.h File Reference

Classes

- class wxColourDialog

  This class represents the colour chooser dialog.

Functions

- wxColour wxGetColourFromUser (wxWindow *parent, const wxColour &colInit, const wxString &caption=wxEmptyString, wxColourData *data=NULL)

  Shows the colour selection dialog and returns the colour selected by user or invalid colour (use wxColour::IsOk() to test whether a colour is valid) if the dialog was cancelled.

22.219 interface/wx/colour.h File Reference

Classes

- class wxColour

  A colour is an object representing a combination of Red, Green, and Blue (RGB) intensity values, and is used to determine drawing colours.

Enumerations

- enum {
  wxC2S_NAME = 1,
  wxC2S_CSS_SYNTAX = 2,
  wxC2S_HTML_SYNTAX = 4 }

  Flags for wxColour -> wxString conversion (see wxColour::GetAsString).

Functions

- bool wxFromString (const wxString &string, wxColour *colour)

  Converts string to a wxColour best represented by the given string.
- wxString wxToString (const wxColour &colour)

  Converts the given wxColour into a string.

Variables

- const unsigned char wxALPHA_TRANSPARENT = 0
- const unsigned char wxALPHA_OPAQUE = 0xff

Predefined colors.

- wxColour wxNullColour
- wxColour wxTransparentColour
- wxColour * wxBLACK
- wxColour * wxBLUE
- wxColour * wxCYAN
- wxColour * wxGREEN
- wxColour * wxYELLOW
- wxColour * wxLIGHT_GREY
- wxColour * wxRED
- wxColour * wxWHITE

22.219.1 Enumeration Type Documentation

anonymous enum

Flags for wxColour -> wxString conversion (see wxColour::GetAsString).

Enumerator

wxC2S_NAME

wxC2S_CSS_SYNTAX

wxC2S_HTML_SYNTAX

22.219.2 Variable Documentation

const unsigned char wxALPHA_OPAQUE = 0xff

const unsigned char wxALPHA_TRANSPARENT = 0

wxColour* wxBLACK

wxColour* wxBLUE

wxColour* wxCYAN

wxColour* wxGREEN

wxColour* wxLIGHT_GREY

wxColour wxNullColour

wxColour* wxRED

wxColour wxTransparentColour

wxColour* wxWHITE

wxColour* wxYELLOW

22.220 interface/wx/colourdata.h File Reference

Classes

- class wxColourData

  This class holds a variety of information related to colour dialogs.

22.221 interface/wx/combo.h File Reference

Classes

- class wxComboPopup

  In order to use a custom popup with wxComboCtrl, an interface class must be derived from wxComboPopup.
- struct wxComboCtrlFeatures

  Features enabled for wxComboCtrl.
- class wxComboCtrl

  A combo control is a generic combobox that allows totally custom popup.

Enumerations

- enum {
  wxCC_SPECIAL_DCLICK = 0x0100,
  wxCC_STD_BUTTON = 0x0200 }

22.221.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxCC_SPECIAL_DCLICK

wxCC_STD_BUTTON

22.222 interface/wx/combobox.h File Reference

Classes

- class wxComboBox

  A combobox is like a combination of an edit control and a listbox.

22.223 interface/wx/commandlinkbutton.h File Reference

Classes

- class wxCommandLinkButton

  Objects of this class are similar in appearance to the normal wxButtons but are similar to the links in a web page in functionality.

22.224 interface/wx/containr.h File Reference

Classes

- class wxNavigationEnabled< W >

  A helper class implementing TAB navigation among the window children.

22.225 interface/wx/control.h File Reference

Classes

- class wxControl

  This is the base class for a control or "widget".

Enumerations

- enum wxEllipsizeFlags {
  wxELLIPSIZE_FLAGS_NONE = 0,
  wxELLIPSIZE_FLAGS_PROCESS_MNEMONICS = 1,
  wxELLIPSIZE_FLAGS_EXPAND_TABS = 2,
  wxELLIPSIZE_FLAGS_DEFAULT }

  Flags used by wxControl::Ellipsize function.
- enum wxEllipsizeMode {
  wxELLIPSIZE_NONE,
  wxELLIPSIZE_START,
  wxELLIPSIZE_MIDDLE,
  wxELLIPSIZE_END }

  The different ellipsization modes supported by the wxControl::Ellipsize function.

22.225.1 Enumeration Type Documentation

enum wxEllipsizeFlags

Flags used by wxControl::Ellipsize function.

Enumerator

wxELLIPSIZE_FLAGS_NONE No special flags.

wxELLIPSIZE_FLAGS_PROCESS_MNEMONICS Take mnemonics into account when calculating the text width. With this flag when calculating the size of the passed string, mnemonics characters (see wxControl::SetLabel) will be automatically reduced to a single character. This leads to correct calculations only if the string passed to Ellipsize() will be used with wxControl::SetLabel. If you don't want ampersand to be interpreted as mnemonics (e.g. because you use wxControl::SetLabelText) then don't use this flag.

wxELLIPSIZE_FLAGS_EXPAND_TABS Expand tabs in spaces when calculating the text width. This flag tells wxControl::Ellipsize() to calculate the width of tab characters `'\t'` as 6 spaces.

wxELLIPSIZE_FLAGS_DEFAULT The default flags for wxControl::Ellipsize.

enum wxEllipsizeMode

The different ellipsization modes supported by the wxControl::Ellipsize function.

Enumerator

wxELLIPSIZE_NONE Don't ellipsize the text at all.

Since

2.9.1

wxELLIPSIZE_START Put the ellipsis at the start of the string, if the string needs ellipsization.

wxELLIPSIZE_MIDDLE Put the ellipsis in the middle of the string, if the string needs ellipsization.

wxELLIPSIZE_END Put the ellipsis at the end of the string, if the string needs ellipsization.

22.226 interface/wx/ribbon/control.h File Reference

Classes

- class wxRibbonControl

  wxRibbonControl serves as a base class for all controls which share the ribbon characteristics of having a ribbon art provider, and (optionally) non-continuous resizing.

22.227 interface/wx/convauto.h File Reference

Classes

- class wxConvAuto

  This class implements a Unicode to/from multibyte converter capable of automatically recognizing the encoding of the multibyte text on input.

Enumerations

- enum wxBOM {
  wxBOM_Unknown = -1,
  wxBOM_None,
  wxBOM_UTF32BE,
  wxBOM_UTF32LE,
  wxBOM_UTF16BE,
  wxBOM_UTF16LE,
  wxBOM_UTF8 }

  Constants representing various BOM types.

22.227.1 Enumeration Type Documentation

enum wxBOM

Constants representing various BOM types.

BOM is an abbreviation for "Byte Order Mark", a special Unicode character which may be inserted into the beginning of a text stream to indicate its encoding.

Since

2.9.3

Enumerator

wxBOM_Unknown Unknown BOM. This is returned if BOM presence couldn't be determined and normally happens because not enough bytes of input have been analysed.

wxBOM_None No BOM. The stream doesn't contain BOM character at all.

wxBOM_UTF32BE UTF-32 big endian BOM. The stream is encoded in big endian variant of UTF-32.

wxBOM_UTF32LE UTF-32 little endian BOM. The stream is encoded in little endian variant of UTF-32.

wxBOM_UTF16BE UTF-16 big endian BOM. The stream is encoded in big endian variant of UTF-16.

wxBOM_UTF16LE UTF-16 little endian BOM. The stream is encoded in little endian variant of UTF-16.

wxBOM_UTF8 UTF-8 BOM. The stream is encoded in UTF-8.

Notice that contrary to a popular belief, it's perfectly possible and, n fact, common under Microsoft Windows systems, to have a BOM in an UTF-8 stream: while it's not used to indicate the endianness of UTF-8 stream (as it's byte-oriented), the BOM can still be useful just as an unambiguous indicator of UTF-8 being used.

22.228 interface/wx/cpp.h File Reference

Macros

- #define wxCONCAT(x1, x2)

  This macro returns the concatenation of the arguments passed.
- #define wxCONCAT3(x1, x2, x3)
- #define wxCONCAT4(x1, x2, x3, x4)
- #define wxCONCAT5(x1, x2, x3, x4, x5)
- #define wxSTRINGIZE(x)

  Returns the string representation of the given symbol which can be either a literal or a macro (hence the advantage of using this macro instead of the standard preprocessor # operator which doesn't work with macros).
- #define wxSTRINGIZE_T(x)

  Returns the string representation of the given symbol as either an ASCII or Unicode string, depending on the current build.
- #define __WXFUNCTION__

  This macro expands to the name of the current function if the compiler supports any of ***`__FUNCTION__`****,* ***`__func__`*** *or equivalent variables or macros or to NULL if none of them is available.*

22.229 interface/wx/cshelp.h File Reference

Classes

- class wxHelpProvider

wxHelpProvider is an abstract class used by a program implementing context-sensitive help to show the help text for the given window.

- class wxHelpControllerHelpProvider

  wxHelpControllerHelpProvider is an implementation of wxHelpProvider which supports both context identifiers and plain text help strings.

- class wxContextHelp

  This class changes the cursor to a query and puts the application into a 'context-sensitive help mode'.

- class wxContextHelpButton

  Instances of this class may be used to add a question mark button that when pressed, puts the application into context-help mode.

- class wxSimpleHelpProvider

  wxSimpleHelpProvider is an implementation of wxHelpProvider which supports only plain text help strings, and shows the string associated with the control (if any) in a tooltip.

22.230 interface/wx/ctrlsub.h File Reference

Classes

- class wxItemContainerImmutable

  wxItemContainer defines an interface which is implemented by all controls which have string subitems each of which may be selected.

- class wxItemContainer

  This class is an abstract base class for some wxWidgets controls which contain several items such as wxListBox, wxCheckListBox, wxComboBox or wxChoice.

- class wxControlWithItems

  This is convenience class that derives from both wxControl and wxItemContainer.

22.231 interface/wx/cursor.h File Reference

Classes

- class wxCursor

  A cursor is a small bitmap usually used for denoting where the mouse pointer is, with a picture that might indicate the interpretation of a mouse click.

Variables

Predefined cursors.

See also

wxStockCursor

- wxCursor wxNullCursor
- wxCursor * wxSTANDARD_CURSOR
- wxCursor * wxHOURGLASS_CURSOR
- wxCursor * wxCROSS_CURSOR

22.231.1 Variable Documentation

wxCursor* wxCROSS_CURSOR

wxCursor* wxHOURGLASS_CURSOR

wxCursor wxNullCursor

wxCursor* wxSTANDARD_CURSOR

22.232 interface/wx/custombgwin.h File Reference

Classes

- class wxCustomBackgroundWindow< W >

  A helper class making it possible to use custom background for any window.

22.233 interface/wx/dataobj.h File Reference

Classes

- class wxDataFormat

  A wxDataFormat is an encapsulation of a platform-specific format handle which is used by the system for the clipboard and drag and drop operations.
- class wxDataObject

  A wxDataObject represents data that can be copied to or from the clipboard, or dragged and dropped.
- class wxCustomDataObject

  wxCustomDataObject is a specialization of wxDataObjectSimple for some application-specific data in arbitrary (either custom or one of the standard ones).
- class wxDataObjectComposite

  wxDataObjectComposite is the simplest wxDataObject derivation which may be used to support multiple formats.
- class wxDataObjectSimple

  This is the simplest possible implementation of the wxDataObject class.
- class wxBitmapDataObject

  wxBitmapDataObject is a specialization of wxDataObject for bitmap data.
- class wxURLDataObject

  wxURLDataObject is a wxDataObject containing an URL and can be used e.g.
- class wxTextDataObject

  wxTextDataObject is a specialization of wxDataObjectSimple for text data.
- class wxFileDataObject

  wxFileDataObject is a specialization of wxDataObject for file names.
- class wxHTMLDataObject

  wxHTMLDataObject is used for working with HTML-formatted text.

Variables

- const wxDataFormat wxFormatInvalid

22.233.1 Variable Documentation

const wxDataFormat wxFormatInvalid

22.234 interface/wx/dataview.h File Reference

Classes

- class wxDataViewModel

  wxDataViewModel is the base class for all data model to be displayed by a wxDataViewCtrl.
- class wxDataViewListModel

  Base class with abstract API for wxDataViewIndexListModel and wxDataViewVirtualListModel.
- class wxDataViewIndexListModel

  wxDataViewIndexListModel is a specialized data model which lets you address an item by its position (row) rather than its wxDataViewItem (which you can obtain from this class).
- class wxDataViewVirtualListModel

  wxDataViewVirtualListModel is a specialized data model which lets you address an item by its position (row) rather than its wxDataViewItem and as such offers the exact same interface as wxDataViewIndexListModel.
- class wxDataViewItemAttr

  This class is used to indicate to a wxDataViewCtrl that a certain item (see wxDataViewItem) has extra font attributes for its renderer.
- class wxDataViewItem

  wxDataViewItem is a small opaque class that represents an item in a wxDataViewCtrl in a persistent way, i.e.
- class wxDataViewCtrl

  wxDataViewCtrl is a control to display data either in a tree like fashion or in a tabular form or both.
- class wxDataViewModelNotifier

  A wxDataViewModelNotifier instance is owned by a wxDataViewModel and mirrors its notification interface.
- class wxDataViewRenderer

  This class is used by wxDataViewCtrl to render the individual cells.
- class wxDataViewTextRenderer

  wxDataViewTextRenderer is used for rendering text.
- class wxDataViewIconTextRenderer

  The wxDataViewIconTextRenderer class is used to display text with a small icon next to it as it is typically done in a file manager.
- class wxDataViewProgressRenderer

  This class is used by wxDataViewCtrl to render progress bars.
- class wxDataViewSpinRenderer

  This is a specialized renderer for rendering integer values.
- class wxDataViewToggleRenderer

  This class is used by wxDataViewCtrl to render toggle controls.
- class wxDataViewChoiceRenderer

  A wxDataViewCtrl renderer using wxChoice control and values of strings in it.
- class wxDataViewChoiceByIndexRenderer

  A wxDataViewCtrl renderer using wxChoice control and indexes into it.
- class wxDataViewDateRenderer

  This class is used by wxDataViewCtrl to render calendar controls.
- class wxDataViewCustomRenderer

  You need to derive a new class from wxDataViewCustomRenderer in order to write a new renderer.
- class wxDataViewBitmapRenderer

  This class is used by wxDataViewCtrl to render bitmap controls.
- class wxDataViewColumn

This class represents a column in a wxDataViewCtrl.

- class wxDataViewListCtrl

  This class is a wxDataViewCtrl which internally uses a wxDataViewListStore and forwards most of its API to that class.

- class wxDataViewTreeCtrl

  This class is a wxDataViewCtrl which internally uses a wxDataViewTreeStore and forwards most of its API to that class.

- class wxDataViewListStore

  wxDataViewListStore is a specialised wxDataViewModel for storing a simple table of data.

- class wxDataViewTreeStore

  wxDataViewTreeStore is a specialised wxDataViewModel for storing simple trees very much like wxTreeCtrl does and it offers a similar API.

- class wxDataViewIconText

  wxDataViewIconText is used by wxDataViewIconTextRenderer for data transfer.

- class wxDataViewEvent

  This is the event class for the wxDataViewCtrl notifications.

Macros

- #define wxDVC_DEFAULT_RENDERER_SIZE 20
- #define wxDVC_DEFAULT_WIDTH 80
- #define wxDVC_TOGGLE_DEFAULT_WIDTH 30
- #define wxDVC_DEFAULT_MINWIDTH 30
- #define wxDVR_DEFAULT_ALIGNMENT -1
- #define wxDV_SINGLE 0x0000
- #define wxDV_MULTIPLE 0x0001
- #define wxDV_NO_HEADER 0x0002
- #define wxDV_HORIZ_RULES 0x0004
- #define wxDV_VERT_RULES 0x0008
- #define wxDV_ROW_LINES 0x0010
- #define wxDV_VARIABLE_LINE_HEIGHT 0x0020

Enumerations

- enum wxDataViewCellMode {
  wxDATAVIEW_CELL_INERT,
  wxDATAVIEW_CELL_ACTIVATABLE,
  wxDATAVIEW_CELL_EDITABLE }

  The mode of a data-view cell; see wxDataViewRenderer for more info.

- enum wxDataViewCellRenderState {
  wxDATAVIEW_CELL_SELECTED = 1,
  wxDATAVIEW_CELL_PRELIT = 2,
  wxDATAVIEW_CELL_INSENSITIVE = 4,
  wxDATAVIEW_CELL_FOCUSED = 8 }

  The values of this enum controls how a wxDataViewRenderer should display its contents in a cell.

- enum wxDataViewColumnFlags {
  wxDATAVIEW_COL_RESIZABLE = 1,
  wxDATAVIEW_COL_SORTABLE = 2,
  wxDATAVIEW_COL_REORDERABLE = 4,
  wxDATAVIEW_COL_HIDDEN = 8 }

  The flags used by wxDataViewColumn.

Variables

- wxEventType wxEVT_DATAVIEW_SELECTION_CHANGED
- wxEventType wxEVT_DATAVIEW_ITEM_ACTIVATED
- wxEventType wxEVT_DATAVIEW_ITEM_COLLAPSING
- wxEventType wxEVT_DATAVIEW_ITEM_COLLAPSED
- wxEventType wxEVT_DATAVIEW_ITEM_EXPANDING
- wxEventType wxEVT_DATAVIEW_ITEM_EXPANDED
- wxEventType wxEVT_DATAVIEW_ITEM_START_EDITING
- wxEventType wxEVT_DATAVIEW_ITEM_EDITING_STARTED
- wxEventType wxEVT_DATAVIEW_ITEM_EDITING_DONE
- wxEventType wxEVT_DATAVIEW_ITEM_VALUE_CHANGED
- wxEventType wxEVT_DATAVIEW_ITEM_CONTEXT_MENU
- wxEventType wxEVT_DATAVIEW_COLUMN_HEADER_CLICK
- wxEventType wxEVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK
- wxEventType wxEVT_DATAVIEW_COLUMN_SORTED
- wxEventType wxEVT_DATAVIEW_COLUMN_REORDERED
- wxEventType wxEVT_DATAVIEW_CACHE_HINT
- wxEventType wxEVT_DATAVIEW_ITEM_BEGIN_DRAG
- wxEventType wxEVT_DATAVIEW_ITEM_DROP_POSSIBLE
- wxEventType wxEVT_DATAVIEW_ITEM_DROP

22.234.1 Macro Definition Documentation

#define wxDV_HORIZ_RULES 0x0004

#define wxDV_MULTIPLE 0x0001

#define wxDV_NO_HEADER 0x0002

#define wxDV_ROW_LINES 0x0010

#define wxDV_SINGLE 0x0000

#define wxDV_VARIABLE_LINE_HEIGHT 0x0020

#define wxDV_VERT_RULES 0x0008

#define wxDVC_DEFAULT_MINWIDTH 30

#define wxDVC_DEFAULT_RENDERER_SIZE 20

#define wxDVC_DEFAULT_WIDTH 80

#define wxDVC_TOGGLE_DEFAULT_WIDTH 30

#define wxDVR_DEFAULT_ALIGNMENT -1

22.234.2 Enumeration Type Documentation

enum wxDataViewCellMode

The mode of a data-view cell; see wxDataViewRenderer for more info.

Enumerator

wxDATAVIEW_CELL_INERT The cell only displays information and cannot be manipulated or otherwise interacted with in any way. Note that this doesn't mean that the row being drawn can't be selected, just that a particular element of it cannot be individually modified.

wxDATAVIEW_CELL_ACTIVATABLE Indicates that the cell can be *activated* by clicking it or using keyboard. Activating a cell is an alternative to showing inline editor when the value can be edited in a simple way that doesn't warrant full editor control. The most typical use of cell activation is toggling the checkbox in wxDataViewToggleRenderer; others would be e.g. an embedded volume slider or a five-star rating column.

The exact means of activating a cell are platform-dependent, but they are usually similar to those used for inline editing of values. Typically, a cell would be activated by Space or Enter keys or by left mouse click.

Note

Do not confuse this with item activation in wxDataViewCtrl and the wxEVT_DATAVIEW_ITEM_↩ ACTIVATED event. That one is used for activating the item (or, to put it differently, the entire row) similarly to analogous messages in wxTreeCtrl and wxListCtrl, and the effect differs (play a song, open a file etc.). Cell activation, on the other hand, is all about interacting with the individual cell.

See also

wxDataViewCustomRenderer::ActivateCell()

wxDATAVIEW_CELL_EDITABLE Indicates that the user can edit the data in-place in an inline editor control that will show up when the user wants to edit the cell. A typical example of this behaviour is changing the filename in a file managers.

Editing is typically triggered by slowly double-clicking the cell or by a platform-dependent keyboard shortcut (F2 is typical on Windows, Space and/or Enter is common elsewhere and supported on Windows too).

See also

wxDataViewCustomRenderer::CreateEditorCtrl()

enum wxDataViewCellRenderState

The values of this enum controls how a wxDataViewRenderer should display its contents in a cell.

Enumerator

wxDATAVIEW_CELL_SELECTED

wxDATAVIEW_CELL_PRELIT

wxDATAVIEW_CELL_INSENSITIVE

wxDATAVIEW_CELL_FOCUSED

enum wxDataViewColumnFlags

The flags used by wxDataViewColumn.

Can be combined together.

Enumerator

wxDATAVIEW_COL_RESIZABLE

wxDATAVIEW_COL_SORTABLE

wxDATAVIEW_COL_REORDERABLE

wxDATAVIEW_COL_HIDDEN

22.234.3 Variable Documentation

wxEventType wxEVT_DATAVIEW_CACHE_HINT

wxEventType wxEVT_DATAVIEW_COLUMN_HEADER_CLICK

wxEventType wxEVT_DATAVIEW_COLUMN_HEADER_RIGHT_CLICK

wxEventType wxEVT_DATAVIEW_COLUMN_REORDERED

wxEventType wxEVT_DATAVIEW_COLUMN_SORTED

wxEventType wxEVT_DATAVIEW_ITEM_ACTIVATED

wxEventType wxEVT_DATAVIEW_ITEM_BEGIN_DRAG

wxEventType wxEVT_DATAVIEW_ITEM_COLLAPSED

wxEventType wxEVT_DATAVIEW_ITEM_COLLAPSING

wxEventType wxEVT_DATAVIEW_ITEM_CONTEXT_MENU

wxEventType wxEVT_DATAVIEW_ITEM_DROP

wxEventType wxEVT_DATAVIEW_ITEM_DROP_POSSIBLE

wxEventType wxEVT_DATAVIEW_ITEM_EDITING_DONE

wxEventType wxEVT_DATAVIEW_ITEM_EDITING_STARTED

wxEventType wxEVT_DATAVIEW_ITEM_EXPANDED

wxEventType wxEVT_DATAVIEW_ITEM_EXPANDING

wxEventType wxEVT_DATAVIEW_ITEM_START_EDITING

wxEventType wxEVT_DATAVIEW_ITEM_VALUE_CHANGED

wxEventType wxEVT_DATAVIEW_SELECTION_CHANGED

22.235 interface/wx/datectrl.h File Reference

Classes

- class wxDatePickerCtrl

  This control allows the user to select a date.

Enumerations

- enum {
  wxDP_DEFAULT = 0,
  wxDP_SPIN = 1,
  wxDP_DROPDOWN = 2,
  wxDP_SHOWCENTURY = 4,
  wxDP_ALLOWNONE = 8 }

wxDatePickerCtrl styles

22.235.1 Enumeration Type Documentation

anonymous enum

wxDatePickerCtrl styles

Enumerator

wxDP_DEFAULT default style on this platform, either wxDP_SPIN or wxDP_DROPDOWN

wxDP_SPIN a spin control-like date picker (not supported in generic version)

wxDP_DROPDOWN a combobox-like date picker (not supported in mac version)

wxDP_SHOWCENTURY always show century in the default date display (otherwise it depends on the system date format which may include the century or not)

wxDP_ALLOWNONE allow not having any valid date in the control (by default it always has some date, today initially if no valid date specified in ctor)

22.236 interface/wx/dateevt.h File Reference

Classes

- class wxDateEvent

  This event class holds information about a date change and is used together with wxDatePickerCtrl.

Variables

- wxEventType wxEVT_DATE_CHANGED
- wxEventType wxEVT_TIME_CHANGED

22.236.1 Variable Documentation

wxEventType wxEVT_DATE_CHANGED

wxEventType wxEVT_TIME_CHANGED

22.237 interface/wx/datstrm.h File Reference

Classes

- class wxDataOutputStream

  This class provides functions that write binary data types in a portable way.
- class wxDataInputStream

  This class provides functions that read binary data types in a portable way.

22.238 interface/wx/dcbuffer.h File Reference

Classes

- class wxBufferedDC

This class provides a simple way to avoid flicker: when drawing on it, everything is in fact first drawn on an in-memory buffer (a wxBitmap) and then copied to the screen, using the associated wxDC, only once, when this object is destroyed.

- class wxAutoBufferedPaintDC

  This wxDC derivative can be used inside of an `EVT_PAINT()` event handler to achieve double-buffered drawing.

- class wxBufferedPaintDC

  This is a subclass of wxBufferedDC which can be used inside of an `EVT_PAINT()` event handler to achieve double-buffered drawing.

Macros

- #define wxBUFFER_VIRTUAL_AREA 0x01
- #define wxBUFFER_CLIENT_AREA 0x02
- #define wxBUFFER_USES_SHARED_BUFFER 0x04

Functions

- wxDC * wxAutoBufferedPaintDCFactory (wxWindow *window)

  Check if the window is natively double buffered and will return a wxPaintDC if it is, a wxBufferedPaintDC otherwise.

22.238.1 Macro Definition Documentation

#define wxBUFFER_CLIENT_AREA 0x02

#define wxBUFFER_USES_SHARED_BUFFER 0x04

#define wxBUFFER_VIRTUAL_AREA 0x01

22.238.2 Function Documentation

wxDC* wxAutoBufferedPaintDCFactory (wxWindow * *window*)

Check if the window is natively double buffered and will return a wxPaintDC if it is, a wxBufferedPaintDC otherwise.

It is the caller's responsibility to delete the wxDC pointer when finished with it.

22.239 interface/wx/dcclient.h File Reference

Classes

- class wxPaintDC

  A wxPaintDC must be constructed if an application wishes to paint on the client area of a window from within an EVT_PAINT() event handler.

- class wxClientDC

  A wxClientDC must be constructed if an application wishes to paint on the client area of a window from outside an EVT_PAINT() handler.

- class wxWindowDC

  A wxWindowDC must be constructed if an application wishes to paint on the whole area of a window (client and decorations).

22.240 interface/wx/dcgraph.h File Reference

Classes

- class wxGCDC

  wxGCDC is a device context that draws on a wxGraphicsContext.

22.241 interface/wx/dcmemory.h File Reference

Classes

- class wxMemoryDC

  A memory device context provides a means to draw graphics onto a bitmap.

22.242 interface/wx/dcmirror.h File Reference

Classes

- class wxMirrorDC

  wxMirrorDC is a simple wrapper class which is always associated with a real wxDC object and either forwards all of its operations to it without changes (no mirroring takes place) or exchanges x and y coordinates which makes it possible to reuse the same code to draw a figure and its mirror – i.e.

22.243 interface/wx/dcprint.h File Reference

Classes

- class wxPrinterDC

  A printer device context is specific to MSW and Mac, and allows access to any printer with a Windows or Macintosh driver.

22.244 interface/wx/dcps.h File Reference

Classes

- class wxPostScriptDC

  This defines the wxWidgets Encapsulated PostScript device context, which can write PostScript files on any platform.

22.245 interface/wx/dcscreen.h File Reference

Classes

- class wxScreenDC

  A wxScreenDC can be used to paint on the screen.

22.246 interface/wx/dcsvg.h File Reference

Classes

- class wxSVGFileDC

  A wxSVGFileDC is a device context onto which graphics and text can be drawn, and the output produced as a vector file, in SVG format.
- class wxSVGBitmapHandler

  Abstract base class for handling bitmaps inside a wxSVGFileDC.
- class wxSVGBitmapEmbedHandler

  Handler embedding bitmaps as base64-encoded PNGs into the SVG.
- class wxSVGBitmapFileHandler

  Handler saving a bitmap to an external file and linking to it from the SVG.

22.247 interface/wx/dde.h File Reference

Classes

- class wxDDEConnection

  A wxDDEConnection object represents the connection between a client and a server.
- class wxDDEClient

  A wxDDEClient object represents the client part of a client-server DDE (Dynamic Data Exchange) conversation.
- class wxDDEServer

  A wxDDEServer object represents the server part of a client-server DDE (Dynamic Data Exchange) conversation.

Functions

- void wxDDECleanUp ()

  Called when wxWidgets exits, to clean up the DDE system.
- void wxDDEInitialize ()

  Initializes the DDE system.

22.248 interface/wx/debug.h File Reference

Macros

- #define wxDEBUG_LEVEL

  Preprocessor symbol defining the level of debug support available.
- #define __WXDEBUG__

  Compatibility macro indicating presence of debug support.
- #define wxASSERT(condition)

  Assert macro.
- #define wxASSERT_LEVEL_2(condition)

  Assert macro for expensive run-time checks.
- #define wxASSERT_LEVEL_2_MSG(condition, msg)

  Assert macro with a custom message for expensive run-time checks.
- #define wxASSERT_MIN_BITSIZE(type, size)

  This macro results in a compile time assertion failure if the size of the given `type` is less than `size` bits.
- #define wxASSERT_MSG(condition, message)

Assert macro with message.

- #define wxASSERT_MSG_AT(condition, message, file, line, func)

  Assert macro pretending to assert at the specified location.

- #define wxCHECK(condition, retValue)

  Checks that the condition is true, returns with the given return value if not (stops execution in debug mode).

- #define wxCHECK_MSG(condition, retValue, message)

  Checks that the condition is true, returns with the given return value if not (stops execution in debug mode).

- #define wxCHECK_RET(condition, message)

  Checks that the condition is true, and returns if not (stops execution with the given error message in debug mode).

- #define wxCHECK2(condition, operation)

  Checks that the condition is true, and if not, it will wxFAIL() and execute the given `operation` if it is not.

- #define wxCHECK2_MSG(condition, operation, message)

  This is the same as wxCHECK2(), but wxFAIL_MSG() with the specified `message` is called instead of wxFAIL() if the `condition` is false.

- #define wxCOMPILE_TIME_ASSERT(condition, message)

  Using wxCOMPILE_TIME_ASSERT() results in a compilation error if the specified `condition` is false.

- #define wxCOMPILE_TIME_ASSERT2(condition, message, name)

  This macro is identical to wxCOMPILE_TIME_ASSERT() except that it allows you to specify a unique `name` for the struct internally defined by this macro to avoid getting the compilation errors described for wxCOMPILE_TIME_AS↩SERT().

- #define wxDISABLE_ASSERTS_IN_RELEASE_BUILD() wxDisableAsserts()

  Use this macro to disable asserts in release build when not using wxIMPLEMENT_APP().

- #define wxFAIL

  Will always generate an assert error if this code is reached (in debug mode).

- #define wxFAIL_MSG(message)

  Will always generate an assert error with specified message if this code is reached (in debug mode).

- #define wxFAIL_MSG_AT(message, file, line, func)

  Assert failure macro pretending to assert at the specified location.

Typedefs

- typedef void(∗ wxAssertHandler_t)(const wxString &file, int line, const wxString &func, const wxString &cond, const wxString &msg)

  Type for the function called in case of assert failure.

Functions

- void wxAbort ()

  Exits the program immediately.

- void wxDisableAsserts ()

  Disable the condition checks in the assertions.

- bool wxIsDebuggerRunning ()

  Returns true if the program is running under debugger, false otherwise.

- wxAssertHandler_t wxSetAssertHandler (wxAssertHandler_t handler)

  Sets the function to be called in case of assertion failure.

- void wxSetDefaultAssertHandler ()

  Reset the assert handler to default function which shows a message box when an assert happens.

- void wxTrap ()

  Generate a debugger exception meaning that the control is passed to the debugger if one is attached to the process.

22.249 interface/wx/debugrpt.h File Reference

Classes

- class wxDebugReportPreview

  This class presents the debug report to the user and allows him to veto report entirely or remove some parts of it.
- class wxDebugReportCompress

  wxDebugReportCompress is a wxDebugReport which compresses all the files in this debug report into a single ZIP file in its wxDebugReport::Process() function.
- class wxDebugReport

  wxDebugReport is used to generate a debug report, containing information about the program current state.
- class wxDebugReportPreviewStd

  wxDebugReportPreviewStd is a standard debug report preview window.
- class wxDebugReportUpload

  This class is used to upload a compressed file using HTTP POST request.

22.250 interface/wx/defs.h File Reference

Macros

- #define wxSIZE_AUTO_WIDTH 0x0001
- #define wxSIZE_AUTO_HEIGHT 0x0002
- #define wxSIZE_AUTO (wxSIZE_AUTO_WIDTH|wxSIZE_AUTO_HEIGHT)
- #define wxSIZE_USE_EXISTING 0x0000
- #define wxSIZE_ALLOW_MINUS_ONE 0x0004
- #define wxSIZE_NO_ADJUSTMENTS 0x0008
- #define wxSIZE_FORCE 0x0010
- #define wxSIZE_FORCE_EVENT 0x0020
- #define wxVSCROLL 0x80000000
- #define wxHSCROLL 0x40000000
- #define wxCAPTION 0x20000000
- #define wxDOUBLE_BORDER wxBORDER_DOUBLE
- #define wxSUNKEN_BORDER wxBORDER_SUNKEN
- #define wxRAISED_BORDER wxBORDER_RAISED
- #define wxBORDER wxBORDER_SIMPLE
- #define wxSIMPLE_BORDER wxBORDER_SIMPLE
- #define wxSTATIC_BORDER wxBORDER_STATIC
- #define wxNO_BORDER wxBORDER_NONE
- #define wxALWAYS_SHOW_SB 0x00800000
- #define wxCLIP_CHILDREN 0x00400000
- #define wxCLIP_SIBLINGS 0x20000000
- #define wxTRANSPARENT_WINDOW 0x00100000
- #define wxTAB_TRAVERSAL 0x00080000
- #define wxWANTS_CHARS 0x00040000
- #define wxRETAINED 0x00000000
- #define wxBACKINGSTORE wxRETAINED
- #define wxPOPUP_WINDOW 0x00020000
- #define wxFULL_REPAINT_ON_RESIZE 0x00010000
- #define wxNO_FULL_REPAINT_ON_RESIZE 0
- #define wxWINDOW_STYLE_MASK
- #define wxWS_EX_VALIDATE_RECURSIVELY 0x00000001
- #define wxWS_EX_BLOCK_EVENTS 0x00000002

- #define wxWS_EX_TRANSIENT 0x00000004
- #define wxWS_EX_THEMED_BACKGROUND 0x00000008
- #define wxWS_EX_PROCESS_IDLE 0x00000010
- #define wxWS_EX_PROCESS_UI_UPDATES 0x00000020
- #define wxFRAME_EX_METAL 0x00000040
- #define wxDIALOG_EX_METAL 0x00000040
- #define wxWS_EX_CONTEXTHELP 0x00000080
- #define wxFRAME_EX_CONTEXTHELP wxWS_EX_CONTEXTHELP
- #define wxDIALOG_EX_CONTEXTHELP wxWS_EX_CONTEXTHELP
- #define wxFRAME_DRAWER 0x0020
- #define wxFRAME_NO_WINDOW_MENU 0x0100
- #define wxMB_DOCKABLE 0x0001
- #define wxMENU_TEAROFF 0x0001
- #define wxCOLOURED 0x0800
- #define wxFIXED_LENGTH 0x0400
- #define wxLB_SORT 0x0010
- #define wxLB_SINGLE 0x0020
- #define wxLB_MULTIPLE 0x0040
- #define wxLB_EXTENDED 0x0080
- #define wxLB_NEEDED_SB 0x0000
- #define wxLB_OWNERDRAW 0x0100
- #define wxLB_ALWAYS_SB 0x0200
- #define wxLB_NO_SB 0x0400
- #define wxLB_HSCROLL wxHSCROLL
- #define wxLB_INT_HEIGHT 0x0800
- #define wxCB_SIMPLE 0x0004
- #define wxCB_SORT 0x0008
- #define wxCB_READONLY 0x0010
- #define wxCB_DROPDOWN 0x0020
- #define wxRA_LEFTTORIGHT 0x0001
- #define wxRA_TOPTOBOTTOM 0x0002
- #define wxRA_SPECIFY_COLS wxHORIZONTAL
- #define wxRA_SPECIFY_ROWS wxVERTICAL
- #define wxRA_HORIZONTAL wxHORIZONTAL
- #define wxRA_VERTICAL wxVERTICAL
- #define wxRB_GROUP 0x0004
- #define wxRB_SINGLE 0x0008
- #define wxSB_HORIZONTAL wxHORIZONTAL
- #define wxSB_VERTICAL wxVERTICAL
- #define wxSP_HORIZONTAL wxHORIZONTAL /* 4 */
- #define wxSP_VERTICAL wxVERTICAL /* 8 */
- #define wxSP_ARROW_KEYS 0x4000
- #define wxSP_WRAP 0x8000
- #define wxTC_RIGHTJUSTIFY 0x0010
- #define wxTC_FIXEDWIDTH 0x0020
- #define wxTC_TOP 0x0000 /* default */
- #define wxTC_LEFT 0x0020
- #define wxTC_RIGHT 0x0040
- #define wxTC_BOTTOM 0x0080
- #define wxTC_MULTILINE 0x0200 /* == wxNB_MULTILINE */
- #define wxTC_OWNERDRAW 0x0400
- #define wxBI_EXPAND wxEXPAND
- #define wxLI_HORIZONTAL wxHORIZONTAL
- #define wxLI_VERTICAL wxVERTICAL
- #define wxYES 0x00000002

- #define wxOK 0x00000004
- #define wxNO 0x00000008
- #define wxYES_NO (wxYES | wxNO)
- #define wxCANCEL 0x00000010
- #define wxAPPLY 0x00000020
- #define wxCLOSE 0x00000040
- #define wxOK_DEFAULT 0x00000000 /* has no effect (default) */
- #define wxYES_DEFAULT 0x00000000 /* has no effect (default) */
- #define wxNO_DEFAULT 0x00000080 /* only valid with wxYES_NO */
- #define wxCANCEL_DEFAULT 0x80000000 /* only valid with wxCANCEL */
- #define wxICON_EXCLAMATION 0x00000100
- #define wxICON_HAND 0x00000200
- #define wxICON_WARNING wxICON_EXCLAMATION
- #define wxICON_ERROR wxICON_HAND
- #define wxICON_QUESTION 0x00000400
- #define wxICON_INFORMATION 0x00000800
- #define wxICON_STOP wxICON_HAND
- #define wxICON_ASTERISK wxICON_INFORMATION
- #define wxHELP 0x00001000
- #define wxFORWARD 0x00002000
- #define wxBACKWARD 0x00004000
- #define wxRESET 0x00008000
- #define wxMORE 0x00010000
- #define wxSETUP 0x00020000
- #define wxICON_NONE 0x00040000
- #define wxICON_AUTH_NEEDED 0x00080000
- #define wxICON_MASK (wxICON_EXCLAMATION|wxICON_HAND|wxICON_QUESTION|wxICON_INFO↩ RMATION|wxICON_NONE)
- #define wxNOT_FOUND (-1)
- #define wxPRINT_QUALITY_HIGH -1

  Predefined print quality constants.
- #define wxPRINT_QUALITY_MEDIUM -2
- #define wxPRINT_QUALITY_LOW -3
- #define wxPRINT_QUALITY_DRAFT -4
- #define wxSTAY_ON_TOP 0x8000

  Top level window styles common to wxFrame and wxDialog.
- #define wxICONIZE 0x4000
- #define wxMINIMIZE wxICONIZE
- #define wxMAXIMIZE 0x2000
- #define wxCLOSE_BOX 0x1000
- #define wxSYSTEM_MENU 0x0800
- #define wxMINIMIZE_BOX 0x0400
- #define wxMAXIMIZE_BOX 0x0200
- #define wxTINY_CAPTION 0x0080
- #define wxRESIZE_BORDER 0x0040
- #define wxINT32_SWAP_ALWAYS(wxInt32_value)

  This macro will swap the bytes of the value variable from little endian to big endian or vice versa unconditionally, i.e.
- #define wxUINT32_SWAP_ALWAYS(wxUint32_value)
- #define wxINT16_SWAP_ALWAYS(wxInt16_value)
- #define wxUINT16_SWAP_ALWAYS(wxUint16_value)
- #define wxINT32_SWAP_ON_BE(wxInt32_value)

  This macro will swap the bytes of the value variable from little endian to big endian or vice versa if the program is compiled on a big-endian architecture (such as Sun work stations).
- #define wxUINT32_SWAP_ON_BE(wxUint32_value)

- #define wxINT16_SWAP_ON_BE(wxInt16_value)
- #define wxUINT16_SWAP_ON_BE(wxUint16_value)
- #define wxINT32_SWAP_ON_LE(wxInt32_value)

  This macro will swap the bytes of the value variable from little endian to big endian or vice versa if the program is compiled on a little-endian architecture (such as Intel PCs).
- #define wxUINT32_SWAP_ON_LE(wxUint32_value)
- #define wxINT16_SWAP_ON_LE(wxInt16_value)
- #define wxUINT16_SWAP_ON_LE(wxUint16_value)
- #define wxDECLARE_NO_ASSIGN_CLASS(classname)

  This macro can be used in a class declaration to disable the generation of default assignment operator.
- #define wxDECLARE_NO_COPY_CLASS(classname)

  This macro can be used in a class declaration to disable the generation of default copy ctor and assignment operator.
- #define wxDECLARE_NO_COPY_TEMPLATE_CLASS(classname, arg)

  Analog of wxDECLARE_NO_COPY_CLASS() for template classes.
- #define wxDECLARE_NO_COPY_TEMPLATE_CLASS_2(classname, arg1, arg2)

  Analog of wxDECLARE_NO_COPY_TEMPLATE_CLASS() for templates with 2 parameters.
- #define wxDEPRECATED(function)

  Generate deprecation warning with the given message when a function is used.
- #define wxDEPRECATED_BUT_USED_INTERNALLY(function)

  This is a special version of wxDEPRECATED() macro which only does something when the deprecated function is used from the code outside wxWidgets itself but doesn't generate warnings when it is used from wxWidgets.
- #define wxDEPRECATED_INLINE(func, body)

  This macro is similar to wxDEPRECATED() but can be used to not only declare the function function as deprecated but to also provide its (inline) implementation body.
- #define wxDEPRECATED_ACCESSOR(func, what)

  A helper macro allowing to easily define a simple deprecated accessor.
- #define wxDEPRECATED_BUT_USED_INTERNALLY_INLINE(func, body)

  Combination of wxDEPRECATED_BUT_USED_INTERNALLY() and wxDEPRECATED_INLINE().
- #define wxEXPLICIT

  `wxEXPLICIT` is a macro which expands to the C++ `explicit` keyword if the compiler supports it or nothing otherwise.
- #define wxOVERRIDE

  `wxOVERRIDE` expands to the C++11 `override` keyword if it's supported by the compiler or nothing otherwise.
- #define wxSUPPRESS_GCC_PRIVATE_DTOR_WARNING(name)

  GNU C++ compiler gives a warning for any class whose destructor is private unless it has a friend.

- #define wxINT8_MIN CHAR_MIN

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT8_MAX CHAR_MAX

  C99-like sized MIN/MAX constants for all integer types.
- #define wxUINT8_MAX UCHAR_MAX

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT16_MIN SHRT_MIN

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT16_MAX SHRT_MAX

  C99-like sized MIN/MAX constants for all integer types.
- #define wxUINT16_MAX USHRT_MAX

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT32_MIN INT_MIN-or-LONG_MIN

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT32_MAX INT_MAX-or-LONG_MAX

  C99-like sized MIN/MAX constants for all integer types.

- #define wxUINT32_MAX UINT_MAX-or-LONG_MAX

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT64_MIN LLONG_MIN

  C99-like sized MIN/MAX constants for all integer types.
- #define wxINT64_MAX LLONG_MAX

  C99-like sized MIN/MAX constants for all integer types.
- #define wxUINT64_MAX ULLONG_MAX

  C99-like sized MIN/MAX constants for all integer types.

Typedefs

- typedef int wxPrintQuality

  Specifies the print quality as either a predefined level or explicit resolution.
- typedef int wxCoord

  The type for screen and DC coordinates.
- typedef float wxFloat32

  32 bit IEEE float (1 sign, 8 exponent bits, 23 fraction bits).
- typedef double wxFloat64

  64 bit IEEE float (1 sign, 11 exponent bits, 52 fraction bits).
- typedef double wxDouble

  Native fastest representation that has at least wxFloat64 precision, so use the IEEE types for storage, and this for calculations.

- typedef signed char wxInt8

  8 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef unsigned char wxUint8

  8 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef wxUint8 wxByte

  8 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

- typedef signed short wxInt16

  16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef unsigned short wxUint16

  16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef wxUint16 wxWord

  16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef wxUint16 wxChar16

  16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

- typedef int wxInt32

  32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef unsigned int wxUint32

  32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef wxUint32 wxDword

  32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef wxUint32 wxChar32

  32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

- typedef wxLongLong_t wxInt64

  64 bit type (the mapping is more complex than a simple `typedef` and is not shown here).
- typedef wxULongLong_t wxUint64

64 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

- typedef ssize_t wxIntPtr

  Signed and unsigned integral types big enough to contain all of `long`, `size_t` and `void*`.

- typedef size_t wxUIntPtr

  Signed and unsigned integral types big enough to contain all of `long`, `size_t` and `void*`.

Enumerations

- enum wxGeometryCentre {
  wxCENTRE = 0x0001,
  wxCENTER = wxCENTRE }

  Generic flags.

- enum wxOrientation {
  wxHORIZONTAL = 0x0004,
  wxVERTICAL = 0x0008,
  wxBOTH = wxVERTICAL | wxHORIZONTAL,
  wxORIENTATION_MASK = wxBOTH }

  A generic orientation value.

- enum wxDirection {
  wxLEFT = 0x0010,
  wxRIGHT = 0x0020,
  wxUP = 0x0040,
  wxDOWN = 0x0080,
  wxTOP = wxUP,
  wxBOTTOM = wxDOWN,
  wxNORTH = wxUP,
  wxSOUTH = wxDOWN,
  wxWEST = wxLEFT,
  wxEAST = wxRIGHT,
  wxALL = (wxUP | wxDOWN | wxRIGHT | wxLEFT),
  wxDIRECTION_MASK = wxALL }

  A generic direction value.

- enum wxAlignment {
  wxALIGN_INVALID = -1,
  wxALIGN_NOT = 0x0000,
  wxALIGN_CENTER_HORIZONTAL = 0x0100,
  wxALIGN_CENTRE_HORIZONTAL = wxALIGN_CENTER_HORIZONTAL,
  wxALIGN_LEFT = wxALIGN_NOT,
  wxALIGN_TOP = wxALIGN_NOT,
  wxALIGN_RIGHT = 0x0200,
  wxALIGN_BOTTOM = 0x0400,
  wxALIGN_CENTER_VERTICAL = 0x0800,
  wxALIGN_CENTRE_VERTICAL = wxALIGN_CENTER_VERTICAL,
  wxALIGN_CENTER = (wxALIGN_CENTER_HORIZONTAL | wxALIGN_CENTER_VERTICAL),
  wxALIGN_CENTRE = wxALIGN_CENTER,
  wxALIGN_MASK = 0x0f00 }

  Generic alignment values.

- enum wxSizerFlagBits {
  wxFIXED_MINSIZE = 0x8000,
  wxRESERVE_SPACE_EVEN_IF_HIDDEN = 0x0002,
  wxSIZER_FLAG_BITS_MASK = 0x8002 }

  Miscellaneous flags for wxSizer items.

- enum wxStretch {
  wxSTRETCH_NOT = 0x0000,
  wxSHRINK = 0x1000,
  wxGROW = 0x2000,
  wxEXPAND = wxGROW,
  wxSHAPED = 0x4000,
  wxTILE = wxSHAPED | wxFIXED_MINSIZE,
  wxSTRETCH_MASK = 0x7000 }

  Generic stretch values.

- enum wxBorder {
  wxBORDER_DEFAULT = 0,
  wxBORDER_NONE = 0x00200000,
  wxBORDER_STATIC = 0x01000000,
  wxBORDER_SIMPLE = 0x02000000,
  wxBORDER_RAISED = 0x04000000,
  wxBORDER_SUNKEN = 0x08000000,
  wxBORDER_DOUBLE = 0x10000000,
  wxBORDER_THEME = wxBORDER_DOUBLE,
  wxBORDER_MASK = 0x1f200000 }

  Border flags for wxWindow.

- enum wxBackgroundStyle {
  wxBG_STYLE_ERASE,
  wxBG_STYLE_SYSTEM,
  wxBG_STYLE_PAINT,
  wxBG_STYLE_COLOUR,
  wxBG_STYLE_TRANSPARENT }

  Background styles.

- enum wxStandardID {
  wxID_AUTO_LOWEST,
  wxID_AUTO_HIGHEST,
  wxID_NONE = -3,
  wxID_SEPARATOR = -2,
  wxID_ANY = -1,
  wxID_LOWEST = 4999,
  wxID_OPEN,
  wxID_CLOSE,
  wxID_NEW,
  wxID_SAVE,
  wxID_SAVEAS,
  wxID_REVERT,
  wxID_EXIT,
  wxID_UNDO,
  wxID_REDO,
  wxID_HELP,
  wxID_PRINT,
  wxID_PRINT_SETUP,
  wxID_PAGE_SETUP,
  wxID_PREVIEW,
  wxID_ABOUT,
  wxID_HELP_CONTENTS,
  wxID_HELP_INDEX,
  wxID_HELP_SEARCH,
  wxID_HELP_COMMANDS,
  wxID_HELP_PROCEDURES,
  wxID_HELP_CONTEXT,
  wxID_CLOSE_ALL,
  wxID_PREFERENCES,
  wxID_EDIT = 5030,
  wxID_CUT,
  wxID_COPY,
  wxID_PASTE,
  wxID_CLEAR,
  wxID_FIND,
  wxID_DUPLICATE,
  wxID_SELECTALL,
  wxID_DELETE,
  wxID_REPLACE,
  wxID_REPLACE_ALL,
  wxID_PROPERTIES,
  wxID_VIEW_DETAILS,
  wxID_VIEW_LARGEICONS,
  wxID_VIEW_SMALLICONS,
  wxID_VIEW_LIST,
  wxID_VIEW_SORTDATE,
  wxID_VIEW_SORTNAME,
  wxID_VIEW_SORTSIZE,
  wxID_VIEW_SORTTYPE,
  wxID_FILE = 5050,
  wxID_FILE1,
  wxID_FILE2,
  wxID_FILE3,
  wxID_FILE4,
  wxID_FILE5,
  wxID_FILE6,
  wxID_FILE7,
  wxID_FILE8,
  wxID_FILE9,
  wxID_OK = 5100,
  wxID_CANCEL,
  wxID_APPLY,
  wxID_YES,

Standard IDs.

- enum wxItemKind {
  wxITEM_SEPARATOR = -1,
  wxITEM_NORMAL,
  wxITEM_CHECK,
  wxITEM_RADIO,
  wxITEM_DROPDOWN,
  wxITEM_MAX }

Item kinds for use with wxMenu, wxMenuItem, and wxToolBar.

- enum wxHitTest {
  wxHT_NOWHERE,
  wxHT_SCROLLBAR_FIRST = wxHT_NOWHERE,
  wxHT_SCROLLBAR_ARROW_LINE_1,
  wxHT_SCROLLBAR_ARROW_LINE_2,
  wxHT_SCROLLBAR_ARROW_PAGE_1,
  wxHT_SCROLLBAR_ARROW_PAGE_2,
  wxHT_SCROLLBAR_THUMB,
  wxHT_SCROLLBAR_BAR_1,
  wxHT_SCROLLBAR_BAR_2,
  wxHT_SCROLLBAR_LAST,
  wxHT_WINDOW_OUTSIDE,
  wxHT_WINDOW_INSIDE,
  wxHT_WINDOW_VERT_SCROLLBAR,
  wxHT_WINDOW_HORZ_SCROLLBAR,
  wxHT_WINDOW_CORNER,
  wxHT_MAX }

Generic hit test results.

- enum wxDataFormatId {
  wxDF_INVALID = 0,
  wxDF_TEXT = 1,
  wxDF_BITMAP = 2,
  wxDF_METAFILE = 3,
  wxDF_SYLK = 4,
  wxDF_DIF = 5,
  wxDF_TIFF = 6,
  wxDF_OEMTEXT = 7,
  wxDF_DIB = 8,
  wxDF_PALETTE = 9,
  wxDF_PENDATA = 10,
  wxDF_RIFF = 11,
  wxDF_WAVE = 12,
  wxDF_UNICODETEXT = 13,
  wxDF_ENHMETAFILE = 14,
  wxDF_FILENAME = 15,
  wxDF_LOCALE = 16,
  wxDF_PRIVATE = 20,
  wxDF_HTML = 30,
  wxDF_MAX }

Data format IDs used by wxDataFormat.

- enum wxKeyCode {
  WXK_NONE = 0,
  WXK_CONTROL_A = 1,
  WXK_CONTROL_B,
  WXK_CONTROL_C,
  WXK_CONTROL_D,
  WXK_CONTROL_E,
  WXK_CONTROL_F,
  WXK_CONTROL_G,
  WXK_CONTROL_H,
  WXK_CONTROL_I,
  WXK_CONTROL_J,
  WXK_CONTROL_K,
  WXK_CONTROL_L,
  WXK_CONTROL_M,
  WXK_CONTROL_N,
  WXK_CONTROL_O,
  WXK_CONTROL_P,
  WXK_CONTROL_Q,
  WXK_CONTROL_R,
  WXK_CONTROL_S,
  WXK_CONTROL_T,
  WXK_CONTROL_U,
  WXK_CONTROL_V,
  WXK_CONTROL_W,
  WXK_CONTROL_X,
  WXK_CONTROL_Y,
  WXK_CONTROL_Z,
  WXK_BACK = 8,
  WXK_TAB = 9,
  WXK_RETURN = 13,
  WXK_ESCAPE = 27,
  WXK_SPACE = 32,
  WXK_DELETE = 127,
  WXK_START = 300,
  WXK_LBUTTON,
  WXK_RBUTTON,
  WXK_CANCEL,
  WXK_MBUTTON,
  WXK_CLEAR,
  WXK_SHIFT,
  WXK_ALT,
  WXK_CONTROL,
  WXK_RAW_CONTROL,
  WXK_MENU,
  WXK_PAUSE,
  WXK_CAPITAL,
  WXK_END,
  WXK_HOME,
  WXK_LEFT,
  WXK_UP,
  WXK_RIGHT,
  WXK_DOWN,
  WXK_SELECT,
  WXK_PRINT,
  WXK_EXECUTE,
  WXK_SNAPSHOT,
  WXK_INSERT,
  WXK_HELP,
  WXK_NUMPAD0,
  WXK_NUMPAD1,
  WXK_NUMPAD2,
  WXK_NUMPAD3,
  WXK_NUMPAD4,
  WXK_NUMPAD5,

Virtual keycodes used by wxKeyEvent and some other wxWidgets functions.

- enum wxKeyModifier {
  wxMOD_NONE = 0x0000,
  wxMOD_ALT = 0x0001,
  wxMOD_CONTROL = 0x0002,
  wxMOD_ALTGR = wxMOD_ALT | wxMOD_CONTROL,
  wxMOD_SHIFT = 0x0004,
  wxMOD_META = 0x0008,
  wxMOD_WIN = wxMOD_META,
  wxMOD_RAW_CONTROL,
  wxMOD_CMD = wxMOD_CONTROL,
  wxMOD_ALL = 0xffff }

This enum contains bit mask constants used in wxKeyEvent.

- enum wxPaperSize {
  wxPAPER_10X11,
  wxPAPER_10X14,
  wxPAPER_11X17,
  wxPAPER_12X11,
  wxPAPER_15X11,
  wxPAPER_9X11,
  wxPAPER_A2,
  wxPAPER_A3,
  wxPAPER_A3_EXTRA,
  wxPAPER_A3_EXTRA_TRANSVERSE,
  wxPAPER_A3_ROTATED,
  wxPAPER_A3_TRANSVERSE,
  wxPAPER_A4,
  wxPAPER_A4SMALL,
  wxPAPER_A4_EXTRA,
  wxPAPER_A4_PLUS,
  wxPAPER_A4_ROTATED,
  wxPAPER_A4_TRANSVERSE,
  wxPAPER_A5,
  wxPAPER_A5_EXTRA,
  wxPAPER_A5_ROTATED,
  wxPAPER_A5_TRANSVERSE,
  wxPAPER_A6,
  wxPAPER_A6_ROTATED,
  wxPAPER_A_PLUS,
  wxPAPER_B4,
  wxPAPER_B4_JIS_ROTATED,
  wxPAPER_B5,
  wxPAPER_B5_EXTRA,
  wxPAPER_B5_JIS_ROTATED,
  wxPAPER_B5_TRANSVERSE,
  wxPAPER_B6_JIS,
  wxPAPER_B6_JIS_ROTATED,
  wxPAPER_B_PLUS,
  wxPAPER_CSHEET,
  wxPAPER_DBL_JAPANESE_POSTCARD,
  wxPAPER_DBL_JAPANESE_POSTCARD_ROTATED,
  wxPAPER_DSHEET,
  wxPAPER_ENV_10,
  wxPAPER_ENV_11,
  wxPAPER_ENV_12,
  wxPAPER_ENV_14,
  wxPAPER_ENV_9,
  wxPAPER_ENV_B4,
  wxPAPER_ENV_B5,
  wxPAPER_ENV_B6,
  wxPAPER_ENV_C3,
  wxPAPER_ENV_C4,
  wxPAPER_ENV_C5,
  wxPAPER_ENV_C6,
  wxPAPER_ENV_C65,
  wxPAPER_ENV_DL,
  wxPAPER_ENV_INVITE,
  wxPAPER_ENV_ITALY,
  wxPAPER_ENV_MONARCH,
  wxPAPER_ENV_PERSONAL,
  wxPAPER_ESHEET,
  wxPAPER_EXECUTIVE,
  wxPAPER_FANFOLD_LGL_GERMAN,
  wxPAPER_FANFOLD_STD_GERMAN,
  wxPAPER_FANFOLD_US,
  wxPAPER_FOLIO,
  wxPAPER_ISO_B4,
  wxPAPER_JAPANESE_POSTCARD,

Paper size types for use with the printing framework.

- enum wxPrintOrientation {
  wxPORTRAIT,
  wxLANDSCAPE }

  Printing orientation.
- enum wxDuplexMode {
  wxDUPLEX_SIMPLEX,
  wxDUPLEX_HORIZONTAL,
  wxDUPLEX_VERTICAL }

  Duplex printing modes.
- enum wxPrintMode {
  wxPRINT_MODE_NONE = 0,
  wxPRINT_MODE_PREVIEW = 1,
  wxPRINT_MODE_FILE = 2,
  wxPRINT_MODE_PRINTER = 3,
  wxPRINT_MODE_STREAM = 4 }

  Print mode (currently PostScript only).
- enum wxUpdateUI {
  wxUPDATE_UI_NONE,
  wxUPDATE_UI_RECURSE,
  wxUPDATE_UI_FROMIDLE }

  Flags which can be used in wxWindow::UpdateWindowUI().

Functions

- template<typename T >
  wxDELETE (T *&ptr)

  A function which deletes and nulls the pointer.
- template<typename T >
  wxDELETEA (T *&array)

  A function which deletes and nulls the pointer.
- template<typename T >
  wxSwap (T &first, T &second)

  Swaps the contents of two variables.
- void wxVaCopy (va_list argptrDst, va_list argptrSrc)

  This macro is the same as the standard C99 `va_copy` for the compilers which support it or its replacement for those that don't.

Variables

- wxCoord wxDefaultCoord = -1

  A special value meaning "use default coordinate".

22.250.1 Macro Definition Documentation

#define wxALWAYS_SHOW_SB 0x00800000

#define wxAPPLY 0x00000020

#define wxBACKINGSTORE wxRETAINED

#define wxBACKWARD 0x00004000

#define wxBI_EXPAND wxEXPAND

#define wxBORDER wxBORDER_SIMPLE

#define wxCANCEL 0x00000010

#define wxCANCEL_DEFAULT 0x80000000 /∗ only valid with wxCANCEL ∗/

#define wxCAPTION 0x20000000

#define wxCB_DROPDOWN 0x0020

#define wxCB_READONLY 0x0010

#define wxCB_SIMPLE 0x0004

#define wxCB_SORT 0x0008

#define wxCLIP_CHILDREN 0x00400000

#define wxCLIP_SIBLINGS 0x20000000

#define wxCLOSE 0x00000040

#define wxCLOSE_BOX 0x1000

#define wxCOLOURED 0x0800

#define wxDIALOG_EX_CONTEXTHELP wxWS_EX_CONTEXTHELP

#define wxDIALOG_EX_METAL 0x00000040

#define wxDOUBLE_BORDER wxBORDER_DOUBLE

#define wxFIXED_LENGTH 0x0400

#define wxFORWARD 0x00002000

#define wxFRAME_DRAWER 0x0020

#define wxFRAME_EX_CONTEXTHELP wxWS_EX_CONTEXTHELP

#define wxFRAME_EX_METAL 0x00000040

#define wxFRAME_NO_WINDOW_MENU 0x0100

#define wxFULL_REPAINT_ON_RESIZE 0x00010000

#define wxHELP 0x00001000

#define wxHSCROLL 0x40000000

#define wxICON_ASTERISK wxICON_INFORMATION

#define wxICON_AUTH_NEEDED 0x00080000

#define wxICON_ERROR wxICON_HAND

#define wxICON_EXCLAMATION 0x00000100

#define wxICON_HAND 0x00000200

#define wxICON_INFORMATION 0x00000800

#define wxICON_MASK (wxICON_EXCLAMATION|wxICON_HAND|wxICON_QUESTION|wxICON_INFORMATI↩ ON|wxICON_NONE)

#define wxICON_NONE 0x00040000

#define wxICON_QUESTION 0x00000400

#define wxICON_STOP wxICON_HAND

#define wxICON_WARNING wxICON_EXCLAMATION

#define wxICONIZE 0x4000

#define wxINT16_MAX SHRT_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxINT16_MIN SHRT_MIN

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxINT32_MAX INT_MAX-or-LONG_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxINT32_MIN INT_MIN-or-LONG_MIN

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxINT64_MAX LLONG_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxINT64_MIN LLONG_MIN

C99-like sized MIN/MAX constants for all integer types.

For each n in the set 8, 16, 32, 64 we define wxINTn_MIN, wxINTn_MAX and wxUINTc_MAX (wxUINTc_MIN is always 0 and so is not defined).

#define wxINT8_MAX CHAR_MAX

C99-like sized MIN/MAX constants for all integer types.

For each n in the set 8, 16, 32, 64 we define wxINTn_MIN, wxINTn_MAX and wxUINTc_MAX (wxUINTc_MIN is always 0 and so is not defined).

#define wxINT8_MIN CHAR_MIN

C99-like sized MIN/MAX constants for all integer types.

For each n in the set 8, 16, 32, 64 we define wxINTn_MIN, wxINTn_MAX and wxUINTc_MAX (wxUINTc_MIN is always 0 and so is not defined).

#define wxLB_ALWAYS_SB 0x0200

#define wxLB_EXTENDED 0x0080

#define wxLB_HSCROLL wxHSCROLL

#define wxLB_INT_HEIGHT 0x0800

#define wxLB_MULTIPLE 0x0040

#define wxLB_NEEDED_SB 0x0000

#define wxLB_NO_SB 0x0400

#define wxLB_OWNERDRAW 0x0100

#define wxLB_SINGLE 0x0020

#define wxLB_SORT 0x0010

#define wxLI_HORIZONTAL wxHORIZONTAL

#define wxLI_VERTICAL wxVERTICAL

#define wxMAXIMIZE 0x2000

#define wxMAXIMIZE_BOX 0x0200

#define wxMB_DOCKABLE 0x0001

#define wxMENU_TEAROFF 0x0001

#define wxMINIMIZE wxICONIZE

#define wxMINIMIZE_BOX 0x0400

#define wxMORE 0x00010000

#define wxNO 0x00000008

#define wxNO_BORDER wxBORDER_NONE

#define wxNO_DEFAULT 0x00000080 /* only valid with wxYES_NO */

#define wxNO_FULL_REPAINT_ON_RESIZE 0

#define wxNOT_FOUND (-1)

#define wxOK 0x00000004

#define wxOK_DEFAULT 0x00000000 /* has no effect (default) */

#define wxPOPUP_WINDOW 0x00020000

#define wxPRINT_QUALITY_DRAFT -4

#define wxPRINT_QUALITY_HIGH -1

Predefined print quality constants.

See also

wxPrintQuality

#define wxPRINT_QUALITY_LOW -3

#define wxPRINT_QUALITY_MEDIUM -2

#define wxRA_HORIZONTAL wxHORIZONTAL

#define wxRA_LEFTTORIGHT 0x0001

#define wxRA_SPECIFY_COLS wxHORIZONTAL

#define wxRA_SPECIFY_ROWS wxVERTICAL

#define wxRA_TOPTOBOTTOM 0x0002

#define wxRA_VERTICAL wxVERTICAL

#define wxRAISED_BORDER wxBORDER_RAISED

#define wxRB_GROUP 0x0004

#define wxRB_SINGLE 0x0008

#define wxRESET 0x00008000

#define wxRESIZE_BORDER 0x0040

#define wxRETAINED 0x00000000

#define wxSB_HORIZONTAL wxHORIZONTAL

#define wxSB_VERTICAL wxVERTICAL

#define wxSETUP 0x00020000

#define wxSIMPLE_BORDER wxBORDER_SIMPLE

#define wxSIZE_ALLOW_MINUS_ONE 0x0004

#define wxSIZE_AUTO (wxSIZE_AUTO_WIDTH|wxSIZE_AUTO_HEIGHT)

#define wxSIZE_AUTO_HEIGHT 0x0002

#define wxSIZE_AUTO_WIDTH 0x0001

#define wxSIZE_FORCE 0x0010

#define wxSIZE_FORCE_EVENT 0x0020

#define wxSIZE_NO_ADJUSTMENTS 0x0008

#define wxSIZE_USE_EXISTING 0x0000

#define wxSP_ARROW_KEYS 0x4000

#define wxSP_HORIZONTAL wxHORIZONTAL /* 4 */

#define wxSP_VERTICAL wxVERTICAL /* 8 */

#define wxSP_WRAP 0x8000

#define wxSTATIC_BORDER wxBORDER_STATIC

#define wxSTAY_ON_TOP 0x8000

Top level window styles common to wxFrame and wxDialog.

#define wxSUNKEN_BORDER wxBORDER_SUNKEN

#define wxSYSTEM_MENU 0x0800

#define wxTAB_TRAVERSAL 0x00080000

#define wxTC_BOTTOM 0x0080

#define wxTC_FIXEDWIDTH 0x0020

#define wxTC_LEFT 0x0020

#define wxTC_MULTILINE 0x0200 /* == wxNB_MULTILINE */

#define wxTC_OWNERDRAW 0x0400

#define wxTC_RIGHT 0x0040

#define wxTC_RIGHTJUSTIFY 0x0010

#define wxTC_TOP 0x0000 /* default */

#define wxTINY_CAPTION 0x0080

#define wxTRANSPARENT_WINDOW 0x00100000

#define wxUINT16_MAX USHRT_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxUINT32_MAX UINT_MAX-or-LONG_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxUINT64_MAX ULLONG_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxUINT8_MAX UCHAR_MAX

C99-like sized MIN/MAX constants for all integer types.

For each `n` in the set 8, 16, 32, 64 we define `wxINTn_MIN`, `wxINTn_MAX` and `wxUINTc_MAX` (`wxUINTc_MIN` is always 0 and so is not defined).

#define wxVSCROLL 0x80000000

#define wxWANTS_CHARS 0x00040000

#define wxWINDOW_STYLE_MASK

Value:

```
(wxVSCROLL|wxHSCROLL|wxBORDER_MASK|
      wxALWAYS_SHOW_SB|wxCLIP_CHILDREN| \
     wxCLIP_SIBLINGS|wxTRANSPARENT_WINDOW|
      wxTAB_TRAVERSAL|wxWANTS_CHARS| \
     wxRETAINED|wxPOPUP_WINDOW|
      wxFULL_REPAINT_ON_RESIZE)
```

#define wxWS_EX_BLOCK_EVENTS 0x00000002

#define wxWS_EX_CONTEXTHELP 0x00000080

#define wxWS_EX_PROCESS_IDLE 0x00000010

#define wxWS_EX_PROCESS_UI_UPDATES 0x00000020

#define wxWS_EX_THEMED_BACKGROUND 0x00000008

#define wxWS_EX_TRANSIENT 0x00000004

#define wxWS_EX_VALIDATE_RECURSIVELY 0x00000001

#define wxYES 0x00000002

#define wxYES_DEFAULT 0x00000000 /* has no effect (default) */

#define wxYES_NO (wxYES | wxNO)

22.250.2 Typedef Documentation

typedef wxUint8 wxByte

8 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef wxUint16 wxChar16

16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef wxUint32 wxChar32

32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef int wxCoord

The type for screen and DC coordinates.

typedef double wxDouble

Native fastest representation that has at least wxFloat64 precision, so use the IEEE types for storage, and this for calculations.

(The mapping is more complex than a simple `typedef` and is not shown here).

typedef wxUint32 wxDword

32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef float wxFloat32

32 bit IEEE float (1 sign, 8 exponent bits, 23 fraction bits).

(The mapping is more complex than a simple `typedef` and is not shown here).

typedef double wxFloat64

64 bit IEEE float (1 sign, 11 exponent bits, 52 fraction bits).

(The mapping is more complex than a simple `typedef` and is not shown here).

typedef signed short wxInt16

16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef int wxInt32

32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef wxLongLong_t wxInt64

64 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef signed char wxInt8

8 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef ssize_t wxIntPtr

Signed and unsigned integral types big enough to contain all of `long`, `size_t` and `void*`.

(The mapping is more complex than a simple `typedef` and is not shown here).

typedef int wxPrintQuality

Specifies the print quality as either a predefined level or explicit resolution.

The print quality may be one of wxPRINT_QUALITY_HIGH, wxPRINT_QUALITY_MEDIUM, wxPRINT_QUALITY_LOW or wxPRINT_QUALITY_DRAFT (which are all negative) or express the desired resolution, in DPI, e.g. 600.

typedef unsigned short wxUint16

16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef unsigned int wxUint32

32 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef wxULongLong_t wxUint64

64 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef unsigned char wxUint8

8 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

typedef size_t wxUIntPtr

Signed and unsigned integral types big enough to contain all of `long`, `size_t` and `void*`.

(The mapping is more complex than a simple `typedef` and is not shown here).

typedef wxUint16 wxWord

16 bit type (the mapping is more complex than a simple `typedef` and is not shown here).

22.250.3 Enumeration Type Documentation

enum wxAlignment

Generic alignment values.

Can be combined together.

Enumerator

wxALIGN_INVALID A value different from any valid alignment value. Note that you shouldn't use 0 for this as it's the value of (valid) alignments wxALIGN_LEFT and wxALIGN_TOP.

Since

2.9.1

wxALIGN_NOT

wxALIGN_CENTER_HORIZONTAL

wxALIGN_CENTRE_HORIZONTAL

wxALIGN_LEFT

wxALIGN_TOP

wxALIGN_RIGHT

wxALIGN_BOTTOM

wxALIGN_CENTER_VERTICAL

wxALIGN_CENTRE_VERTICAL

wxALIGN_CENTER

wxALIGN_CENTRE

wxALIGN_MASK A mask to extract alignment from the combination of flags.

enum wxBackgroundStyle

Background styles.

See also

wxWindow::SetBackgroundStyle()

Enumerator

wxBG_STYLE_ERASE Default background style value indicating that the background may be erased in the user-defined EVT_ERASE_BACKGROUND handler. If no such handler is defined (or if it skips the event), the effect of this style is the same as wxBG_STYLE_SYSTEM. If an empty handler (*not* skipping the event) is defined, the effect is the same as wxBG_STYLE_PAINT, i.e. the background is not erased at all until EVT_PAINT handler is executed.

This is the only background style value for which erase background events are generated at all.

wxBG_STYLE_SYSTEM Use the default background, as determined by the system or the current theme. If the window has been assigned a non-default background colour, it will be used for erasing its background. Otherwise the default background (which might be a gradient or a pattern) will be used.

EVT_ERASE_BACKGROUND event will not be generated at all for windows with this style.

wxBG_STYLE_PAINT Indicates that the background is only erased in the user-defined EVT_PAINT handler. Using this style avoids flicker which would result from redrawing the background twice if the EVT_PAINT handler entirely overwrites it. It must not be used however if the paint handler leaves any parts of the window unpainted as their contents is then undetermined. Only use it if you repaint the whole window in your handler.

EVT_ERASE_BACKGROUND event will not be generated at all for windows with this style.

wxBG_STYLE_COLOUR

wxBG_STYLE_TRANSPARENT Indicates that the window background is not erased, letting the parent window show through. Currently this style is only supported in wxOSX and wxGTK with compositing available, see wxWindow::IsTransparentBackgroundSupported().

enum wxBorder

Border flags for wxWindow.

Enumerator

wxBORDER_DEFAULT This is different from wxBORDER_NONE as by default the controls do have a border.

wxBORDER_NONE

wxBORDER_STATIC

wxBORDER_SIMPLE

wxBORDER_RAISED

wxBORDER_SUNKEN

wxBORDER_DOUBLE

wxBORDER_THEME

wxBORDER_MASK

enum wxDataFormatId

Data format IDs used by wxDataFormat.

Enumerator

wxDF_INVALID

wxDF_TEXT

wxDF_BITMAP

wxDF_METAFILE

wxDF_SYLK

wxDF_DIF

wxDF_TIFF

wxDF_OEMTEXT

wxDF_DIB

wxDF_PALETTE

wxDF_PENDATA

wxDF_RIFF

wxDF_WAVE

wxDF_UNICODETEXT

wxDF_ENHMETAFILE

wxDF_FILENAME

wxDF_LOCALE

wxDF_PRIVATE

wxDF_HTML

wxDF_MAX

enum wxDirection

A generic direction value.

Enumerator

wxLEFT

wxRIGHT

wxUP

wxDOWN

wxTOP

wxBOTTOM

wxNORTH

wxSOUTH

wxWEST

wxEAST

wxALL

wxDIRECTION_MASK A mask to extract direction from the combination of flags.

enum wxDuplexMode

Duplex printing modes.

Enumerator

wxDUPLEX_SIMPLEX Non-duplex.

wxDUPLEX_HORIZONTAL

wxDUPLEX_VERTICAL

enum wxGeometryCentre

Generic flags.

Enumerator

wxCENTRE

wxCENTER

enum wxHitTest

Generic hit test results.

Enumerator

wxHT_NOWHERE

wxHT_SCROLLBAR_FIRST

wxHT_SCROLLBAR_ARROW_LINE_1 left or upper arrow to scroll by line

wxHT_SCROLLBAR_ARROW_LINE_2 right or down

wxHT_SCROLLBAR_ARROW_PAGE_1 left or upper arrow to scroll by page

wxHT_SCROLLBAR_ARROW_PAGE_2 right or down

wxHT_SCROLLBAR_THUMB on the thumb

wxHT_SCROLLBAR_BAR_1 bar to the left/above the thumb

wxHT_SCROLLBAR_BAR_2 bar to the right/below the thumb

wxHT_SCROLLBAR_LAST

wxHT_WINDOW_OUTSIDE not in this window at all

wxHT_WINDOW_INSIDE in the client area

wxHT_WINDOW_VERT_SCROLLBAR on the vertical scrollbar

wxHT_WINDOW_HORZ_SCROLLBAR on the horizontal scrollbar

wxHT_WINDOW_CORNER on the corner between 2 scrollbars

wxHT_MAX

enum wxItemKind

Item kinds for use with wxMenu, wxMenuItem, and wxToolBar.

See also

wxMenu::Append(), wxMenuItem::wxMenuItem(), wxToolBar::AddTool()

Enumerator

wxITEM_SEPARATOR

wxITEM_NORMAL Normal tool button / menu item.

See also

wxToolBar::AddTool(), wxMenu::AppendItem().

wxITEM_CHECK Check (or toggle) tool button / menu item.

See also

wxToolBar::AddCheckTool(), wxMenu::AppendCheckItem().

wxITEM_RADIO Radio tool button / menu item.

See also

wxToolBar::AddRadioTool(), wxMenu::AppendRadioItem().

wxITEM_DROPDOWN Normal tool button with a dropdown arrow next to it. Clicking the dropdown arrow sends a `wxEVT_TOOL_DROPDOWN` event and may also display the menu previously associated with the item with wxToolBar::SetDropdownMenu(). Currently this type of tools is supported under MSW and GTK.

wxITEM_MAX

enum wxKeyCode

Virtual keycodes used by wxKeyEvent and some other wxWidgets functions.

Note that the range `0..255` corresponds to the characters of the current locale, in particular the `32..127` subrange is for the ASCII symbols, and all the special key values such as `WXK_END` lie above this range.

Enumerator

WXK_NONE No key. This value is returned by wxKeyEvent::GetKeyCode() if there is no non-Unicode representation for the pressed key (e.g. a Cyrillic letter was entered when not using a Cyrillic locale) and by wxKeyEvent::GetUnicodeKey() if there is no Unicode representation for the key (this happens for the special, non printable, keys only, e.g. WXK_HOME).

Since

2.9.2 (you can simply use 0 with previous versions).

WXK_CONTROL_A

WXK_CONTROL_B

WXK_CONTROL_C

WXK_CONTROL_D

WXK_CONTROL_E

WXK_CONTROL_F

WXK_CONTROL_G

WXK_CONTROL_H

WXK_CONTROL_I

WXK_CONTROL_J

WXK_CONTROL_K

WXK_CONTROL_L

WXK_CONTROL_M

WXK_CONTROL_N

WXK_CONTROL_O

WXK_CONTROL_P

WXK_CONTROL_Q

WXK_CONTROL_R

WXK_CONTROL_S

WXK_CONTROL_T

WXK_CONTROL_U

WXK_CONTROL_V

WXK_CONTROL_W

WXK_CONTROL_X

WXK_CONTROL_Y

WXK_CONTROL_Z

WXK_BACK Backspace.

WXK_TAB

WXK_RETURN

WXK_ESCAPE

WXK_SPACE

WXK_DELETE

WXK_START Special key values. These are, by design, not compatible with Unicode characters. If you want to get a Unicode character from a key event, use wxKeyEvent::GetUnicodeKey() instead.

WXK_LBUTTON

WXK_RBUTTON

WXK_CANCEL

WXK_MBUTTON

WXK_CLEAR

WXK_SHIFT

WXK_ALT

WXK_CONTROL Note that under Mac OS X, to improve compatibility with other systems, 'WXK_CONTROL' represents the 'Command' key. Use this constant to work with keyboard shortcuts. See 'WXK_RAW_C↩ ONTROL' to get the state of the actual 'Control' key.

WXK_RAW_CONTROL Under Mac OS X, where the 'Command' key is mapped to 'Control' to improve compatibility with other systems, WXK_RAW_CONTROL may be used to obtain the state of the actual 'Control' key ('WXK_CONTROL' would obtain the status of the 'Command' key). Under Windows/Linux/Others, this is equivalent to WXK_CONTROL

WXK_MENU

WXK_PAUSE

WXK_CAPITAL

WXK_END

WXK_HOME

WXK_LEFT

WXK_UP

WXK_RIGHT

WXK_DOWN

WXK_SELECT

WXK_PRINT

WXK_EXECUTE

WXK_SNAPSHOT

WXK_INSERT

WXK_HELP

WXK_NUMPAD0

WXK_NUMPAD1

WXK_NUMPAD2

WXK_NUMPAD3

WXK_NUMPAD4

WXK_NUMPAD5

WXK_NUMPAD6

WXK_NUMPAD7

WXK_NUMPAD8

WXK_NUMPAD9

WXK_MULTIPLY

WXK_ADD

WXK_SEPARATOR

WXK_SUBTRACT

WXK_DECIMAL

WXK_DIVIDE

WXK_F1

WXK_F2
WXK_F3
WXK_F4
WXK_F5
WXK_F6
WXK_F7
WXK_F8
WXK_F9
WXK_F10
WXK_F11
WXK_F12
WXK_F13
WXK_F14
WXK_F15
WXK_F16
WXK_F17
WXK_F18
WXK_F19
WXK_F20
WXK_F21
WXK_F22
WXK_F23
WXK_F24
WXK_NUMLOCK
WXK_SCROLL
WXK_PAGEUP
WXK_PAGEDOWN
WXK_NUMPAD_SPACE
WXK_NUMPAD_TAB
WXK_NUMPAD_ENTER
WXK_NUMPAD_F1
WXK_NUMPAD_F2
WXK_NUMPAD_F3
WXK_NUMPAD_F4
WXK_NUMPAD_HOME
WXK_NUMPAD_LEFT
WXK_NUMPAD_UP
WXK_NUMPAD_RIGHT
WXK_NUMPAD_DOWN
WXK_NUMPAD_PAGEUP
WXK_NUMPAD_PAGEDOWN
WXK_NUMPAD_END
WXK_NUMPAD_BEGIN
WXK_NUMPAD_INSERT
WXK_NUMPAD_DELETE

WXK_NUMPAD_EQUAL

WXK_NUMPAD_MULTIPLY

WXK_NUMPAD_ADD

WXK_NUMPAD_SEPARATOR

WXK_NUMPAD_SUBTRACT

WXK_NUMPAD_DECIMAL

WXK_NUMPAD_DIVIDE

WXK_WINDOWS_LEFT The following key codes are only generated under Windows currently.

WXK_WINDOWS_RIGHT

WXK_WINDOWS_MENU

WXK_COMMAND This special key code was used to represent the key used for keyboard shortcuts. Under Mac OS X, this key maps to the 'Command' (aka logo or 'Apple') key, whereas on Linux/Windows/others this is the Control key, with the new semantic of WXK_CONTROL, WXK_COMMAND is not needed anymore

WXK_SPECIAL1 Hardware-specific buttons.

WXK_SPECIAL2

WXK_SPECIAL3

WXK_SPECIAL4

WXK_SPECIAL5

WXK_SPECIAL6

WXK_SPECIAL7

WXK_SPECIAL8

WXK_SPECIAL9

WXK_SPECIAL10

WXK_SPECIAL11

WXK_SPECIAL12

WXK_SPECIAL13

WXK_SPECIAL14

WXK_SPECIAL15

WXK_SPECIAL16

WXK_SPECIAL17

WXK_SPECIAL18

WXK_SPECIAL19

WXK_SPECIAL20

enum wxKeyModifier

This enum contains bit mask constants used in wxKeyEvent.

Enumerator

wxMOD_NONE

wxMOD_ALT

wxMOD_CONTROL Ctlr Key, corresponds to Command key on OS X.

wxMOD_ALTGR

wxMOD_SHIFT

wxMOD_META

wxMOD_WIN

wxMOD_RAW_CONTROL used to describe the true Ctrl Key under OSX, identic to `wxMOD_CONTROL` on other platforms

wxMOD_CMD deprecated, identic to `wxMOD_CONTROL` on all platforms

wxMOD_ALL

enum wxOrientation

A generic orientation value.

Enumerator

wxHORIZONTAL

wxVERTICAL

wxBOTH A mask value to indicate both vertical and horizontal orientations.

wxORIENTATION_MASK A synonym for `wxBOTH`.

enum wxPaperSize

Paper size types for use with the printing framework.

See also

overview_printing, wxPrintData::SetPaperId()

Enumerator

wxPAPER_10X11 10 x 11 in

wxPAPER_10X14 10-by-14-inch sheet

wxPAPER_11X17 11-by-17-inch sheet

wxPAPER_12X11 12 x 11 in

wxPAPER_15X11 15 x 11 in

wxPAPER_9X11 9 x 11 in

wxPAPER_A2 A2 420 x 594 mm.

wxPAPER_A3 A3 sheet, 297 by 420 millimeters.

wxPAPER_A3_EXTRA A3 Extra 322 x 445 mm.

wxPAPER_A3_EXTRA_TRANSVERSE A3 Extra Transverse 322 x 445 mm.

wxPAPER_A3_ROTATED A3 Rotated 420 x 297 mm.

wxPAPER_A3_TRANSVERSE A3 Transverse 297 x 420 mm.

wxPAPER_A4 A4 Sheet, 210 by 297 millimeters.

wxPAPER_A4SMALL A4 small sheet, 210 by 297 millimeters.

wxPAPER_A4_EXTRA A4 Extra 9.27 x 12.69 in.

wxPAPER_A4_PLUS A4 Plus 210 x 330 mm.

wxPAPER_A4_ROTATED A4 Rotated 297 x 210 mm.

wxPAPER_A4_TRANSVERSE A4 Transverse 210 x 297 mm.

wxPAPER_A5 A5 sheet, 148 by 210 millimeters.

wxPAPER_A5_EXTRA A5 Extra 174 x 235 mm.

wxPAPER_A5_ROTATED A5 Rotated 210 x 148 mm.

wxPAPER_A5_TRANSVERSE A5 Transverse 148 x 210 mm.

wxPAPER_A6 A6 105 x 148 mm.

wxPAPER_A6_ROTATED A6 Rotated 148 x 105 mm.

wxPAPER_A_PLUS SuperA/SuperA/A4 227 x 356 mm.

wxPAPER_B4 B4 sheet, 250 by 354 millimeters.

wxPAPER_B4_JIS_ROTATED B4 (JIS) Rotated 364 x 257 mm.

wxPAPER_B5 B5 sheet, 182-by-257-millimeter paper.

wxPAPER_B5_EXTRA B5 (ISO) Extra 201 x 276 mm.

wxPAPER_B5_JIS_ROTATED B5 (JIS) Rotated 257 x 182 mm.

wxPAPER_B5_TRANSVERSE B5 (JIS) Transverse 182 x 257 mm.

wxPAPER_B6_JIS B6 (JIS) 128 x 182 mm.

wxPAPER_B6_JIS_ROTATED B6 (JIS) Rotated 182 x 128 mm.

wxPAPER_B_PLUS SuperB/SuperB/A3 305 x 487 mm.

wxPAPER_CSHEET C Sheet, 17 by 22 inches.

wxPAPER_DBL_JAPANESE_POSTCARD Japanese Double Postcard 200 x 148 mm.

wxPAPER_DBL_JAPANESE_POSTCARD_ROTATED Double Japanese Postcard Rotated 148 x 200 mm.

wxPAPER_DSHEET D Sheet, 22 by 34 inches.

wxPAPER_ENV_10 #10 Envelope, 4 1/8 by 9 1/2 inches

wxPAPER_ENV_11 #11 Envelope, 4 1/2 by 10 3/8 inches

wxPAPER_ENV_12 #12 Envelope, 4 3/4 by 11 inches

wxPAPER_ENV_14 #14 Envelope, 5 by 11 1/2 inches

wxPAPER_ENV_9 #9 Envelope, 3 7/8 by 8 7/8 inches

wxPAPER_ENV_B4 B4 Envelope, 250 by 353 millimeters.

wxPAPER_ENV_B5 B5 Envelope, 176 by 250 millimeters.

wxPAPER_ENV_B6 B6 Envelope, 176 by 125 millimeters.

wxPAPER_ENV_C3 C3 Envelope, 324 by 458 millimeters.

wxPAPER_ENV_C4 C4 Envelope, 229 by 324 millimeters.

wxPAPER_ENV_C5 C5 Envelope, 162 by 229 millimeters.

wxPAPER_ENV_C6 C6 Envelope, 114 by 162 millimeters.

wxPAPER_ENV_C65 C65 Envelope, 114 by 229 millimeters.

wxPAPER_ENV_DL DL Envelope, 110 by 220 millimeters.

wxPAPER_ENV_INVITE Envelope Invite 220 x 220 mm.

wxPAPER_ENV_ITALY Italy Envelope, 110 by 230 millimeters.

wxPAPER_ENV_MONARCH Monarch Envelope, 3 7/8 by 7 1/2 inches.

wxPAPER_ENV_PERSONAL 6 3/4 Envelope, 3 5/8 by 6 1/2 inches

wxPAPER_ESHEET E Sheet, 34 by 44 inches.

wxPAPER_EXECUTIVE Executive, 7 1/4 by 10 1/2 inches.

wxPAPER_FANFOLD_LGL_GERMAN German Legal Fanfold, 8 1/2 by 13 inches.

wxPAPER_FANFOLD_STD_GERMAN German Std Fanfold, 8 1/2 by 12 inches.

wxPAPER_FANFOLD_US US Std Fanfold, 14 7/8 by 11 inches.

wxPAPER_FOLIO Folio, 8-1/2-by-13-inch paper.

wxPAPER_ISO_B4 B4 (ISO) 250 x 353 mm.

wxPAPER_JAPANESE_POSTCARD Japanese Postcard 100 x 148 mm.

wxPAPER_JAPANESE_POSTCARD_ROTATED Japanese Postcard Rotated 148 x 100 mm.

wxPAPER_JENV_CHOU3 Japanese Envelope Chou #3.

wxPAPER_JENV_CHOU3_ROTATED Japanese Envelope Chou #3 Rotated.

wxPAPER_JENV_CHOU4 Japanese Envelope Chou #4.

wxPAPER_JENV_CHOU4_ROTATED Japanese Envelope Chou #4 Rotated.

wxPAPER_JENV_KAKU2 Japanese Envelope Kaku #2.

wxPAPER_JENV_KAKU2_ROTATED Japanese Envelope Kaku #2 Rotated.

wxPAPER_JENV_KAKU3 Japanese Envelope Kaku #3.

wxPAPER_JENV_KAKU3_ROTATED Japanese Envelope Kaku #3 Rotated.

wxPAPER_JENV_YOU4 Japanese Envelope You #4.

wxPAPER_JENV_YOU4_ROTATED Japanese Envelope You #4 Rotated.

wxPAPER_LEDGER Ledger, 17 by 11 inches.

wxPAPER_LEGAL Legal, 8 1/2 by 14 inches.

wxPAPER_LEGAL_EXTRA Legal Extra 9.5 x 15 in.

wxPAPER_LETTER Letter, 8 1/2 by 11 inches.

wxPAPER_LETTERSMALL Letter Small, 8 1/2 by 11 inches.

wxPAPER_LETTER_EXTRA Letter Extra 9.5 x 12 in.

wxPAPER_LETTER_EXTRA_TRANSVERSE Letter Extra Transverse 9.5 x 12 in.

wxPAPER_LETTER_PLUS Letter Plus 8.5 x 12.69 in.

wxPAPER_LETTER_ROTATED Letter Rotated 11 x 8 1/2 in.

wxPAPER_LETTER_TRANSVERSE Letter Transverse 8.5 x 11 in.

wxPAPER_NONE Use specific dimensions.

wxPAPER_NOTE Note, 8 1/2 by 11 inches.

wxPAPER_P16K PRC 16K 146 x 215 mm.

wxPAPER_P16K_ROTATED PRC 16K Rotated.

wxPAPER_P32K PRC 32K 97 x 151 mm.

wxPAPER_P32KBIG PRC 32K(Big) 97 x 151 mm.

wxPAPER_P32KBIG_ROTATED PRC 32K(Big) Rotated.

wxPAPER_P32K_ROTATED PRC 32K Rotated.

wxPAPER_PENV_1 PRC Envelope #1 102 x 165 mm.

wxPAPER_PENV_10 PRC Envelope #10 324 x 458 mm.

wxPAPER_PENV_10_ROTATED PRC Envelope #10 Rotated 458 x 324 m.

wxPAPER_PENV_1_ROTATED PRC Envelope #1 Rotated 165 x 102 mm.

wxPAPER_PENV_2 PRC Envelope #2 102 x 176 mm.

wxPAPER_PENV_2_ROTATED PRC Envelope #2 Rotated 176 x 102 mm.

wxPAPER_PENV_3 PRC Envelope #3 125 x 176 mm.

wxPAPER_PENV_3_ROTATED PRC Envelope #3 Rotated 176 x 125 mm.

wxPAPER_PENV_4 PRC Envelope #4 110 x 208 mm.

wxPAPER_PENV_4_ROTATED PRC Envelope #4 Rotated 208 x 110 mm.

wxPAPER_PENV_5 PRC Envelope #5 110 x 220 mm.

wxPAPER_PENV_5_ROTATED PRC Envelope #5 Rotated 220 x 110 mm.

wxPAPER_PENV_6 PRC Envelope #6 120 x 230 mm.

wxPAPER_PENV_6_ROTATED PRC Envelope #6 Rotated 230 x 120 mm.

wxPAPER_PENV_7 PRC Envelope #7 160 x 230 mm.

wxPAPER_PENV_7_ROTATED PRC Envelope #7 Rotated 230 x 160 mm.

wxPAPER_PENV_8 PRC Envelope #8 120 x 309 mm.

wxPAPER_PENV_8_ROTATED PRC Envelope #8 Rotated 309 x 120 mm.

wxPAPER_PENV_9 PRC Envelope #9 229 x 324 mm.

wxPAPER_PENV_9_ROTATED PRC Envelope #9 Rotated 324 x 229 mm.

wxPAPER_QUARTO Quarto, 215-by-275-millimeter paper.

wxPAPER_STATEMENT Statement, 5 1/2 by 8 1/2 inches.

wxPAPER_TABLOID Tabloid, 11 by 17 inches.

wxPAPER_TABLOID_EXTRA Tabloid Extra 11.69 x 18 in.

enum wxPrintMode

Print mode (currently PostScript only).

Enumerator

wxPRINT_MODE_NONE

wxPRINT_MODE_PREVIEW Preview in external application.

wxPRINT_MODE_FILE Print to file.

wxPRINT_MODE_PRINTER Send to printer.

wxPRINT_MODE_STREAM Send postscript data into a stream.

enum wxPrintOrientation

Printing orientation.

Enumerator

wxPORTRAIT

wxLANDSCAPE

enum wxSizerFlagBits

Miscellaneous flags for wxSizer items.

Enumerator

wxFIXED_MINSIZE

wxRESERVE_SPACE_EVEN_IF_HIDDEN

wxSIZER_FLAG_BITS_MASK

enum wxStandardID

Standard IDs.

Notice that some, but *not* all, of these IDs are also stock IDs, i.e. you can use them for the button or menu items without specifying the label which will be provided by the underlying platform itself. See thelist of stock items" for the subset of standard IDs which are stock IDs as well.

Enumerator

wxID_AUTO_LOWEST This id delimits the lower bound of the range used by automatically-generated ids (i.e. those used when wxID_ANY is specified during construction).

wxID_AUTO_HIGHEST This id delimits the upper bound of the range used by automatically-generated ids (i.e. those used when wxID_ANY is specified during construction).

wxID_NONE No id matches this one when compared to it.

wxID_SEPARATOR Id for a separator line in the menu (invalid for normal item).

wxID_ANY Any id: means that we don't care about the id, whether when installing an event handler or when creating a new window.

wxID_LOWEST

wxID_OPEN

wxID_CLOSE

wxID_NEW

wxID_SAVE

wxID_SAVEAS

wxID_REVERT

wxID_EXIT

wxID_UNDO

wxID_REDO

wxID_HELP

wxID_PRINT

wxID_PRINT_SETUP

wxID_PAGE_SETUP

wxID_PREVIEW

wxID_ABOUT

wxID_HELP_CONTENTS

wxID_HELP_INDEX

wxID_HELP_SEARCH

wxID_HELP_COMMANDS

wxID_HELP_PROCEDURES

wxID_HELP_CONTEXT

wxID_CLOSE_ALL

wxID_PREFERENCES

wxID_EDIT

wxID_CUT

wxID_COPY

wxID_PASTE

wxID_CLEAR

wxID_FIND

wxID_DUPLICATE

wxID_SELECTALL

wxID_DELETE

wxID_REPLACE

wxID_REPLACE_ALL

wxID_PROPERTIES

wxID_VIEW_DETAILS

wxID_VIEW_LARGEICONS

wxID_VIEW_SMALLICONS

wxID_VIEW_LIST

wxID_VIEW_SORTDATE

wxID_VIEW_SORTNAME

wxID_VIEW_SORTSIZE

wxID_VIEW_SORTTYPE

wxID_FILE

wxID_FILE1

wxID_FILE2

wxID_FILE3

wxID_FILE4

wxID_FILE5

wxID_FILE6

wxID_FILE7

wxID_FILE8

wxID_FILE9

wxID_OK Standard button and menu IDs.

wxID_CANCEL

wxID_APPLY

wxID_YES

wxID_NO

wxID_STATIC

wxID_FORWARD

wxID_BACKWARD

wxID_DEFAULT

wxID_MORE

wxID_SETUP

wxID_RESET

wxID_CONTEXT_HELP

wxID_YESTOALL

wxID_NOTOALL

wxID_ABORT

wxID_RETRY

wxID_IGNORE

wxID_ADD

wxID_REMOVE

wxID_UP

wxID_DOWN

wxID_HOME

wxID_REFRESH

wxID_STOP

wxID_INDEX

wxID_BOLD

wxID_ITALIC

wxID_JUSTIFY_CENTER

wxID_JUSTIFY_FILL

wxID_JUSTIFY_RIGHT
wxID_JUSTIFY_LEFT
wxID_UNDERLINE
wxID_INDENT
wxID_UNINDENT
wxID_ZOOM_100
wxID_ZOOM_FIT
wxID_ZOOM_IN
wxID_ZOOM_OUT
wxID_UNDELETE
wxID_REVERT_TO_SAVED
wxID_CDROM
wxID_CONVERT
wxID_EXECUTE
wxID_FLOPPY
wxID_HARDDISK
wxID_BOTTOM
wxID_FIRST
wxID_LAST
wxID_TOP
wxID_INFO
wxID_JUMP_TO
wxID_NETWORK
wxID_SELECT_COLOR
wxID_SELECT_FONT
wxID_SORT_ASCENDING
wxID_SORT_DESCENDING
wxID_SPELL_CHECK
wxID_STRIKETHROUGH
wxID_SYSTEM_MENU System menu IDs (used by wxUniv):
wxID_CLOSE_FRAME
wxID_MOVE_FRAME
wxID_RESIZE_FRAME
wxID_MAXIMIZE_FRAME
wxID_ICONIZE_FRAME
wxID_RESTORE_FRAME
wxID_MDI_WINDOW_FIRST MDI window menu ids.
wxID_MDI_WINDOW_CASCADE
wxID_MDI_WINDOW_TILE_HORZ
wxID_MDI_WINDOW_TILE_VERT
wxID_MDI_WINDOW_ARRANGE_ICONS
wxID_MDI_WINDOW_PREV
wxID_MDI_WINDOW_NEXT
wxID_MDI_WINDOW_LAST
wxID_FILEDLGG IDs used by generic file dialog (13 consecutive starting from this value)
wxID_FILECTRL IDs used by generic file ctrl (4 consecutive starting from this value)
wxID_HIGHEST

enum wxStretch

Generic stretch values.

Enumerator

wxSTRETCH_NOT

wxSHRINK

wxGROW

wxEXPAND

wxSHAPED

wxTILE

wxSTRETCH_MASK

enum wxUpdateUI

Flags which can be used in wxWindow::UpdateWindowUI().

Enumerator

wxUPDATE_UI_NONE

wxUPDATE_UI_RECURSE

wxUPDATE_UI_FROMIDLE Invoked from On(Internal)Idle.

22.250.4 Variable Documentation

wxCoord wxDefaultCoord = -1

A special value meaning "use default coordinate".

22.251 interface/wx/dialup.h File Reference

Classes

- class wxDialUpManager

  This class encapsulates functions dealing with verifying the connection status of the workstation (connected to the Internet via a direct connection, connected through a modem or not connected at all) and to establish this connection if possible/required (i.e.
- class wxDialUpEvent

  This is the event class for the dialup events sent by wxDialUpManager.

22.252 interface/wx/dir.h File Reference

Classes

- class wxDirTraverser

  wxDirTraverser is an abstract interface which must be implemented by objects passed to wxDir::Traverse() function.
- class wxDir

  wxDir is a portable equivalent of Unix open/read/closedir functions which allow enumerating of the files in a directory.

Enumerations

- enum wxDirTraverseResult {
  wxDIR_IGNORE = -1,
  wxDIR_STOP,
  wxDIR_CONTINUE }

  Possible return values of wxDirTraverser callback functions.
- enum wxDirFlags {
  wxDIR_FILES = 0x0001,
  wxDIR_DIRS = 0x0002,
  wxDIR_HIDDEN = 0x0004,
  wxDIR_DOTDOT = 0x0008,
  wxDIR_NO_FOLLOW = 0x0010,
  wxDIR_DEFAULT = wxDIR_FILES | wxDIR_DIRS | wxDIR_HIDDEN }

  These flags affect the behaviour of GetFirst/GetNext() and Traverse(), determining what types are included in the list of items they produce.

22.252.1 Enumeration Type Documentation

enum wxDirFlags

These flags affect the behaviour of GetFirst/GetNext() and Traverse(), determining what types are included in the list of items they produce.

Enumerator

wxDIR_FILES Includes files.

wxDIR_DIRS Includes directories.

wxDIR_HIDDEN Includes hidden files.

wxDIR_DOTDOT Includes "." and "..".

wxDIR_NO_FOLLOW Don't follow symbolic links during the directory traversal. This flag is ignored under systems not supporting symbolic links (i.e. non-Unix ones).

Notice that this flag is *not* included in wxDIR_DEFAULT and so the default behaviour of wxDir::Traverse() is to follow symbolic links, even if they lead outside of the directory being traversed.

Since

2.9.5

wxDIR_DEFAULT Default directory traversal flags include both files and directories, even hidden. Notice that by default wxDIR_NO_FOLLOW is *not* included, meaning that symbolic links are followed by default. If this is not desired, you must pass that flag explicitly.

enum wxDirTraverseResult

Possible return values of wxDirTraverser callback functions.

Enumerator

wxDIR_IGNORE Ignore this directory but continue with others.

wxDIR_STOP Stop traversing.

wxDIR_CONTINUE Continue into this directory.

22.253 interface/wx/dirctrl.h File Reference

Classes

- class wxGenericDirCtrl

  This control can be used to place a directory listing (with optional files) on an arbitrary window.
- class wxDirFilterListCtrl

Enumerations

- enum {
  wxDIRCTRL_DIR_ONLY = 0x0010,
  wxDIRCTRL_SELECT_FIRST = 0x0020,
  wxDIRCTRL_SHOW_FILTERS = 0x0040,
  wxDIRCTRL_3D_INTERNAL = 0x0080,
  wxDIRCTRL_EDIT_LABELS = 0x0100,
  wxDIRCTRL_MULTIPLE = 0x0200,
  wxDIRCTRL_DEFAULT_STYLE = wxDIRCTRL_3D_INTERNAL }

Variables

- wxEventType wxEVT_DIRCTRL_SELECTIONCHANGED
- wxEventType wxEVT_DIRCTRL_FILEACTIVATED

22.253.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxDIRCTRL_DIR_ONLY

wxDIRCTRL_SELECT_FIRST

wxDIRCTRL_SHOW_FILTERS

wxDIRCTRL_3D_INTERNAL

wxDIRCTRL_EDIT_LABELS

wxDIRCTRL_MULTIPLE

wxDIRCTRL_DEFAULT_STYLE

22.253.2 Variable Documentation

wxEventType wxEVT_DIRCTRL_FILEACTIVATED

wxEventType wxEVT_DIRCTRL_SELECTIONCHANGED

22.254 interface/wx/dirdlg.h File Reference

Classes

- class wxDirDialog

  This class represents the directory chooser dialog.

Macros

- #define wxDD_CHANGE_DIR 0x0100
- #define wxDD_DIR_MUST_EXIST 0x0200
- #define wxDD_NEW_DIR_BUTTON 0
- #define wxDD_DEFAULT_STYLE (wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER)

Functions

- wxString wxDirSelector (const wxString &message=wxDirSelectorPromptStr, const wxString &default_↩ path=wxEmptyString, long style=0, const wxPoint &pos=wxDefaultPosition, wxWindow *parent=NULL)

  Pops up a directory selector dialog.

Variables

- const char wxDirDialogDefaultFolderStr [] = "/"

  Initial folder for generic directory dialog.
- const char wxDirSelectorPromptStr [] = "Select a directory"

  Default message for directory selector dialog.
- const char wxDirDialogNameStr [] = "wxDirCtrl"

  Default name for directory selector dialog.

22.254.1 Macro Definition Documentation

#define wxDD_CHANGE_DIR 0x0100

#define wxDD_DEFAULT_STYLE (wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER)

#define wxDD_DIR_MUST_EXIST 0x0200

#define wxDD_NEW_DIR_BUTTON 0

22.254.2 Variable Documentation

const char wxDirDialogDefaultFolderStr[ ] = "/"

Initial folder for generic directory dialog.

const char wxDirDialogNameStr[ ] = "wxDirCtrl"

Default name for directory selector dialog.

const char wxDirSelectorPromptStr[ ] = "Select a directory"

Default message for directory selector dialog.

22.255 interface/wx/display.h File Reference

Classes

- class wxDisplay

  Determines the sizes and locations of displays connected to the system.

22.256 interface/wx/docmdi.h File Reference

Classes

- class wxDocMDIParentFrame

  The wxDocMDIParentFrame class provides a default top-level frame for applications using the document/view framework.
- class wxDocMDIChildFrame

  The wxDocMDIChildFrame class provides a default frame for displaying documents on separate windows.

22.257 interface/wx/dragimag.h File Reference

Classes

- class wxDragImage

  This class is used when you wish to drag an object on the screen, and a simple cursor is not enough.

22.258 interface/wx/dynarray.h File Reference

Classes

- class wxArray< T >

  This section describes the so called "dynamic arrays".

Macros

- #define WX_APPEND_ARRAY(wxArray_arrayToModify, wxArray_arrayToBeAppended)

  This macro may be used to append all elements of the wxArray_arrayToBeAppended array to the wxArray_arrayTo↩Modify.
- #define WX_CLEAR_ARRAY(wxArray_arrayToBeCleared)

  This macro may be used to delete all elements of the array before emptying it.
- #define WX_PREPEND_ARRAY(wxArray_arrayToModify, wxArray_arrayToBePrepended)

  This macro may be used to prepend all elements of the wxArray_arrayToBePrepended array to the wxArray_array↩ToModify.

- #define WX_DECLARE_OBJARRAY(T, name)

  This macro declares a new object array class named name and containing the elements of type T.
- #define WX_DECLARE_EXPORTED_OBJARRAY(T, name)

  This macro declares a new object array class named name and containing the elements of type T.
- #define WX_DECLARE_USER_EXPORTED_OBJARRAY(T, name)

  This macro declares a new object array class named name and containing the elements of type T.

- #define WX_DEFINE_ARRAY(T, name)

  This macro defines a new array class named name and containing the elements of type T.
- #define WX_DEFINE_EXPORTED_ARRAY(T, name)

  This macro defines a new array class named name and containing the elements of type T.
- #define WX_DEFINE_USER_EXPORTED_ARRAY(T, name, exportspec)

  This macro defines a new array class named name and containing the elements of type T.

- #define WX_DEFINE_OBJARRAY(name)

This macro defines the methods of the array class name not defined by the WX_DECLARE_OBJARRAY() macro.

- #define WX_DEFINE_EXPORTED_OBJARRAY(name)

  This macro defines the methods of the array class name not defined by the WX_DECLARE_OBJARRAY() macro.
- #define WX_DEFINE_USER_EXPORTED_OBJARRAY(name)

  This macro defines the methods of the array class name not defined by the WX_DECLARE_OBJARRAY() macro.

- #define WX_DEFINE_SORTED_ARRAY(T, name)

  This macro defines a new sorted array class named name and containing the elements of type T.
- #define WX_DEFINE_SORTED_EXPORTED_ARRAY(T, name)

  This macro defines a new sorted array class named name and containing the elements of type T.
- #define WX_DEFINE_SORTED_USER_EXPORTED_ARRAY(T, name)

  This macro defines a new sorted array class named name and containing the elements of type T.

Typedefs

- typedef wxArray< int > wxArrayInt

  Predefined specialization of wxArray<T> for standard types.
- typedef wxArray< long > wxArrayLong

  Predefined specialization of wxArray<T> for standard types.
- typedef wxArray< short > wxArrayShort

  Predefined specialization of wxArray<T> for standard types.
- typedef wxArray< double > wxArrayDouble

  Predefined specialization of wxArray<T> for standard types.
- typedef wxArray< void * > wxArrayPtrVoid

  Predefined specialization of wxArray<T> for standard types.

22.258.1 Macro Definition Documentation

#define WX_APPEND_ARRAY(*wxArray_arrayToModify, wxArray_arrayToBeAppended*)

This macro may be used to append all elements of the *wxArray_arrayToBeAppended* array to the *wxArray_array↩ ToModify*.

The two arrays must be of the same type.

#define WX_CLEAR_ARRAY(*wxArray_arrayToBeCleared*)

This macro may be used to delete all elements of the array before emptying it.

It cannot be used with wxObjArrays - but they will delete their elements anyway when you call Empty().

#define WX_DECLARE_EXPORTED_OBJARRAY(*T, name*)

This macro declares a new object array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 class MyClass;
2 WX_DECLARE_OBJARRAY(MyClass, wxArrayOfMyClass); // note: not "MyClass *"!
```

You must use WX_DEFINE_OBJARRAY() macro to define the array class, otherwise you would get link errors.

#define WX_DECLARE_OBJARRAY(*T, name*)

This macro declares a new object array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 class MyClass;
2 WX_DECLARE_OBJARRAY(MyClass, wxArrayOfMyClass); // note: not "MyClass *"!
```

You must use WX_DEFINE_OBJARRAY() macro to define the array class, otherwise you would get link errors.

#define WX_DECLARE_USER_EXPORTED_OBJARRAY(*T, name*)

This macro declares a new object array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 class MyClass;
2 WX_DECLARE_OBJARRAY(MyClass, wxArrayOfMyClass); // note: not "MyClass *"!
```

You must use WX_DEFINE_OBJARRAY() macro to define the array class, otherwise you would get link errors.

#define WX_DEFINE_ARRAY(*T, name*)

This macro defines a new array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 WX_DEFINE_ARRAY_INT(int, MyArrayInt);
2
3 class MyClass;
4 WX_DEFINE_ARRAY(MyClass *, ArrayOfMyClass);
```

Note that wxWidgets predefines the following standard array classes: **wxArrayInt**, **wxArrayLong**, **wxArrayShort**, **wxArrayDouble**, **wxArrayPtrVoid**.

#define WX_DEFINE_EXPORTED_ARRAY(*T, name*)

This macro defines a new array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 WX_DEFINE_ARRAY_INT(int, MyArrayInt);
2
3 class MyClass;
4 WX_DEFINE_ARRAY(MyClass *, ArrayOfMyClass);
```

Note that wxWidgets predefines the following standard array classes: **wxArrayInt**, **wxArrayLong**, **wxArrayShort**, **wxArrayDouble**, **wxArrayPtrVoid**.

#define WX_DEFINE_EXPORTED_OBJARRAY(*name*)

This macro defines the methods of the array class *name* not defined by the WX_DECLARE_OBJARRAY() macro.

You must include the file <wx/arrimpl.cpp> before using this macro and you must have the full declaration of the class of array elements in scope! If you forget to do the first, the error will be caught by the compiler, but, unfortunately, many compilers will not give any warnings if you forget to do the second - but the objects of the class will not be copied correctly and their real destructor will not be called.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example of usage:

```
1 // first declare the class!
2 class MyClass
3 {
4 public:
5     MyClass(const MyClass&);
6
7     // ...
8
9     virtual ~MyClass();
10 };
11
12 #include <wx/arrimpl.cpp>
13 WX_DEFINE_OBJARRAY(wxArrayOfMyClass);
```

#define WX_DEFINE_OBJARRAY(*name*)

This macro defines the methods of the array class *name* not defined by the WX_DECLARE_OBJARRAY() macro.

You must include the file <wx/arrimpl.cpp> before using this macro and you must have the full declaration of the class of array elements in scope! If you forget to do the first, the error will be caught by the compiler, but, unfortunately, many compilers will not give any warnings if you forget to do the second - but the objects of the class will not be copied correctly and their real destructor will not be called.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example of usage:

```
1 // first declare the class!
2 class MyClass
3 {
4 public:
5     MyClass(const MyClass&);
6
7     // ...
8
9     virtual ~MyClass();
10 };
11
12 #include <wx/arrimpl.cpp>
13 WX_DEFINE_OBJARRAY(wxArrayOfMyClass);
```

#define WX_DEFINE_SORTED_ARRAY(*T*, *name*)

This macro defines a new sorted array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 WX_DEFINE_SORTED_ARRAY_INT(int, MySortedArrayInt);
2
3 class MyClass;
4 WX_DEFINE_SORTED_ARRAY(MyClass *, ArrayOfMyClass);
```

You will have to initialize the objects of this class by passing a comparison function to the array object constructor like this:

```
1 int CompareInts(int n1, int n2)
2 {
3     return n1 - n2;
4 }
5
6 MySortedArrayInt sorted(CompareInts);
7
8 int CompareMyClassObjects(MyClass *item1, MyClass *item2)
9 {
10     // sort the items by their address...
11     return Stricmp(item1->GetAddress(), item2->GetAddress());
12 }
13
14 ArrayOfMyClass another(CompareMyClassObjects);
```

#define WX_DEFINE_SORTED_EXPORTED_ARRAY(*T, name*)

This macro defines a new sorted array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 WX_DEFINE_SORTED_ARRAY_INT(int, MySortedArrayInt);
2
3 class MyClass;
4 WX_DEFINE_SORTED_ARRAY(MyClass *, ArrayOfMyClass);
```

You will have to initialize the objects of this class by passing a comparison function to the array object constructor like this:

```
1 int CompareInts(int n1, int n2)
2 {
3     return n1 - n2;
4 }
5
6 MySortedArrayInt sorted(CompareInts);
7
8 int CompareMyClassObjects(MyClass *item1, MyClass *item2)
9 {
10     // sort the items by their address...
11     return Stricmp(item1->GetAddress(), item2->GetAddress());
12 }
13
14 ArrayOfMyClass another(CompareMyClassObjects);
```

#define WX_DEFINE_SORTED_USER_EXPORTED_ARRAY(*T, name*)

This macro defines a new sorted array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 WX_DEFINE_SORTED_ARRAY_INT(int, MySortedArrayInt);
2
3 class MyClass;
4 WX_DEFINE_SORTED_ARRAY(MyClass *, ArrayOfMyClass);
```

You will have to initialize the objects of this class by passing a comparison function to the array object constructor like this:

```
1 int CompareInts(int n1, int n2)
2 {
3     return n1 - n2;
4 }
5
6 MySortedArrayInt sorted(CompareInts);
7
8 int CompareMyClassObjects(MyClass *item1, MyClass *item2)
9 {
10     // sort the items by their address...
11     return Stricmp(item1->GetAddress(), item2->GetAddress());
12 }
13
14 ArrayOfMyClass another(CompareMyClassObjects);
```

#define WX_DEFINE_USER_EXPORTED_ARRAY(*T, name, exportspec*)

This macro defines a new array class named *name* and containing the elements of type *T*.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example:

```
1 WX_DEFINE_ARRAY_INT(int, MyArrayInt);
2
3 class MyClass;
4 WX_DEFINE_ARRAY(MyClass *, ArrayOfMyClass);
```

Note that wxWidgets predefines the following standard array classes: **wxArrayInt**, **wxArrayLong**, **wxArrayShort**, **wxArrayDouble**, **wxArrayPtrVoid**.

#define WX_DEFINE_USER_EXPORTED_OBJARRAY(*name*)

This macro defines the methods of the array class *name* not defined by the WX_DECLARE_OBJARRAY() macro.

You must include the file <wx/arrimpl.cpp> before using this macro and you must have the full declaration of the class of array elements in scope! If you forget to do the first, the error will be caught by the compiler, but, unfortunately, many compilers will not give any warnings if you forget to do the second - but the objects of the class will not be copied correctly and their real destructor will not be called.

An exported array is used when compiling wxWidgets as a DLL under Windows and the array needs to be visible outside the DLL. An user exported array needed for exporting an array from a user DLL.

Example of usage:

```
1 // first declare the class!
2 class MyClass
3 {
4 public:
5     MyClass(const MyClass&);
6
7     // ...
8
9     virtual ~MyClass();
10 };
11
12 #include <wx/arrimpl.cpp>
13 WX_DEFINE_OBJARRAY(wxArrayOfMyClass);
```

#define WX_PREPEND_ARRAY(*wxArray_arrayToModify, wxArray_arrayToBePrepended*)

This macro may be used to prepend all elements of the *wxArray_arrayToBePrepended* array to the *wxArray_array↩ ToModify*.

The two arrays must be of the same type.

22.258.2 Typedef Documentation

typedef wxArray<double> wxArrayDouble

Predefined specialization of wxArray<T> for standard types.

typedef wxArray<int> wxArrayInt

Predefined specialization of wxArray<T> for standard types.

typedef wxArray<long> wxArrayLong

Predefined specialization of wxArray<T> for standard types.

typedef wxArray<void*> wxArrayPtrVoid

Predefined specialization of wxArray<T> for standard types.

typedef wxArray<short> wxArrayShort

Predefined specialization of wxArray<T> for standard types.

22.259 interface/wx/dynlib.h File Reference

Classes

- class wxDynamicLibraryDetails

  This class is used for the objects returned by the wxDynamicLibrary::ListLoaded() method and contains the information about a single module loaded into the address space of the current process.
- class wxDynamicLibrary

  wxDynamicLibrary is a class representing dynamically loadable library (Windows DLL, shared library under Unix etc).

Macros

- #define wxDYNLIB_FUNCTION(type, name, dynlib)

  When loading a function from a DLL you always have to cast the returned `void *` pointer to the correct type and, even more annoyingly, you have to repeat this type twice if you want to declare and define a function pointer all in one line.

Enumerations

- enum wxDynamicLibraryCategory {
  wxDL_LIBRARY,
  wxDL_MODULE }

  Dynamic library category used with wxDynamicLibrary::CanonicalizeName().
- enum wxPluginCategory {
  wxDL_PLUGIN_GUI,
  wxDL_PLUGIN_BASE }

  Dynamic library plugin category used with wxDynamicLibrary::CanonicalizePluginName().

22.259.1 Enumeration Type Documentation

enum wxDynamicLibraryCategory

Dynamic library category used with wxDynamicLibrary::CanonicalizeName().

Enumerator

wxDL_LIBRARY Standard library.
wxDL_MODULE Loadable module/plugin.

enum wxPluginCategory

Dynamic library plugin category used with wxDynamicLibrary::CanonicalizePluginName().

Enumerator

wxDL_PLUGIN_GUI Plugin that uses GUI classes.
wxDL_PLUGIN_BASE wxBase-only plugin.

22.260 interface/wx/editlbox.h File Reference

Classes

- class wxEditableListBox
  An editable listbox is composite control that lets the user easily enter, delete and reorder a list of strings.

Macros

- #define wxEL_ALLOW_NEW 0x0100
- #define wxEL_ALLOW_EDIT 0x0200
- #define wxEL_ALLOW_DELETE 0x0400
- #define wxEL_NO_REORDER 0x0800
- #define wxEL_DEFAULT_STYLE (wxEL_ALLOW_NEW | wxEL_ALLOW_EDIT | wxEL_ALLOW_DELETE)

22.260.1 Macro Definition Documentation

#define wxEL_ALLOW_DELETE 0x0400

#define wxEL_ALLOW_EDIT 0x0200

#define wxEL_ALLOW_NEW 0x0100

#define wxEL_DEFAULT_STYLE (wxEL_ALLOW_NEW | wxEL_ALLOW_EDIT | wxEL_ALLOW_DELETE)

#define wxEL_NO_REORDER 0x0800

22.261 interface/wx/encconv.h File Reference

Classes

- class wxEncodingConverter
  This class is capable of converting strings between two 8-bit encodings/charsets.

22.262 interface/wx/event.h File Reference

Classes

- class wxEvent

  An event is a structure holding information about an event passed to a callback or member function.
- class wxEventBlocker

  This class is a special event handler which allows to discard any event (or a set of event types) directed to a specific window.
- class wxPropagationDisabler

  Helper class to temporarily change an event to not propagate.
- class wxPropagateOnce

  Helper class to temporarily lower propagation level.
- class wxEvtHandler

  A class that can handle events from the windowing system.
- class wxKeyEvent

  This event class contains information about key press and release events.
- class wxJoystickEvent

  This event class contains information about joystick events, particularly events received by windows.
- class wxScrollWinEvent

  A scroll event holds information about events sent from scrolling windows.
- class wxSysColourChangedEvent

  This class is used for system colour change events, which are generated when the user changes the colour settings using the control panel.
- class wxCommandEvent

  This event class contains information about command events, which originate from a variety of simple controls.
- class wxWindowCreateEvent

  This event is sent just after the actual window associated with a wxWindow object has been created.
- class wxPaintEvent

  A paint event is sent when a window's contents needs to be repainted.
- class wxMaximizeEvent

  An event being sent when a top level window is maximized.
- class wxUpdateUIEvent

  This class is used for pseudo-events which are called by wxWidgets to give an application the chance to update various user interface elements.
- class wxClipboardTextEvent

  This class represents the events generated by a control (typically a wxTextCtrl but other windows can generate these events as well) when its content gets copied or cut to, or pasted from the clipboard.
- class wxMouseEvent

  This event class contains information about the events generated by the mouse: they include mouse buttons press and release events and mouse move events.
- class wxDropFilesEvent

  This class is used for drop files events, that is, when files have been dropped onto the window.
- class wxActivateEvent

  An activate event is sent when a window or application is being activated or deactivated.
- class wxContextMenuEvent

  This class is used for context menu events, sent to give the application a chance to show a context (popup) menu for a wxWindow.
- class wxEraseEvent

  An erase event is sent when a window's background needs to be repainted.
- class wxFocusEvent

  A focus event is sent when a window's focus changes.

- class wxChildFocusEvent

  A child focus event is sent to a (parent-)window when one of its child windows gains focus, so that the window could restore the focus back to its corresponding child if it loses it now and regains later.

- class wxMouseCaptureLostEvent

  A mouse capture lost event is sent to a window that had obtained mouse capture, which was subsequently lost due to an "external" event (for example, when a dialog box is shown or if another application captures the mouse).

- class wxDisplayChangedEvent
- class wxPaletteChangedEvent
- class wxQueryNewPaletteEvent
- class wxNotifyEvent

  This class is not used by the event handlers by itself, but is a base class for other event classes (such as wxBookCtrlEvent).

- class wxThreadEvent

  This class adds some simple functionality to wxEvent to facilitate inter-thread communication.

- class wxHelpEvent

  A help event is sent when the user has requested context-sensitive help.

- class wxScrollEvent

  A scroll event holds information about events sent from stand-alone scrollbars (see wxScrollBar) and sliders (see wxSlider).

- class wxIdleEvent

  This class is used for idle events, which are generated when the system becomes idle.

- class wxInitDialogEvent

  A wxInitDialogEvent is sent as a dialog or panel is being initialised.

- class wxWindowDestroyEvent

  This event is sent as early as possible during the window destruction process.

- class wxNavigationKeyEvent

  This event class contains information about navigation events, generated by navigation keys such as tab and page down.

- class wxMouseCaptureChangedEvent

  An mouse capture changed event is sent to a window that loses its mouse capture.

- class wxCloseEvent

  This event class contains information about window and session close events.

- class wxMenuEvent

  This class is used for a variety of menu-related events.

- class wxShowEvent

  An event being sent when the window is shown or hidden.

- class wxIconizeEvent

  An event being sent when the frame is iconized (minimized) or restored.

- class wxMoveEvent

  A move event holds information about wxTopLevelWindow move change events.

- class wxSizeEvent

  A size event holds information about size change events of wxWindow.

- class wxSetCursorEvent

  A wxSetCursorEvent is generated from wxWindow when the mouse cursor is about to be set as a result of mouse motion.

Macros

- #define wxDEFINE_EVENT(name, cls) const wxEventTypeTag< cls > name(wxNewEventType())

  Define a new event type associated with the specified event class.
- #define wxDECLARE_EVENT(name, cls) wxDECLARE_EXPORTED_EVENT(wxEMPTY_PARAMETER_↩ VALUE, name, cls)

  Declares a custom event type.
- #define wxDECLARE_EXPORTED_EVENT(expdecl, name, cls) extern const expdecl wxEventTypeTag< cls > name;

  Variant of wxDECLARE_EVENT() used for event types defined inside a shared library.
- #define wxEVENT_HANDLER_CAST(functype, func) (&func)

  Helper macro for definition of custom event table macros.
- #define wx__DECLARE_EVT1(evt, id, fn) wx__DECLARE_EVT2(evt, id, wxID_ANY, fn)

  This macro is used to define event table macros for handling custom events.
- #define wx__DECLARE_EVT2(evt, id1, id2, fn) DECLARE_EVENT_TABLE_ENTRY(evt, id1, id2, fn, NULL),

  Generalized version of the wx__DECLARE_EVT1() macro taking a range of IDs instead of a single one.
- #define wx__DECLARE_EVT0(evt, fn) wx__DECLARE_EVT1(evt, wxID_ANY, fn)

  Simplified version of the wx__DECLARE_EVT1() macro, to be used when the event type must be handled regardless of the ID associated with the specific event instances.
- #define wxDECLARE_EVENT_TABLE()

  Use this macro inside a class declaration to declare a static event table for that class.
- #define wxBEGIN_EVENT_TABLE(theClass, baseClass)

  Use this macro in a source file to start listing static event handlers for a specific class.
- #define wxEND_EVENT_TABLE()

  Use this macro in a source file to end listing static event handlers for a specific class.

Typedefs

- typedef int wxEventType

  A value uniquely identifying the type of the event.

Enumerations

- enum wxEventPropagation {
  wxEVENT_PROPAGATE_NONE = 0,
  wxEVENT_PROPAGATE_MAX = INT_MAX }

  The predefined constants for the number of times we propagate event upwards window child-parent chain.
- enum wxEventCategory {
  wxEVT_CATEGORY_UI = 1,
  wxEVT_CATEGORY_USER_INPUT = 2,
  wxEVT_CATEGORY_SOCKET = 4,
  wxEVT_CATEGORY_TIMER = 8,
  wxEVT_CATEGORY_THREAD = 16,
  wxEVT_CATEGORY_ALL }

  The different categories for a wxEvent; see wxEvent::GetEventCategory.
- enum wxKeyCategoryFlags {
  WXK_CATEGORY_ARROW,
  WXK_CATEGORY_PAGING,
  WXK_CATEGORY_JUMP,
  WXK_CATEGORY_TAB,
  WXK_CATEGORY_CUT,
  WXK_CATEGORY_NAVIGATION }

  Flags for categories of keys.

- enum {
  wxJOYSTICK1,
  wxJOYSTICK2 }
- enum {
  wxJOY_BUTTON_ANY = -1,
  wxJOY_BUTTON1 = 1,
  wxJOY_BUTTON2 = 2,
  wxJOY_BUTTON3 = 4,
  wxJOY_BUTTON4 = 8 }
- enum wxUpdateUIMode {
  wxUPDATE_UI_PROCESS_ALL,
  wxUPDATE_UI_PROCESS_SPECIFIED }

  The possibles modes to pass to wxUpdateUIEvent::SetMode().
- enum wxMouseWheelAxis {
  wxMOUSE_WHEEL_VERTICAL,
  wxMOUSE_WHEEL_HORIZONTAL }

  Possible axis values for mouse wheel scroll events.
- enum wxIdleMode {
  wxIDLE_PROCESS_ALL,
  wxIDLE_PROCESS_SPECIFIED }

  See wxIdleEvent::SetMode() for more info.

Functions

- wxEventType wxNewEventType ()

  Generates a new unique event type.
- void wxPostEvent (wxEvtHandler ∗dest, const wxEvent &event)

  In a GUI application, this function posts event to the specified dest object using wxEvtHandler::AddPendingEvent().
- void wxQueueEvent (wxEvtHandler ∗dest, wxEvent ∗event)

  Queue an event for processing on the given object.

Variables

- wxEventType wxEVT_NULL

  A special event type usually used to indicate that some wxEvent has yet no type assigned.
- wxEventType wxEVT_ANY
- wxEventType wxEVT_BUTTON
- wxEventType wxEVT_CHECKBOX
- wxEventType wxEVT_CHOICE
- wxEventType wxEVT_LISTBOX
- wxEventType wxEVT_LISTBOX_DCLICK
- wxEventType wxEVT_CHECKLISTBOX
- wxEventType wxEVT_MENU
- wxEventType wxEVT_SLIDER
- wxEventType wxEVT_RADIOBOX
- wxEventType wxEVT_RADIOBUTTON
- wxEventType wxEVT_SCROLLBAR
- wxEventType wxEVT_VLBOX
- wxEventType wxEVT_COMBOBOX
- wxEventType wxEVT_TOOL_RCLICKED
- wxEventType wxEVT_TOOL_DROPDOWN
- wxEventType wxEVT_TOOL_ENTER
- wxEventType wxEVT_COMBOBOX_DROPDOWN

- wxEventType wxEVT_COMBOBOX_CLOSEUP
- wxEventType wxEVT_THREAD
- wxEventType wxEVT_LEFT_DOWN
- wxEventType wxEVT_LEFT_UP
- wxEventType wxEVT_MIDDLE_DOWN
- wxEventType wxEVT_MIDDLE_UP
- wxEventType wxEVT_RIGHT_DOWN
- wxEventType wxEVT_RIGHT_UP
- wxEventType wxEVT_MOTION
- wxEventType wxEVT_ENTER_WINDOW
- wxEventType wxEVT_LEAVE_WINDOW
- wxEventType wxEVT_LEFT_DCLICK
- wxEventType wxEVT_MIDDLE_DCLICK
- wxEventType wxEVT_RIGHT_DCLICK
- wxEventType wxEVT_SET_FOCUS
- wxEventType wxEVT_KILL_FOCUS
- wxEventType wxEVT_CHILD_FOCUS
- wxEventType wxEVT_MOUSEWHEEL
- wxEventType wxEVT_AUX1_DOWN
- wxEventType wxEVT_AUX1_UP
- wxEventType wxEVT_AUX1_DCLICK
- wxEventType wxEVT_AUX2_DOWN
- wxEventType wxEVT_AUX2_UP
- wxEventType wxEVT_AUX2_DCLICK
- wxEventType wxEVT_CHAR
- wxEventType wxEVT_CHAR_HOOK
- wxEventType wxEVT_NAVIGATION_KEY
- wxEventType wxEVT_KEY_DOWN
- wxEventType wxEVT_KEY_UP
- wxEventType wxEVT_HOTKEY
- wxEventType wxEVT_SET_CURSOR
- wxEventType wxEVT_SCROLL_TOP
- wxEventType wxEVT_SCROLL_BOTTOM
- wxEventType wxEVT_SCROLL_LINEUP
- wxEventType wxEVT_SCROLL_LINEDOWN
- wxEventType wxEVT_SCROLL_PAGEUP
- wxEventType wxEVT_SCROLL_PAGEDOWN
- wxEventType wxEVT_SCROLL_THUMBTRACK
- wxEventType wxEVT_SCROLL_THUMBRELEASE
- wxEventType wxEVT_SCROLL_CHANGED
- wxEventType wxEVT_SPIN_UP
- wxEventType wxEVT_SPIN_DOWN
- wxEventType wxEVT_SPIN
- wxEventType wxEVT_SCROLLWIN_TOP
- wxEventType wxEVT_SCROLLWIN_BOTTOM
- wxEventType wxEVT_SCROLLWIN_LINEUP
- wxEventType wxEVT_SCROLLWIN_LINEDOWN
- wxEventType wxEVT_SCROLLWIN_PAGEUP
- wxEventType wxEVT_SCROLLWIN_PAGEDOWN
- wxEventType wxEVT_SCROLLWIN_THUMBTRACK
- wxEventType wxEVT_SCROLLWIN_THUMBRELEASE
- wxEventType wxEVT_SIZE
- wxEventType wxEVT_MOVE
- wxEventType wxEVT_CLOSE_WINDOW
- wxEventType wxEVT_END_SESSION

- wxEventType wxEVT_QUERY_END_SESSION
- wxEventType wxEVT_ACTIVATE_APP
- wxEventType wxEVT_ACTIVATE
- wxEventType wxEVT_CREATE
- wxEventType wxEVT_DESTROY
- wxEventType wxEVT_SHOW
- wxEventType wxEVT_ICONIZE
- wxEventType wxEVT_MAXIMIZE
- wxEventType wxEVT_MOUSE_CAPTURE_CHANGED
- wxEventType wxEVT_MOUSE_CAPTURE_LOST
- wxEventType wxEVT_PAINT
- wxEventType wxEVT_ERASE_BACKGROUND
- wxEventType wxEVT_NC_PAINT
- wxEventType wxEVT_MENU_OPEN
- wxEventType wxEVT_MENU_CLOSE
- wxEventType wxEVT_MENU_HIGHLIGHT
- wxEventType wxEVT_CONTEXT_MENU
- wxEventType wxEVT_SYS_COLOUR_CHANGED
- wxEventType wxEVT_DISPLAY_CHANGED
- wxEventType wxEVT_QUERY_NEW_PALETTE
- wxEventType wxEVT_PALETTE_CHANGED
- wxEventType wxEVT_JOY_BUTTON_DOWN
- wxEventType wxEVT_JOY_BUTTON_UP
- wxEventType wxEVT_JOY_MOVE
- wxEventType wxEVT_JOY_ZMOVE
- wxEventType wxEVT_DROP_FILES
- wxEventType wxEVT_INIT_DIALOG
- wxEventType wxEVT_IDLE
- wxEventType wxEVT_UPDATE_UI
- wxEventType wxEVT_SIZING
- wxEventType wxEVT_MOVING
- wxEventType wxEVT_MOVE_START
- wxEventType wxEVT_MOVE_END
- wxEventType wxEVT_HIBERNATE
- wxEventType wxEVT_TEXT_COPY
- wxEventType wxEVT_TEXT_CUT
- wxEventType wxEVT_TEXT_PASTE
- wxEventType wxEVT_COMMAND_LEFT_CLICK
- wxEventType wxEVT_COMMAND_LEFT_DCLICK
- wxEventType wxEVT_COMMAND_RIGHT_CLICK
- wxEventType wxEVT_COMMAND_RIGHT_DCLICK
- wxEventType wxEVT_COMMAND_SET_FOCUS
- wxEventType wxEVT_COMMAND_KILL_FOCUS
- wxEventType wxEVT_COMMAND_ENTER
- wxEventType wxEVT_HELP
- wxEventType wxEVT_DETAILED_HELP
- wxEventType wxEVT_TOOL
- wxEventType wxEVT_WINDOW_MODAL_DIALOG_CLOSED

22.262.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxJOYSTICK1

wxJOYSTICK2

anonymous enum

Enumerator

wxJOY_BUTTON_ANY

wxJOY_BUTTON1

wxJOY_BUTTON2

wxJOY_BUTTON3

wxJOY_BUTTON4

enum **wxEventCategory**

The different categories for a wxEvent; see wxEvent::GetEventCategory.

Note

They are used as OR-combinable flags by wxEventLoopBase::YieldFor.

Enumerator

wxEVT_CATEGORY_UI This is the category for those events which are generated to update the appearance of the GUI but which (usually) do not comport data processing, i.e. which do not provide input or output data (e.g. size events, scroll events, etc). They are events NOT directly generated by the user's input devices.

wxEVT_CATEGORY_USER_INPUT This category groups those events which are generated directly from the user through input devices like mouse and keyboard and usually result in data to be processed from the application (e.g. mouse clicks, key presses, etc).

wxEVT_CATEGORY_SOCKET This category is for wxSocketEvent.

wxEVT_CATEGORY_TIMER This category is for wxTimerEvent.

wxEVT_CATEGORY_THREAD This category is for any event used to send notifications from the secondary threads to the main one or in general for notifications among different threads (which may or may not be user-generated). See e.g. wxThreadEvent.

wxEVT_CATEGORY_ALL This mask is used in wxEventLoopBase::YieldFor to specify that all event categories should be processed.

enum **wxEventPropagation**

The predefined constants for the number of times we propagate event upwards window child-parent chain.

Enumerator

wxEVENT_PROPAGATE_NONE don't propagate it at all

wxEVENT_PROPAGATE_MAX propagate it until it is processed

enum **wxIdleMode**

See wxIdleEvent::SetMode() for more info.

Enumerator

wxIDLE_PROCESS_ALL Send idle events to all windows.

wxIDLE_PROCESS_SPECIFIED Send idle events to windows that have the wxWS_EX_PROCESS_IDLE flag specified.

enum wxKeyCategoryFlags

Flags for categories of keys.

These values are used by wxKeyEvent::IsKeyInCategory(). They may be combined via the bitwise operators |, &, and ~.

Since

2.9.1

Enumerator

WXK_CATEGORY_ARROW arrow keys, on and off numeric keypads

WXK_CATEGORY_PAGING page up and page down keys, on and off numeric keypads

WXK_CATEGORY_JUMP home and end keys, on and off numeric keypads

WXK_CATEGORY_TAB tab key, on and off numeric keypads

WXK_CATEGORY_CUT backspace and delete keys, on and off numeric keypads

WXK_CATEGORY_NAVIGATION union of WXK_CATEGORY_ARROW, WXK_CATEGORY_PAGING, and WXK_CATEGORY_JUMP categories

enum wxMouseWheelAxis

Possible axis values for mouse wheel scroll events.

Since

2.9.4

Enumerator

wxMOUSE_WHEEL_VERTICAL Vertical scroll event.

wxMOUSE_WHEEL_HORIZONTAL Horizontal scroll event.

enum wxUpdateUIMode

The possibles modes to pass to wxUpdateUIEvent::SetMode().

Enumerator

wxUPDATE_UI_PROCESS_ALL Send UI update events to all windows.

wxUPDATE_UI_PROCESS_SPECIFIED Send UI update events to windows that have the wxWS_EX_PROCESS_UI_UPDATES flag specified.

22.263 interface/wx/eventfilter.h File Reference

Classes

- class wxEventFilter

  A global event filter for pre-processing all the events generated in the program.

22.264 interface/wx/evtloop.h File Reference

Classes

- class wxEventLoopBase

  Base class for all event loop implementations.
- class wxEventLoopActivator

  Makes an event loop temporarily active.
- class wxGUIEventLoop

  A generic implementation of the GUI event loop.

22.265 interface/wx/fdrepdlg.h File Reference

Classes

- class wxFindDialogEvent

  wxFindReplaceDialog events.
- class wxFindReplaceData

  wxFindReplaceData holds the data for wxFindReplaceDialog.
- class wxFindReplaceDialog

  wxFindReplaceDialog is a standard modeless dialog which is used to allow the user to search for some text (and possibly replace it with something else).

Enumerations

- enum wxFindReplaceFlags {
  wxFR_DOWN = 1,
  wxFR_WHOLEWORD = 2,
  wxFR_MATCHCASE = 4 }

  See wxFindDialogEvent::GetFlags().
- enum wxFindReplaceDialogStyles {
  wxFR_REPLACEDIALOG = 1,
  wxFR_NOUPDOWN = 2,
  wxFR_NOMATCHCASE = 4,
  wxFR_NOWHOLEWORD = 8 }

  These flags can be specified in wxFindReplaceDialog ctor or Create():

Variables

- wxEventType wxEVT_FIND
- wxEventType wxEVT_FIND_NEXT
- wxEventType wxEVT_FIND_REPLACE
- wxEventType wxEVT_FIND_REPLACE_ALL
- wxEventType wxEVT_FIND_CLOSE

22.265.1 Enumeration Type Documentation

enum wxFindReplaceDialogStyles

These flags can be specified in wxFindReplaceDialog ctor or Create():

Enumerator

wxFR_REPLACEDIALOG replace dialog (otherwise find dialog)
wxFR_NOUPDOWN don't allow changing the search direction
wxFR_NOMATCHCASE don't allow case sensitive searching
wxFR_NOWHOLEWORD don't allow whole word searching

enum wxFindReplaceFlags

See wxFindDialogEvent::GetFlags().

Enumerator

wxFR_DOWN downward search/replace selected (otherwise - upwards)
wxFR_WHOLEWORD whole word search/replace selected
wxFR_MATCHCASE case sensitive search/replace selected (otherwise - case insensitive)

22.265.2 Variable Documentation

wxEventType wxEVT_FIND

wxEventType wxEVT_FIND_CLOSE

wxEventType wxEVT_FIND_NEXT

wxEventType wxEVT_FIND_REPLACE

wxEventType wxEVT_FIND_REPLACE_ALL

22.266 interface/wx/ffile.h File Reference

Classes

- class wxFFile
  wxFFile implements buffered file I/O.

22.267 interface/wx/fileconf.h File Reference

Classes

- class wxFileConfig
  wxFileConfig implements wxConfigBase interface for storing and retrieving configuration information using plain text files.

22.268 interface/wx/filectrl.h File Reference

Classes

- class wxFileCtrl
  This control allows the user to select a file.
- class wxFileCtrlEvent
  A file control event holds information about events associated with wxFileCtrl objects.

Macros

- #define wxFC_DEFAULT_STYLE wxFC_OPEN

Enumerations

- enum {
  wxFC_OPEN = 0x0001,
  wxFC_SAVE = 0x0002,
  wxFC_MULTIPLE = 0x0004,
  wxFC_NOSHOWHIDDEN = 0x0008 }

Variables

- wxEventType wxEVT_FILECTRL_SELECTIONCHANGED
- wxEventType wxEVT_FILECTRL_FILEACTIVATED
- wxEventType wxEVT_FILECTRL_FOLDERCHANGED
- wxEventType wxEVT_FILECTRL_FILTERCHANGED

22.268.1 Macro Definition Documentation

#define wxFC_DEFAULT_STYLE wxFC_OPEN

22.268.2 Enumeration Type Documentation

anonymous enum

Enumerator

wxFC_OPEN

wxFC_SAVE

wxFC_MULTIPLE

wxFC_NOSHOWHIDDEN

22.268.3 Variable Documentation

wxEventType wxEVT_FILECTRL_FILEACTIVATED

wxEventType wxEVT_FILECTRL_FILTERCHANGED

wxEventType wxEVT_FILECTRL_FOLDERCHANGED

wxEventType wxEVT_FILECTRL_SELECTIONCHANGED

22.269 interface/wx/filedlg.h File Reference

Classes

- class wxFileDialog

  This class represents the file chooser dialog.

Macros

- #define wxFD_DEFAULT_STYLE wxFD_OPEN

Enumerations

- enum {
  wxFD_OPEN = 0x0001,
  wxFD_SAVE = 0x0002,
  wxFD_OVERWRITE_PROMPT = 0x0004,
  wxFD_NO_FOLLOW = 0x0008,
  wxFD_FILE_MUST_EXIST = 0x0010,
  wxFD_MULTIPLE = 0x0020,
  wxFD_CHANGE_DIR = 0x0080,
  wxFD_PREVIEW = 0x0100 }

Functions

- wxString wxFileSelector (const wxString &message, const wxString &default_path=wxEmptyString, const wxString &default_filename=wxEmptyString, const wxString &default_extension=wxEmptyString, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, int flags=0, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord)

  Pops up a file selector box.
- wxString wxFileSelectorEx (const wxString &message=wxFileSelectorPromptStr, const wxString &default_path=wxEmptyString, const wxString &default_filename=wxEmptyString, int *indexDefaultExtension=NULL, const wxString &wildcard=wxFileSelectorDefaultWildcardStr, int flags=0, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord)

  An extended version of wxFileSelector.
- wxString wxLoadFileSelector (const wxString &what, const wxString &extension, const wxString &default_name=wxEmptyString, wxWindow *parent=NULL)

  Ask for filename to load.
- wxString wxSaveFileSelector (const wxString &what, const wxString &extension, const wxString &default_name=wxEmptyString, wxWindow *parent=NULL)

  Ask for filename to save.

Variables

- const char wxFileSelectorDefaultWildcardStr [ ]

  Default wildcard string used in wxFileDialog corresponding to all files.

22.269.1 Macro Definition Documentation

#define wxFD_DEFAULT_STYLE wxFD_OPEN

22.269.2 Enumeration Type Documentation

anonymous enum

Enumerator

wxFD_OPEN

wxFD_SAVE

wxFD_OVERWRITE_PROMPT

wxFD_NO_FOLLOW

wxFD_FILE_MUST_EXIST

wxFD_MULTIPLE

wxFD_CHANGE_DIR

wxFD_PREVIEW

22.269.3 Variable Documentation

const char wxFileSelectorDefaultWildcardStr[ ]

Default wildcard string used in wxFileDialog corresponding to all files.

It is defined as "∗.∗" under MSW and OS/2 and "∗" everywhere else.

22.270 interface/wx/filefn.h File Reference

Classes

- class wxPathList

  The path list is a convenient way of storing a number of directories, and when presented with a filename without a directory, searching for an existing file in those directories.

Macros

- #define wxCHANGE_UMASK(mask)

  Under Unix this macro changes the current process umask to the given value, unless it is equal to -1 in which case nothing is done, and restores it to the original value on scope exit.

Typedefs

- typedef off_t wxFileOffset

  The type used to store and provide byte offsets or byte sizes for files or streams.

Enumerations

- enum wxPosixPermissions {
  wxS_IRUSR = 00400,
  wxS_IWUSR = 00200,
  wxS_IXUSR = 00100,
  wxS_IRGRP = 00040,
  wxS_IWGRP = 00020,
  wxS_IXGRP = 00010,
  wxS_IROTH = 00004,
  wxS_IWOTH = 00002,
  wxS_IXOTH = 00001,
  wxPOSIX_USER_READ = wxS_IRUSR,
  wxPOSIX_USER_WRITE = wxS_IWUSR,
  wxPOSIX_USER_EXECUTE = wxS_IXUSR,
  wxPOSIX_GROUP_READ = wxS_IRGRP,
  wxPOSIX_GROUP_WRITE = wxS_IWGRP,
  wxPOSIX_GROUP_EXECUTE = wxS_IXGRP,
  wxPOSIX_OTHERS_READ = wxS_IROTH,
  wxPOSIX_OTHERS_WRITE = wxS_IWOTH,
  wxPOSIX_OTHERS_EXECUTE = wxS_IXOTH,
  wxS_DEFAULT,
  wxS_DIR_DEFAULT }

  File permission bit names.
- enum wxSeekMode {
  wxFromStart,
  wxFromCurrent,
  wxFromEnd }

  Parameter indicating how file offset should be interpreted.
- enum wxFileKind {
  wxFILE_KIND_UNKNOWN,
  wxFILE_KIND_DISK,
  wxFILE_KIND_TERMINAL,
  wxFILE_KIND_PIPE }

  File kind enumerations returned from wxGetFileKind().

Functions

- bool wxGetDiskSpace (const wxString &path, wxLongLong total=NULL, wxLongLong free=NULL)

  This function returns the total number of bytes and number of free bytes on the disk containing the directory path (it should exist).
- wxString wxGetOSDirectory ()

  Returns the Windows directory under Windows; other platforms return an empty string.
- int wxParseCommonDialogsFilter (const wxString &wildCard, wxArrayString &descriptions, wxArrayString &filters)

  Parses the wildCard, returning the number of filters.
- void wxDos2UnixFilename (wxChar *s)

  Converts a DOS to a Unix filename by replacing backslashes with forward slashes.
- void wxUnix2DosFilename (wxChar *s)

  Converts a Unix to a DOS filename by replacing forward slashes with backslashes.
- bool wxDirExists (const wxString &dirname)

  Returns true if dirname exists and is a directory.
- void wxSplitPath (const wxString &fullname, wxString *path, wxString *name, wxString *ext)
- time_t wxFileModificationTime (const wxString &filename)

  Returns time of last modification of given file.

- bool wxRenameFile (const wxString &file1, const wxString &file2, bool overwrite=true)

  Renames file1 to file2, returning true if successful.
- bool wxCopyFile (const wxString &file1, const wxString &file2, bool overwrite=true)

  Copies file1 to file2, returning true if successful.
- bool wxFileExists (const wxString &filename)

  Returns true if the file exists and is a plain file.
- bool wxMatchWild (const wxString &pattern, const wxString &text, bool dot_special)

  Returns true if the pattern matches the text; if dot_special is true, filenames beginning with a dot are not matched with wildcard characters.
- wxString wxGetWorkingDirectory (char *buf=NULL, int sz=1000)
- wxString wxPathOnly (const wxString &path)

  Returns the directory part of the filename.
- bool wxIsWild (const wxString &pattern)

  Returns true if the pattern contains wildcards.
- bool wxIsAbsolutePath (const wxString &filename)

  Returns true if the argument is an absolute filename, i.e. with a slash or drive name at the beginning.
- wxString wxGetCwd ()

  Returns a string containing the current (or working) directory.
- bool wxSetWorkingDirectory (const wxString &dir)

  Sets the current working directory, returning true if the operation succeeded.
- bool wxConcatFiles (const wxString &file1, const wxString &file2, const wxString &file3)

  Concatenates file1 and file2 to file3, returning true if successful.
- bool wxRemoveFile (const wxString &file)

  Removes file, returning true if successful.
- bool wxMkdir (const wxString &dir, int perm=wxS_DIR_DEFAULT)

  Makes the directory dir, returning true if successful.
- bool wxRmdir (const wxString &dir, int flags=0)

  Removes the directory dir, returning true if successful.
- wxString wxFindNextFile ()

  Returns the next file that matches the path passed to wxFindFirstFile().
- wxString wxFindFirstFile (const wxString &spec, int flags=0)

  This function does directory searching; returns the first file that matches the path spec, or the empty string.
- wxFileKind wxGetFileKind (int fd)

  Returns the type of an open file.
- wxFileKind wxGetFileKind (FILE *fp)
- wxString wxFileNameFromPath (const wxString &path)
- char * wxFileNameFromPath (char *path)
- char * wxGetTempFileName (const wxString &prefix, char *buf=NULL)
- bool wxGetTempFileName (const wxString &prefix, wxString &buf)

Variables

- const int wxInvalidOffset = -1

  A special return value of many wxWidgets classes to indicate that an invalid offset was given.

22.271 interface/wx/filehistory.h File Reference

Classes

- class wxFileHistory

  The wxFileHistory encapsulates a user interface convenience, the list of most recently visited files as shown on a menu (usually the File menu).

22.272 interface/wx/filename.h File Reference

Classes

- class wxFileName

  wxFileName encapsulates a file name.

Enumerations

- enum wxPathFormat {
  wxPATH_NATIVE = 0,
  wxPATH_UNIX,
  wxPATH_BEOS = wxPATH_UNIX,
  wxPATH_MAC,
  wxPATH_DOS,
  wxPATH_WIN = wxPATH_DOS,
  wxPATH_OS2 = wxPATH_DOS,
  wxPATH_VMS,
  wxPATH_MAX }

  The various values for the path format: this mainly affects the path separator but also whether or not the path has the drive part (as under Windows).
- enum wxSizeConvention {
  wxSIZE_CONV_TRADITIONAL,
  wxSIZE_CONV_IEC,
  wxSIZE_CONV_SI }

  Different conventions for human readable sizes.
- enum wxPathNormalize {
  wxPATH_NORM_ENV_VARS = 0x0001,
  wxPATH_NORM_DOTS = 0x0002,
  wxPATH_NORM_TILDE = 0x0004,
  wxPATH_NORM_CASE = 0x0008,
  wxPATH_NORM_ABSOLUTE = 0x0010,
  wxPATH_NORM_LONG = 0x0020,
  wxPATH_NORM_SHORTCUT = 0x0040,
  wxPATH_NORM_ALL = 0x00ff & ~wxPATH_NORM_CASE }

  The kind of normalization to do with the file name: these values can be or'd together to perform several operations at once.
- enum {
  wxPATH_RMDIR_FULL = 1,
  wxPATH_RMDIR_RECURSIVE = 2 }

  Flags for wxFileName::Rmdir().
- enum {
  wxFILE_EXISTS_REGULAR = 0x0001,
  wxFILE_EXISTS_DIR = 0x0002,
  wxFILE_EXISTS_SYMLINK = 0x1004,
  wxFILE_EXISTS_DEVICE = 0x0008,
  wxFILE_EXISTS_FIFO = 0x0016,
  wxFILE_EXISTS_SOCKET = 0x0032,
  wxFILE_EXISTS_NO_FOLLOW }

  Flags for wxFileName::Exists().

Variables

- wxULongLong wxInvalidSize

  The return value of wxFileName::GetSize() in case of error.

22.272.1 Enumeration Type Documentation

anonymous enum

Flags for wxFileName::Rmdir().

Enumerator

wxPATH_RMDIR_FULL Delete the specified directory and its subdirectories if they are empty.

wxPATH_RMDIR_RECURSIVE Delete the specified directory and all the files and subdirectories in it recursively. This flag is obviously **dangerous** and should be used with care and after asking the user for confirmation.

anonymous enum

Flags for wxFileName::Exists().

Since

2.9.5

Enumerator

wxFILE_EXISTS_REGULAR Check for existence of a regular file.

wxFILE_EXISTS_DIR Check for existence of a directory.

wxFILE_EXISTS_SYMLINK Check for existence of a symlink. Notice that this flag also sets wxFILE_EXISTS_NO_FOLLOW, otherwise it would never be satisfied as wxFileName::Exists() would be checking for the existence of the symlink target and not the symlink itself.

wxFILE_EXISTS_DEVICE Check for existence of a device.

wxFILE_EXISTS_FIFO Check for existence of a FIFO.

wxFILE_EXISTS_SOCKET Check for existence of a socket.

wxFILE_EXISTS_NO_FOLLOW Don't dereference a contained symbolic link. Check for existence of anything

enum wxPathFormat

The various values for the path format: this mainly affects the path separator but also whether or not the path has the drive part (as under Windows).

See wxFileName for more info.

Enumerator

wxPATH_NATIVE the path format for the current platform.

wxPATH_UNIX

wxPATH_BEOS

wxPATH_MAC

wxPATH_DOS

wxPATH_WIN

wxPATH_OS2

wxPATH_VMS

wxPATH_MAX Not a valid value for specifying path format.

enum wxPathNormalize

The kind of normalization to do with the file name: these values can be or'd together to perform several operations at once.

See wxFileName::Normalize() for more info.

Enumerator

wxPATH_NORM_ENV_VARS Replace environment variables with their values. wxFileName understands both Unix and Windows (but only under Windows) environment variables expansion: i.e. "`$var`", "`$(var)`" and "`${var}`" are always understood and in addition under Windows "`%var%`" is also.

wxPATH_NORM_DOTS Squeeze all "`..`" and "`.`".

wxPATH_NORM_TILDE Replace "`~`" and "`~user`" (Unix only).

wxPATH_NORM_CASE If the platform is case insensitive, make lowercase the path.

wxPATH_NORM_ABSOLUTE Make the path absolute.

wxPATH_NORM_LONG Expand the path to the "long" form (Windows only).

wxPATH_NORM_SHORTCUT Resolve the shortcut, if it is a shortcut (Windows only).

wxPATH_NORM_ALL A value indicating all normalization flags except for `wxPATH_NORM_CASE`.

enum wxSizeConvention

Different conventions for human readable sizes.

See also

wxFileName::GetHumanReadableSize().

Since

2.9.1

Enumerator

wxSIZE_CONV_TRADITIONAL 1024 bytes = 1KB.

wxSIZE_CONV_IEC 1024 bytes = 1KiB.

wxSIZE_CONV_SI 1000 bytes = 1KB.

22.272.2 Variable Documentation

wxULongLong wxInvalidSize

The return value of wxFileName::GetSize() in case of error.

22.273 interface/wx/filepicker.h File Reference

Classes

- class wxFilePickerCtrl

  This control allows the user to select a file.
- class wxDirPickerCtrl

  This control allows the user to select a directory.
- class wxFileDirPickerEvent

  This event class is used for the events generated by wxFilePickerCtrl and by wxDirPickerCtrl.

Macros

- #define wxFLP_OPEN 0x0400
- #define wxFLP_SAVE 0x0800
- #define wxFLP_OVERWRITE_PROMPT 0x1000
- #define wxFLP_FILE_MUST_EXIST 0x2000
- #define wxFLP_CHANGE_DIR 0x4000
- #define wxFLP_SMALL wxPB_SMALL
- #define wxFLP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)
- #define wxFLP_DEFAULT_STYLE (wxFLP_OPEN|wxFLP_FILE_MUST_EXIST)
- #define wxDIRP_DIR_MUST_EXIST 0x0008
- #define wxDIRP_CHANGE_DIR 0x0010
- #define wxDIRP_SMALL wxPB_SMALL
- #define wxDIRP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)
- #define wxDIRP_DEFAULT_STYLE (wxDIRP_DIR_MUST_EXIST)

Variables

- wxEventType wxEVT_FILEPICKER_CHANGED
- wxEventType wxEVT_DIRPICKER_CHANGED

22.273.1 Macro Definition Documentation

#define wxDIRP_CHANGE_DIR 0x0010

#define wxDIRP_DEFAULT_STYLE (wxDIRP_DIR_MUST_EXIST)

#define wxDIRP_DIR_MUST_EXIST 0x0008

#define wxDIRP_SMALL wxPB_SMALL

#define wxDIRP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)

#define wxFLP_CHANGE_DIR 0x4000

#define wxFLP_DEFAULT_STYLE (wxFLP_OPEN|wxFLP_FILE_MUST_EXIST)

#define wxFLP_FILE_MUST_EXIST 0x2000

#define wxFLP_OPEN 0x0400

#define wxFLP_OVERWRITE_PROMPT 0x1000

#define wxFLP_SAVE 0x0800

#define wxFLP_SMALL wxPB_SMALL

#define wxFLP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)

22.273.2 Variable Documentation

wxEventType wxEVT_DIRPICKER_CHANGED

wxEventType wxEVT_FILEPICKER_CHANGED

22.274 interface/wx/filesys.h File Reference

Classes

- class wxFileSystem

  This class provides an interface for opening files on different file systems.
- class wxFSFile

  This class represents a single file opened by wxFileSystem.
- class wxFileSystemHandler

  Classes derived from wxFileSystemHandler are used to access virtual file systems.
- class wxFSInputStream

  Input stream for virtual file stream files.

Enumerations

- enum wxFileSystemOpenFlags {
  wxFS_READ = 1,
  wxFS_SEEKABLE = 4 }

  Open Bit Flags.

22.274.1 Enumeration Type Documentation

enum wxFileSystemOpenFlags

Open Bit Flags.

Enumerator

wxFS_READ Open for reading.

wxFS_SEEKABLE Returned stream will be seekable.

22.275 interface/wx/fontdata.h File Reference

Classes

- class wxFontData

  This class holds a variety of information related to font dialogs.

22.276 interface/wx/fontdlg.h File Reference

Classes

- class wxFontDialog

  This class represents the font chooser dialog.

Functions

- wxFont wxGetFontFromUser (wxWindow *parent, const wxFont &fontInit, const wxString &caption=wxEmptyString)

  Shows the font selection dialog and returns the font selected by user or invalid font (use wxFont::IsOk() to test whether a font is valid) if the dialog was cancelled.

22.277 interface/wx/fontenum.h File Reference

Classes

- class wxFontEnumerator

  wxFontEnumerator enumerates either all available fonts on the system or only the ones with given attributes - either only fixed-width (suited for use in programs such as terminal emulators and the like) or the fonts available in the given encoding).

22.278 interface/wx/fontmap.h File Reference

Classes

- class wxFontMapper

  wxFontMapper manages user-definable correspondence between logical font names and the fonts present on the machine.

22.279 interface/wx/fontpicker.h File Reference

Classes

- class wxFontPickerCtrl

  This control allows the user to select a font.
- class wxFontPickerEvent

  This event class is used for the events generated by wxFontPickerCtrl.

Macros

- #define wxFNTP_FONTDESC_AS_LABEL 0x0008
- #define wxFNTP_USEFONT_FOR_LABEL 0x0010
- #define wxFONTBTN_DEFAULT_STYLE (wxFNTP_FONTDESC_AS_LABEL | wxFNTP_USEFONT_FOR↩_LABEL)
- #define wxFNTP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)
- #define wxFNTP_DEFAULT_STYLE (wxFNTP_FONTDESC_AS_LABEL|wxFNTP_USEFONT_FOR_LAB↩EL)

Variables

- wxEventType wxEVT_FONTPICKER_CHANGED

22.279.1 Macro Definition Documentation

#define wxFNTP_DEFAULT_STYLE (wxFNTP_FONTDESC_AS_LABEL|wxFNTP_USEFONT_FOR_LABEL)

#define wxFNTP_FONTDESC_AS_LABEL 0x0008

#define wxFNTP_USE_TEXTCTRL (wxPB_USE_TEXTCTRL)

#define wxFNTP_USEFONT_FOR_LABEL 0x0010

#define wxFONTBTN_DEFAULT_STYLE (wxFNTP_FONTDESC_AS_LABEL | wxFNTP_USEFONT_FOR_LABEL)

22.279.2 Variable Documentation

wxEventType wxEVT_FONTPICKER_CHANGED

22.280 interface/wx/fontutil.h File Reference

Classes

- class wxNativeFontInfo

  wxNativeFontInfo is platform-specific font representation: this class should be considered as an opaque font description only used by the native functions, the user code can only get the objects of this type from somewhere and pass it somewhere else (possibly save them somewhere using ToString() and restore them using FromString())

22.281 interface/wx/frame.h File Reference

Classes

- class wxFrame

  A frame is a window whose size and position can (usually) be changed by the user.

Macros

- #define wxFRAME_NO_TASKBAR 0x0002

  Frame specific styles.
- #define wxFRAME_TOOL_WINDOW 0x0004
- #define wxFRAME_FLOAT_ON_PARENT 0x0008

22.281.1 Macro Definition Documentation

#define wxFRAME_FLOAT_ON_PARENT 0x0008

#define wxFRAME_NO_TASKBAR 0x0002

Frame specific styles.

#define wxFRAME_TOOL_WINDOW 0x0004

22.282 interface/wx/fs_arc.h File Reference

Classes

- class wxArchiveFSHandler

  A file system handler for accessing files inside of archives.

Typedefs

- typedef wxArchiveFSHandler wxZipFSHandler

22.282.1 Typedef Documentation

typedef wxArchiveFSHandler wxZipFSHandler

22.283 interface/wx/fs_filter.h File Reference

Classes

- class wxFilterFSHandler

  Filter file system handler.

22.284 interface/wx/fs_inet.h File Reference

Classes

- class wxInternetFSHandler

  A file system handler for accessing files from internet servers.

22.285 interface/wx/fs_mem.h File Reference

Classes

- class wxMemoryFSHandler

  This wxFileSystem handler can store arbitrary data in memory stream and make them accessible via an URL.

22.286 interface/wx/fswatcher.h File Reference

Classes

- class wxFileSystemWatcher

  The wxFileSystemWatcher class allows to receive notifications of file system changes.
- class wxFileSystemWatcherEvent

  A class of events sent when a file system event occurs.

Enumerations

- enum wxFSWFlags {
  wxFSW_EVENT_CREATE = 0x01,
  wxFSW_EVENT_DELETE = 0x02,
  wxFSW_EVENT_RENAME = 0x04,
  wxFSW_EVENT_MODIFY = 0x08,
  wxFSW_EVENT_ACCESS = 0x10,
  wxFSW_EVENT_ATTRIB = 0x20,
  wxFSW_EVENT_UNMOUNT = 0x2000,
  wxFSW_EVENT_WARNING = 0x40,
  wxFSW_EVENT_ERROR = 0x80,
  wxFSW_EVENT_ALL }

  These are the possible types of file system change events.

- enum wxFSWWarningType {
  wxFSW_WARNING_NONE,
  wxFSW_WARNING_GENERAL,
  wxFSW_WARNING_OVERFLOW }

  Possible warning types for the warning events generated by wxFileSystemWatcher.

Variables

- wxEventType wxEVT_FSWATCHER

22.286.1 Enumeration Type Documentation

enum wxFSWFlags

These are the possible types of file system change events.

Not all of these events are reported on all platforms currently.

Since

2.9.1

Enumerator

wxFSW_EVENT_CREATE File or directory was created.

wxFSW_EVENT_DELETE File or directory was deleted.

wxFSW_EVENT_RENAME File or directory was renamed. Notice that under MSW this event is sometimes – although not always – followed by a wxFSW_EVENT_MODIFY for the new file.

Under OS X this event is currently not detected and instead separate wxFSW_EVENT_CREATE and wxFSW_EVENT_DELETE events are.

wxFSW_EVENT_MODIFY File or directory was modified. Depending on the program doing the file modification, multiple such events can be reported for a single logical file update.

Under OS X this event is currently not detected.

wxFSW_EVENT_ACCESS File or directory was accessed. This event is currently only detected under Linux.

wxFSW_EVENT_ATTRIB The item's metadata was changed, e.g. its permissions or timestamps. This event is currently only detected under Linux.

Since

2.9.5

wxFSW_EVENT_UNMOUNT The file system containing a watched item was unmounted. wxFSW_EVENT_UNMOUNT cannot be set; unmount events are produced automatically. This flag is therefore not included in wxFSW_EVENT_ALL.

This event is currently only detected under Linux.

Since

2.9.5

wxFSW_EVENT_WARNING A warning condition arose. This is something that probably needs to be shown to the user in an interactive program as it can indicate a relatively serious problem, e.g. some events could have been missed because of an overflow. But more events will still be coming in the future, unlike for the error condition below.

wxFSW_EVENT_ERROR An error condition arose. Errors are fatal, i.e. no more events will be reported after an error and the program can stop watching the directories currently being monitored.

wxFSW_EVENT_ALL

enum wxFSWWarningType

Possible warning types for the warning events generated by wxFileSystemWatcher.

Since

3.0

Enumerator

wxFSW_WARNING_NONE This is not a warning at all.

wxFSW_WARNING_GENERAL A generic warning. Further information may be provided in the user-readable message available from wxFileSystemWatcherEvent::GetErrorDescription()

wxFSW_WARNING_OVERFLOW An overflow event. This warning indicates that some file system changes were not signaled by any events, usually because there were too many of them and the internally used queue has overflown. If such event is received it is recommended to completely rescan the files or directories being monitored.

22.286.2 Variable Documentation

wxEventType wxEVT_FSWATCHER

22.287 interface/wx/gauge.h File Reference

Classes

- class wxGauge

  A gauge is a horizontal or vertical bar which shows a quantity (often time).

Macros

- #define wxGA_HORIZONTAL wxHORIZONTAL
- #define wxGA_VERTICAL wxVERTICAL
- #define wxGA_PROGRESS 0x0010
- #define wxGA_SMOOTH 0x0020
- #define wxGA_TEXT 0x0040

22.287.1 Macro Definition Documentation

#define wxGA_HORIZONTAL wxHORIZONTAL

#define wxGA_PROGRESS 0x0010

#define wxGA_SMOOTH 0x0020

#define wxGA_TEXT 0x0040

#define wxGA_VERTICAL wxVERTICAL

22.288 interface/wx/gbsizer.h File Reference

Classes

- class wxGBPosition

This class represents the position of an item in a virtual grid of rows and columns managed by a wxGridBagSizer.

- class wxGridBagSizer

  A wxSizer that can lay out items in a virtual grid like a wxFlexGridSizer but in this case explicit positioning of the items is allowed using wxGBPosition, and items can optionally span more than one row and/or column using wxGBSpan.

- class wxGBSizerItem

  The wxGBSizerItem class is used by the wxGridBagSizer for tracking the items in the sizer.

- class wxGBSpan

  This class is used to hold the row and column spanning attributes of items in a wxGridBagSizer.

Variables

- const wxGBSpan wxDefaultSpan

22.288.1 Variable Documentation

const wxGBSpan wxDefaultSpan

22.289 interface/wx/gdicmn.h File Reference

Classes

- class wxRealPoint

  A wxRealPoint is a useful data structure for graphics operations.

- class wxRect

  A class for manipulating rectangles.

- class wxPoint

  A wxPoint is a useful data structure for graphics operations.

- class wxColourDatabase

  wxWidgets maintains a database of standard RGB colours for a predefined set of named colours.

- class wxSize

  A wxSize is a useful data structure for graphics operations.

Macros

- #define wxBITMAP(bitmapName)

  This macro loads a bitmap from either application resources (on the platforms for which they exist, i.e. Windows) or from an XPM file.

- #define wxBITMAP_PNG(bitmapName)

  Creates a bitmap from either application resources or embedded image data in PNG format.

- #define wxBITMAP_PNG_FROM_DATA(bitmapName)

  Creates a bitmap from embedded image data in PNG format.

- #define wxICON(iconName)

  This macro loads an icon from either application resources (on the platforms for which they exist, i.e. Windows) or from an XPM file.

Enumerations

- enum wxBitmapType {
  wxBITMAP_TYPE_INVALID,
  wxBITMAP_TYPE_BMP,
  wxBITMAP_TYPE_BMP_RESOURCE,
  wxBITMAP_TYPE_RESOURCE = wxBITMAP_TYPE_BMP_RESOURCE,
  wxBITMAP_TYPE_ICO,
  wxBITMAP_TYPE_ICO_RESOURCE,
  wxBITMAP_TYPE_CUR,
  wxBITMAP_TYPE_CUR_RESOURCE,
  wxBITMAP_TYPE_XBM,
  wxBITMAP_TYPE_XBM_DATA,
  wxBITMAP_TYPE_XPM,
  wxBITMAP_TYPE_XPM_DATA,
  wxBITMAP_TYPE_TIFF,
  wxBITMAP_TYPE_TIF = wxBITMAP_TYPE_TIFF,
  wxBITMAP_TYPE_TIFF_RESOURCE,
  wxBITMAP_TYPE_TIF_RESOURCE = wxBITMAP_TYPE_TIFF_RESOURCE,
  wxBITMAP_TYPE_GIF,
  wxBITMAP_TYPE_GIF_RESOURCE,
  wxBITMAP_TYPE_PNG,
  wxBITMAP_TYPE_PNG_RESOURCE,
  wxBITMAP_TYPE_JPEG,
  wxBITMAP_TYPE_JPEG_RESOURCE,
  wxBITMAP_TYPE_PNM,
  wxBITMAP_TYPE_PNM_RESOURCE,
  wxBITMAP_TYPE_PCX,
  wxBITMAP_TYPE_PCX_RESOURCE,
  wxBITMAP_TYPE_PICT,
  wxBITMAP_TYPE_PICT_RESOURCE,
  wxBITMAP_TYPE_ICON,
  wxBITMAP_TYPE_ICON_RESOURCE,
  wxBITMAP_TYPE_ANI,
  wxBITMAP_TYPE_IFF,
  wxBITMAP_TYPE_TGA,
  wxBITMAP_TYPE_MACCURSOR,
  wxBITMAP_TYPE_MACCURSOR_RESOURCE,
  wxBITMAP_TYPE_ANY = 50 }

  Bitmap type flags.

- enum wxPolygonFillMode {
  wxODDEVEN_RULE = 1,
  wxWINDING_RULE }

  Polygon filling mode.

- enum wxStockCursor {
  wxCURSOR_NONE,
  wxCURSOR_ARROW,
  wxCURSOR_RIGHT_ARROW,
  wxCURSOR_BULLSEYE,
  wxCURSOR_CHAR,
  wxCURSOR_CROSS,
  wxCURSOR_HAND,
  wxCURSOR_IBEAM,
  wxCURSOR_LEFT_BUTTON,
  wxCURSOR_MAGNIFIER,
  wxCURSOR_MIDDLE_BUTTON,
  wxCURSOR_NO_ENTRY,
  wxCURSOR_PAINT_BRUSH,
  wxCURSOR_PENCIL,
  wxCURSOR_POINT_LEFT,
  wxCURSOR_POINT_RIGHT,
  wxCURSOR_QUESTION_ARROW,
  wxCURSOR_RIGHT_BUTTON,
  wxCURSOR_SIZENESW,
  wxCURSOR_SIZENS,
  wxCURSOR_SIZENWSE,
  wxCURSOR_SIZEWE,
  wxCURSOR_SIZING,
  wxCURSOR_SPRAYCAN,
  wxCURSOR_WAIT,
  wxCURSOR_WATCH,
  wxCURSOR_BLANK,
  wxCURSOR_DEFAULT,
  wxCURSOR_COPY_ARROW,
  wxCURSOR_CROSS_REVERSE,
  wxCURSOR_DOUBLE_ARROW,
  wxCURSOR_BASED_ARROW_UP,
  wxCURSOR_BASED_ARROW_DOWN,
  wxCURSOR_ARROWWAIT,
  wxCURSOR_MAX }

  Standard cursors.

Functions

- bool wxColourDisplay ()

  Returns true if the display is colour, false otherwise.
- int wxDisplayDepth ()

  Returns the depth of the display (a value of 1 denotes a monochrome display).
- void wxSetCursor (const wxCursor &cursor)

  Globally sets the cursor; only has an effect on Windows, Mac and GTK+.
- void wxClientDisplayRect (int *x, int *y, int *width, int *height)

  Returns the dimensions of the work area on the display.
- wxRect wxGetClientDisplayRect ()

  Returns the dimensions of the work area on the display.
- wxSize wxGetDisplayPPI ()

  Returns the display resolution in pixels per inch.
- void wxDisplaySize (int *width, int *height)

  Returns the display size in pixels.
- wxSize wxGetDisplaySize ()

Returns the display size in pixels.

- void wxDisplaySizeMM (int *width, int *height)

  Returns the display size in millimeters.

- wxSize wxGetDisplaySizeMM ()

  Returns the display size in millimeters.

Variables

- const wxPoint wxDefaultPosition

  Global instance of a wxPoint initialized with values (-1,-1).

- wxColourDatabase * wxTheColourDatabase

  Global instance of a wxColourDatabase.

- const wxSize wxDefaultSize

  Global instance of a wxSize object initialized to (-1,-1).

22.289.1 Enumeration Type Documentation

enum wxBitmapType

Bitmap type flags.

See wxBitmap and wxImage classes.

Enumerator

wxBITMAP_TYPE_INVALID

wxBITMAP_TYPE_BMP

wxBITMAP_TYPE_BMP_RESOURCE

wxBITMAP_TYPE_RESOURCE

wxBITMAP_TYPE_ICO

wxBITMAP_TYPE_ICO_RESOURCE

wxBITMAP_TYPE_CUR

wxBITMAP_TYPE_CUR_RESOURCE

wxBITMAP_TYPE_XBM

wxBITMAP_TYPE_XBM_DATA

wxBITMAP_TYPE_XPM

wxBITMAP_TYPE_XPM_DATA

wxBITMAP_TYPE_TIFF

wxBITMAP_TYPE_TIF

wxBITMAP_TYPE_TIFF_RESOURCE

wxBITMAP_TYPE_TIF_RESOURCE

wxBITMAP_TYPE_GIF

wxBITMAP_TYPE_GIF_RESOURCE

wxBITMAP_TYPE_PNG

wxBITMAP_TYPE_PNG_RESOURCE

wxBITMAP_TYPE_JPEG

wxBITMAP_TYPE_JPEG_RESOURCE

wxBITMAP_TYPE_PNM

wxBITMAP_TYPE_PNM_RESOURCE

wxBITMAP_TYPE_PCX

wxBITMAP_TYPE_PCX_RESOURCE

wxBITMAP_TYPE_PICT

wxBITMAP_TYPE_PICT_RESOURCE

wxBITMAP_TYPE_ICON

wxBITMAP_TYPE_ICON_RESOURCE

wxBITMAP_TYPE_ANI

wxBITMAP_TYPE_IFF

wxBITMAP_TYPE_TGA

wxBITMAP_TYPE_MACCURSOR

wxBITMAP_TYPE_MACCURSOR_RESOURCE

wxBITMAP_TYPE_ANY

enum wxPolygonFillMode

Polygon filling mode.

See wxDC::DrawPolygon.

Enumerator

wxODDEVEN_RULE

wxWINDING_RULE

enum wxStockCursor

Standard cursors.

Notice that under wxMSW some of these cursors are defined in `wx.rc` file and not by the system itself so you should include this file from your own resource file (possibly creating a trivial resource file just containing a single include line if you don't need it otherwise) to be able to use them.

See wxCursor.

Enumerator

wxCURSOR_NONE

wxCURSOR_ARROW A standard arrow cursor.

wxCURSOR_RIGHT_ARROW A standard arrow cursor pointing to the right.

wxCURSOR_BULLSEYE Bullseye cursor.

wxCURSOR_CHAR Rectangular character cursor.

wxCURSOR_CROSS A cross cursor.

wxCURSOR_HAND A hand cursor.

wxCURSOR_IBEAM An I-beam cursor (vertical line).

wxCURSOR_LEFT_BUTTON Represents a mouse with the left button depressed.

wxCURSOR_MAGNIFIER A magnifier icon.

wxCURSOR_MIDDLE_BUTTON Represents a mouse with the middle button depressed.

wxCURSOR_NO_ENTRY A no-entry sign cursor.

wxCURSOR_PAINT_BRUSH A paintbrush cursor.

wxCURSOR_PENCIL A pencil cursor.

wxCURSOR_POINT_LEFT A cursor that points left.
wxCURSOR_POINT_RIGHT A cursor that points right.
wxCURSOR_QUESTION_ARROW An arrow and question mark.
wxCURSOR_RIGHT_BUTTON Represents a mouse with the right button depressed.
wxCURSOR_SIZENESW A sizing cursor pointing NE-SW.
wxCURSOR_SIZENS A sizing cursor pointing N-S.
wxCURSOR_SIZENWSE A sizing cursor pointing NW-SE.
wxCURSOR_SIZEWE A sizing cursor pointing W-E.
wxCURSOR_SIZING A general sizing cursor.
wxCURSOR_SPRAYCAN A spraycan cursor.
wxCURSOR_WAIT A wait cursor.
wxCURSOR_WATCH A watch cursor.
wxCURSOR_BLANK Transparent cursor.
wxCURSOR_DEFAULT Standard X11 cursor (only in wxGTK).
wxCURSOR_COPY_ARROW MacOS Theme Plus arrow (only in wxMac).
wxCURSOR_CROSS_REVERSE Only available on wxX11.
wxCURSOR_DOUBLE_ARROW Only available on wxX11.
wxCURSOR_BASED_ARROW_UP Only available on wxX11.
wxCURSOR_BASED_ARROW_DOWN Only available on wxX11.
wxCURSOR_ARROWWAIT A wait cursor with a standard arrow.
wxCURSOR_MAX

22.289.2 Variable Documentation

const wxPoint wxDefaultPosition

Global instance of a wxPoint initialized with values (-1,-1).

const wxSize wxDefaultSize

Global instance of a wxSize object initialized to (-1,-1).

wxColourDatabase* wxTheColourDatabase

Global instance of a wxColourDatabase.

22.290 interface/wx/gdiobj.h File Reference

Classes

- class wxGDIObject
  This class allows platforms to implement functionality to optimise GDI objects, such as wxPen, wxBrush and wxFont.

22.291 interface/wx/generic/aboutdlgg.h File Reference

Classes

- class wxGenericAboutDialog
  This class defines a customizable About dialog.

Functions

- void wxGenericAboutBox (const wxAboutDialogInfo &info, wxWindow *parent=NULL)

  Show generic about dialog.

22.291.1 Function Documentation

void wxGenericAboutBox (const wxAboutDialogInfo & *info*, wxWindow * *parent* = NULL)

Show generic about dialog.

This function does the same thing as wxAboutBox() except that it always uses the generic wxWidgets version of the dialog instead of the native one.

22.292 interface/wx/generic/helpext.h File Reference

Classes

- class wxExtHelpController

  This class implements help via an external browser.

22.293 interface/wx/geometry.h File Reference

Classes

- class wxPoint2DInt
- class wxPoint2DDouble
- class wxRect2DDouble
- class wxRect2DInt
- class wxTransform2D

Enumerations

- enum wxOutCode {
  wxInside = 0x00,
  wxOutLeft = 0x01,
  wxOutRight = 0x02,
  wxOutTop = 0x08,
  wxOutBottom = 0x04 }

Functions

- wxPoint2DInt operator+ (const wxPoint2DInt &pt1, const wxPoint2DInt &pt2)
- wxPoint2DInt operator- (const wxPoint2DInt &pt1, const wxPoint2DInt &pt2)
- wxPoint2DInt operator* (const wxPoint2DInt &pt1, const wxPoint2DInt &pt2)
- wxPoint2DInt operator* (wxInt32 n, const wxPoint2DInt &pt)
- wxPoint2DInt operator* (const wxPoint2DInt &pt, wxInt32 n)
- wxPoint2DInt operator/ (const wxPoint2DInt &pt1, const wxPoint2DInt &pt2)
- wxPoint2DInt operator/ (const wxPoint2DInt &pt, wxInt32 n)
- wxPoint2DDouble operator+ (const wxPoint2DDouble &pt1, const wxPoint2DDouble &pt2)
- wxPoint2DDouble operator- (const wxPoint2DDouble &pt1, const wxPoint2DDouble &pt2)

- wxPoint2DDouble operator∗ (const wxPoint2DDouble &pt1, const wxPoint2DDouble &pt2)
- wxPoint2DDouble operator∗ (wxDouble n, const wxPoint2DDouble &pt)
- wxPoint2DDouble operator∗ (wxInt32 n, const wxPoint2DDouble &pt)
- wxPoint2DDouble operator∗ (const wxPoint2DDouble &pt, wxDouble n)
- wxPoint2DDouble operator∗ (const wxPoint2DDouble &pt, wxInt32 n)
- wxPoint2DDouble operator/ (const wxPoint2DDouble &pt1, const wxPoint2DDouble &pt2)
- wxPoint2DDouble operator/ (const wxPoint2DDouble &pt, wxDouble n)
- wxPoint2DDouble operator/ (const wxPoint2DDouble &pt, wxInt32 n)

22.293.1 Enumeration Type Documentation

enum wxOutCode

Enumerator

wxInside

wxOutLeft

wxOutRight

wxOutTop

wxOutBottom

22.293.2 Function Documentation

wxPoint2DInt operator∗ (const wxPoint2DInt & *pt1*, const wxPoint2DInt & *pt2*)

wxPoint2DInt operator∗ (wxInt32 *n*, const wxPoint2DInt & *pt*)

wxPoint2DInt operator∗ (const wxPoint2DInt & *pt*, wxInt32 *n*)

wxPoint2DDouble operator∗ (const wxPoint2DDouble & *pt1*, const wxPoint2DDouble & *pt2*)

wxPoint2DDouble operator∗ (wxDouble *n*, const wxPoint2DDouble & *pt*)

wxPoint2DDouble operator∗ (wxInt32 *n*, const wxPoint2DDouble & *pt*)

wxPoint2DDouble operator∗ (const wxPoint2DDouble & *pt*, wxDouble *n*)

wxPoint2DDouble operator∗ (const wxPoint2DDouble & *pt*, wxInt32 *n*)

wxPoint2DInt operator+ (const wxPoint2DInt & *pt1*, const wxPoint2DInt & *pt2*)

wxPoint2DDouble operator+ (const wxPoint2DDouble & *pt1*, const wxPoint2DDouble & *pt2*)

wxPoint2DInt operator- (const wxPoint2DInt & *pt1*, const wxPoint2DInt & *pt2*)

wxPoint2DDouble operator- (const wxPoint2DDouble & *pt1*, const wxPoint2DDouble & *pt2*)

wxPoint2DInt operator/ (const wxPoint2DInt & *pt1*, const wxPoint2DInt & *pt2*)

wxPoint2DInt operator/ (const wxPoint2DInt & *pt*, wxInt32 *n*)

wxPoint2DDouble operator/ (const wxPoint2DDouble & *pt1*, const wxPoint2DDouble & *pt2*)

wxPoint2DDouble operator/ (const wxPoint2DDouble & *pt*, wxDouble *n*)

wxPoint2DDouble operator/ (const wxPoint2DDouble & *pt*, wxInt32 *n*)

22.294 interface/wx/glcanvas.h File Reference

Classes

- class wxGLContext

  An instance of a wxGLContext represents the state of an OpenGL state machine and the connection between OpenGL and the system.
- class wxGLCanvas

  wxGLCanvas is a class for displaying OpenGL graphics.

Enumerations

- enum {
  WX_GL_RGBA = 1,
  WX_GL_BUFFER_SIZE,
  WX_GL_LEVEL,
  WX_GL_DOUBLEBUFFER,
  WX_GL_STEREO,
  WX_GL_AUX_BUFFERS,
  WX_GL_MIN_RED,
  WX_GL_MIN_GREEN,
  WX_GL_MIN_BLUE,
  WX_GL_MIN_ALPHA,
  WX_GL_DEPTH_SIZE,
  WX_GL_STENCIL_SIZE,
  WX_GL_MIN_ACCUM_RED,
  WX_GL_MIN_ACCUM_GREEN,
  WX_GL_MIN_ACCUM_BLUE,
  WX_GL_MIN_ACCUM_ALPHA,
  WX_GL_SAMPLE_BUFFERS,
  WX_GL_SAMPLES,
  WX_GL_CORE_PROFILE,
  WX_GL_MAJOR_VERSION,
  WX_GL_MINOR_VERSION }

22.294.1 Enumeration Type Documentation

anonymous enum

Constants for use with wxGLCanvas.

Note

Not all implementations support options such as stereo, auxiliary buffers, alpha channel, and accumulator buffer, use wxGLCanvas::IsDisplaySupported() to check for individual attributes support.

Enumerator

WX_GL_RGBA Use true color (the default if no attributes at all are specified); do not use a palette.

WX_GL_BUFFER_SIZE Specifies the number of bits for buffer if not WX_GL_RGBA.

WX_GL_LEVEL Must be followed by 0 for main buffer, >0 for overlay, <0 for underlay.

WX_GL_DOUBLEBUFFER Use double buffering if present (on if no attributes specified).

WX_GL_STEREO Use stereoscopic display.

WX_GL_AUX_BUFFERS Specifies number of auxiliary buffers.

WX_GL_MIN_RED Use red buffer with most bits (> MIN_RED bits)

WX_GL_MIN_GREEN Use green buffer with most bits (> MIN_GREEN bits)

WX_GL_MIN_BLUE Use blue buffer with most bits (> MIN_BLUE bits)

WX_GL_MIN_ALPHA Use alpha buffer with most bits (> MIN_ALPHA bits)

WX_GL_DEPTH_SIZE Specifies number of bits for Z-buffer (typically 0, 16 or 32).

WX_GL_STENCIL_SIZE Specifies number of bits for stencil buffer.

WX_GL_MIN_ACCUM_RED Specifies minimal number of red accumulator bits.

WX_GL_MIN_ACCUM_GREEN Specifies minimal number of green accumulator bits.

WX_GL_MIN_ACCUM_BLUE Specifies minimal number of blue accumulator bits.

WX_GL_MIN_ACCUM_ALPHA Specifies minimal number of alpha accumulator bits.

WX_GL_SAMPLE_BUFFERS 1 for multisampling support (antialiasing)

WX_GL_SAMPLES 4 for 2x2 antialiasing supersampling on most graphics cards

WX_GL_CORE_PROFILE Request an OpenGL core profile. Notice that using this attribute will result in also requesting OpenGL at least version 3.0.

See WX_GL_MAJOR_VERSION and WX_GL_MINOR_VERSION for more precise version selection.

Since

3.1.0

WX_GL_MAJOR_VERSION Request specific OpenGL core major version number (>= 3). This attribute should be followed by the major version number requested.

It has no effect under OS X where specifying WX_GL_CORE_PROFILE will result in using OpenGL version at least 3.2 but can still be used there for portability.

Since

3.1.0

WX_GL_MINOR_VERSION Request specific OpenGL core minor version number. This attribute has the same semantics as WX_GL_MAJOR_VERSION but is for the minor OpenGL version, e.g. 2 if OpenGL 3.2 is requested.

Since

3.1.0

22.295 interface/wx/graphics.h File Reference

Classes

- class wxGraphicsPath

  A wxGraphicsPath is a native representation of a geometric path.
- class wxGraphicsObject

  This class is the superclass of native graphics objects like pens etc.
- class wxGraphicsBitmap

  Represents a bitmap.
- class wxGraphicsContext

  A wxGraphicsContext instance is the object that is drawn upon.
- class wxGraphicsGradientStop

  Represents a single gradient stop in a collection of gradient stops as represented by wxGraphicsGradientStops.
- class wxGraphicsGradientStops

  Represents a collection of wxGraphicGradientStop values for use with CreateLinearGradientBrush and CreateRadialGradientBrush.

- class wxGraphicsRenderer

  A wxGraphicsRenderer is the instance corresponding to the rendering engine used.
- class wxGraphicsBrush

  A wxGraphicsBrush is a native representation of a brush.
- class wxGraphicsFont

  A wxGraphicsFont is a native representation of a font.
- class wxGraphicsPen

  A wxGraphicsPen is a native representation of a pen.
- class wxGraphicsMatrix

  A wxGraphicsMatrix is a native representation of an affine matrix.

Enumerations

- enum wxAntialiasMode {
  wxANTIALIAS_NONE,
  wxANTIALIAS_DEFAULT }

  Anti-aliasing modes used by wxGraphicsContext::SetAntialiasMode().
- enum wxInterpolationQuality {
  wxINTERPOLATION_DEFAULT,
  wxINTERPOLATION_NONE,
  wxINTERPOLATION_FAST,
  wxINTERPOLATION_GOOD,
  wxINTERPOLATION_BEST }

  Interpolation quality used by wxGraphicsContext::SetInterpolationQuality().
- enum wxCompositionMode {
  wxCOMPOSITION_INVALID = -1,
  wxCOMPOSITION_CLEAR,
  wxCOMPOSITION_SOURCE,
  wxCOMPOSITION_OVER,
  wxCOMPOSITION_IN,
  wxCOMPOSITION_OUT,
  wxCOMPOSITION_ATOP,
  wxCOMPOSITION_DEST,
  wxCOMPOSITION_DEST_OVER,
  wxCOMPOSITION_DEST_IN,
  wxCOMPOSITION_DEST_OUT,
  wxCOMPOSITION_DEST_ATOP,
  wxCOMPOSITION_XOR,
  wxCOMPOSITION_ADD }

  Compositing is done using Porter-Duff compositions (see `http://keithp.com/~keithp/porterduff/p253-porter.↩pdf` *) with wxGraphicsContext::SetCompositionMode().*

Variables

- const wxGraphicsPen wxNullGraphicsPen
- const wxGraphicsBrush wxNullGraphicsBrush
- const wxGraphicsFont wxNullGraphicsFont
- const wxGraphicsBitmap wxNullGraphicsBitmap
- const wxGraphicsMatrix wxNullGraphicsMatrix
- const wxGraphicsPath wxNullGraphicsPath

22.295.1 Enumeration Type Documentation

enum wxAntialiasMode

Anti-aliasing modes used by wxGraphicsContext::SetAntialiasMode().

Enumerator

wxANTIALIAS_NONE No anti-aliasing.

wxANTIALIAS_DEFAULT The default anti-aliasing.

enum wxCompositionMode

Compositing is done using Porter-Duff compositions (see http://keithp.com/~keithp/porterduff/p253-porter.pdf) with wxGraphicsContext::SetCompositionMode().

The description give a short equation on how the values of a resulting pixel are calculated. *R* = Result, *S* = Source, *D* = Destination, colors premultiplied with alpha *Ra*, *Sa*, *Da* their alpha components

Enumerator

wxCOMPOSITION_INVALID Indicates invalid or unsupported composition mode. This value can't be passed to wxGraphicsContext::SetCompositionMode().

Since

2.9.2

wxCOMPOSITION_CLEAR *R* = 0

wxCOMPOSITION_SOURCE *R* = S

wxCOMPOSITION_OVER *R* = *S* + *D*∗(1 - *Sa*)

wxCOMPOSITION_IN *R* = *S*∗*Da*

wxCOMPOSITION_OUT *R* = *S*∗(1 - *Da*)

wxCOMPOSITION_ATOP *R* = *S*∗*Da* + *D*∗(1 - *Sa*)

wxCOMPOSITION_DEST *R* = *D*, essentially a noop

wxCOMPOSITION_DEST_OVER *R* = *S*∗(1 - *Da*) + *D*

wxCOMPOSITION_DEST_IN *R* = *D*∗*Sa*

wxCOMPOSITION_DEST_OUT *R* = *D*∗(1 - *Sa*)

wxCOMPOSITION_DEST_ATOP *R* = *S*∗(1 - *Da*) + *D*∗*Sa*

wxCOMPOSITION_XOR *R* = *S*∗(1 - *Da*) + *D*∗(1 - *Sa*)

wxCOMPOSITION_ADD *R* = *S* + *D*

enum wxInterpolationQuality

Interpolation quality used by wxGraphicsContext::SetInterpolationQuality().

Enumerator

wxINTERPOLATION_DEFAULT default interpolation, based on type of context, in general medium quality

wxINTERPOLATION_NONE no interpolation

wxINTERPOLATION_FAST fast interpolation, suited for interactivity

wxINTERPOLATION_GOOD better quality

wxINTERPOLATION_BEST best quality, not suited for interactivity

22.295.2 Variable Documentation

const wxGraphicsBitmap wxNullGraphicsBitmap

const wxGraphicsBrush wxNullGraphicsBrush

const wxGraphicsFont wxNullGraphicsFont

const wxGraphicsMatrix wxNullGraphicsMatrix

const wxGraphicsPath wxNullGraphicsPath

const wxGraphicsPen wxNullGraphicsPen

22.296 interface/wx/hash.h File Reference

Classes

- class wxHashTable

22.297 interface/wx/hashmap.h File Reference

Classes

- class wxHashMap

  This is a simple, type-safe, and reasonably efficient hash map class, whose interface is a subset of the interface of STL containers.

22.298 interface/wx/hashset.h File Reference

Classes

- class wxHashSet

  This is a simple, type-safe, and reasonably efficient hash set class, whose interface is a subset of the interface of STL containers.

22.299 interface/wx/headercol.h File Reference

Classes

- class wxHeaderColumn

  Represents a column header in controls displaying tabular data such as wxDataViewCtrl or wxGrid.
- class wxSettableHeaderColumn

  Adds methods to set the column attributes to wxHeaderColumn.
- class wxHeaderColumnSimple

  Simple container for the information about the column.

Enumerations

- enum {
  wxCOL_WIDTH_DEFAULT = -1,
  wxCOL_WIDTH_AUTOSIZE = -2 }

  Column width special values.
- enum {
  wxCOL_RESIZABLE = 1,
  wxCOL_SORTABLE = 2,
  wxCOL_REORDERABLE = 4,
  wxCOL_HIDDEN = 8,
  wxCOL_DEFAULT_FLAGS = wxCOL_RESIZABLE | wxCOL_REORDERABLE }

  Bit flags used as wxHeaderColumn flags.

22.299.1 Enumeration Type Documentation

anonymous enum

Column width special values.

Enumerator

wxCOL_WIDTH_DEFAULT Special value used for column width meaning unspecified or default.

wxCOL_WIDTH_AUTOSIZE Size the column automatically to fit all values.

Note

On OS X, this style is only implemented in the Cocoa build on OS X >= 10.5; it behaves identically to wxCOL_WIDTH_DEFAULT otherwise.

anonymous enum

Bit flags used as wxHeaderColumn flags.

Enumerator

wxCOL_RESIZABLE Column can be resized (included in default flags).

wxCOL_SORTABLE Column can be clicked to toggle the sort order by its contents.

wxCOL_REORDERABLE Column can be dragged to change its order (included in default).

wxCOL_HIDDEN Column is not shown at all.

wxCOL_DEFAULT_FLAGS Default flags for wxHeaderColumn ctor.

22.300 interface/wx/headerctrl.h File Reference

Classes

- class wxHeaderCtrl

  wxHeaderCtrl is the control containing the column headings which is usually used for display of tabular data.
- class wxHeaderCtrlSimple

  wxHeaderCtrlSimple is a concrete header control which can be used directly, without inheriting from it as you need to do when using wxHeaderCtrl itself.
- class wxHeaderCtrlEvent

  Event class representing the events generated by wxHeaderCtrl.

Enumerations

- enum {
  wxHD_ALLOW_REORDER = 0x0001,
  wxHD_ALLOW_HIDE = 0x0002,
  wxHD_DEFAULT_STYLE = wxHD_ALLOW_REORDER }

Variables

- wxEventType wxEVT_HEADER_CLICK
- wxEventType wxEVT_HEADER_RIGHT_CLICK
- wxEventType wxEVT_HEADER_MIDDLE_CLICK
- wxEventType wxEVT_HEADER_DCLICK
- wxEventType wxEVT_HEADER_RIGHT_DCLICK
- wxEventType wxEVT_HEADER_MIDDLE_DCLICK
- wxEventType wxEVT_HEADER_SEPARATOR_DCLICK
- wxEventType wxEVT_HEADER_BEGIN_RESIZE
- wxEventType wxEVT_HEADER_RESIZING
- wxEventType wxEVT_HEADER_END_RESIZE
- wxEventType wxEVT_HEADER_BEGIN_REORDER
- wxEventType wxEVT_HEADER_END_REORDER
- wxEventType wxEVT_HEADER_DRAGGING_CANCELLED

22.300.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxHD_ALLOW_REORDER

wxHD_ALLOW_HIDE

wxHD_DEFAULT_STYLE

22.300.2 Variable Documentation

wxEventType wxEVT_HEADER_BEGIN_REORDER

wxEventType wxEVT_HEADER_BEGIN_RESIZE

wxEventType wxEVT_HEADER_CLICK

wxEventType wxEVT_HEADER_DCLICK

wxEventType wxEVT_HEADER_DRAGGING_CANCELLED

wxEventType wxEVT_HEADER_END_REORDER

wxEventType wxEVT_HEADER_END_RESIZE

wxEventType wxEVT_HEADER_MIDDLE_CLICK

wxEventType wxEVT_HEADER_MIDDLE_DCLICK

wxEventType wxEVT_HEADER_RESIZING

wxEventType wxEVT_HEADER_RIGHT_CLICK

wxEventType wxEVT_HEADER_RIGHT_DCLICK

wxEventType wxEVT_HEADER_SEPARATOR_DCLICK

22.301 interface/wx/help.h File Reference

Classes

- class wxHelpControllerBase

  This is the abstract base class a family of classes by which applications may invoke a help viewer to provide on-line help.
- class wxHelpController

  This is an alias for one of a family of help controller classes which is most appropriate for the current platform.

Macros

- #define wxHELP_NETSCAPE 1

Enumerations

- enum wxHelpSearchMode {
  wxHELP_SEARCH_INDEX,
  wxHELP_SEARCH_ALL }

  Help search modes for wxHelpController::KeywordSearch().

22.301.1 Macro Definition Documentation

#define wxHELP_NETSCAPE 1

22.301.2 Enumeration Type Documentation

enum wxHelpSearchMode

Help search modes for wxHelpController::KeywordSearch().

Enumerator

wxHELP_SEARCH_INDEX Search the index only.

wxHELP_SEARCH_ALL Search all entries.

22.302 interface/wx/html/helpctrl.h File Reference

Classes

- class wxHtmlHelpController

  This help controller provides an easy way of displaying HTML help in your application (see HTML Sample, test example).
- class wxHtmlModalHelp

  This class uses wxHtmlHelpController to display help in a modal dialog.

Macros

- #define wxID_HTML_HELPFRAME (wxID_HIGHEST + 1)
- #define wxHF_EMBEDDED 0x00008000

  This style indicates that the window is embedded in the application and must not be destroyed by the help controller.
- #define wxHF_DIALOG 0x00010000

  Create a dialog for the help window.
- #define wxHF_FRAME 0x00020000

  Create a frame for the help window.
- #define wxHF_MODAL 0x00040000

  Make the dialog modal when displaying help.

22.302.1 Macro Definition Documentation

#define wxHF_DIALOG 0x00010000

Create a dialog for the help window.

#define wxHF_EMBEDDED 0x00008000

This style indicates that the window is embedded in the application and must not be destroyed by the help controller.

#define wxHF_FRAME 0x00020000

Create a frame for the help window.

#define wxHF_MODAL 0x00040000

Make the dialog modal when displaying help.

#define wxID_HTML_HELPFRAME (wxID_HIGHEST + 1)

22.303 interface/wx/html/helpdata.h File Reference

Classes

- class wxHtmlBookRecord

  Helper class for wxHtmlHelpData.
- class wxHtmlHelpDataItem

  Helper class for wxHtmlHelpData.
- class wxHtmlHelpData

  This class is used by wxHtmlHelpController and wxHtmlHelpFrame to access HTML help items.

22.304 interface/wx/html/helpdlg.h File Reference

Classes

- class wxHtmlHelpDialog

  This class is used by wxHtmlHelpController to display help.

22.305 interface/wx/html/helpfrm.h File Reference

Classes

- class wxHtmlHelpFrame

  This class is used by wxHtmlHelpController to display help.

Macros

- #define wxHF_TOOLBAR 0x0001

  style flags for the Help Frame
- #define wxHF_CONTENTS 0x0002
- #define wxHF_INDEX 0x0004
- #define wxHF_SEARCH 0x0008
- #define wxHF_BOOKMARKS 0x0010
- #define wxHF_OPEN_FILES 0x0020
- #define wxHF_PRINT 0x0040
- #define wxHF_FLAT_TOOLBAR 0x0080
- #define wxHF_MERGE_BOOKS 0x0100
- #define wxHF_ICONS_BOOK 0x0200
- #define wxHF_ICONS_BOOK_CHAPTER 0x0400
- #define wxHF_ICONS_FOLDER 0x0000
- #define wxHF_DEFAULT_STYLE

22.305.1 Macro Definition Documentation

#define wxHF_BOOKMARKS 0x0010

#define wxHF_CONTENTS 0x0002

#define wxHF_DEFAULT_STYLE

Value:

```
(wxHF_TOOLBAR | wxHF_CONTENTS | \
                                wxHF_INDEX |
      wxHF_SEARCH | \
                                wxHF_BOOKMARKS |
      wxHF_PRINT)
```

#define wxHF_FLAT_TOOLBAR 0x0080

#define wxHF_ICONS_BOOK 0x0200

#define wxHF_ICONS_BOOK_CHAPTER 0x0400

#define wxHF_ICONS_FOLDER 0x0000

#define wxHF_INDEX 0x0004

#define wxHF_MERGE_BOOKS 0x0100

#define wxHF_OPEN_FILES 0x0020

#define wxHF_PRINT 0x0040

#define wxHF_SEARCH 0x0008

#define wxHF_TOOLBAR 0x0001

style flags for the Help Frame

22.306 interface/wx/html/helpwnd.h File Reference

Classes

- class wxHtmlHelpWindow

  This class is used by wxHtmlHelpController to display help within a frame or dialog, but you can use it yourself to create an embedded HTML help window.

Enumerations

- enum {
  wxID_HTML_PANEL = wxID_HIGHEST + 10,
  wxID_HTML_BACK,
  wxID_HTML_FORWARD,
  wxID_HTML_UPNODE,
  wxID_HTML_UP,
  wxID_HTML_DOWN,
  wxID_HTML_PRINT,
  wxID_HTML_OPENFILE,
  wxID_HTML_OPTIONS,
  wxID_HTML_BOOKMARKSLIST,
  wxID_HTML_BOOKMARKSADD,
  wxID_HTML_BOOKMARKSREMOVE,
  wxID_HTML_TREECTRL,
  wxID_HTML_INDEXPAGE,
  wxID_HTML_INDEXLIST,
  wxID_HTML_INDEXTEXT,
  wxID_HTML_INDEXBUTTON,
  wxID_HTML_INDEXBUTTONALL,
  wxID_HTML_NOTEBOOK,
  wxID_HTML_SEARCHPAGE,
  wxID_HTML_SEARCHTEXT,
  wxID_HTML_SEARCHLIST,
  wxID_HTML_SEARCHBUTTON,
  wxID_HTML_SEARCHCHOICE,
  wxID_HTML_COUNTINFO }

22.306.1 Enumeration Type Documentation

anonymous enum

Command IDs

Enumerator

wxID_HTML_PANEL

wxID_HTML_BACK

wxID_HTML_FORWARD

wxID_HTML_UPNODE

wxID_HTML_UP

wxID_HTML_DOWN

wxID_HTML_PRINT

wxID_HTML_OPENFILE

wxID_HTML_OPTIONS

wxID_HTML_BOOKMARKSLIST

wxID_HTML_BOOKMARKSADD

wxID_HTML_BOOKMARKSREMOVE

wxID_HTML_TREECTRL

wxID_HTML_INDEXPAGE

wxID_HTML_INDEXLIST

wxID_HTML_INDEXTEXT

wxID_HTML_INDEXBUTTON

wxID_HTML_INDEXBUTTONALL

wxID_HTML_NOTEBOOK

wxID_HTML_SEARCHPAGE

wxID_HTML_SEARCHTEXT

wxID_HTML_SEARCHLIST

wxID_HTML_SEARCHBUTTON

wxID_HTML_SEARCHCHOICE

wxID_HTML_COUNTINFO

22.307 interface/wx/html/htmlcell.h File Reference

Classes

- class wxHtmlSelection
- class wxHtmlRenderingState

  Selection state is passed to wxHtmlCell::Draw so that it can render itself differently e.g.
- class wxHtmlRenderingStyle

  wxHtmlSelection is data holder with information about text selection.
- class wxHtmlRenderingInfo

  This class contains information given to cells when drawing them.
- class wxHtmlCell

  Internal data structure.
- class wxHtmlContainerCell

  The wxHtmlContainerCell class is an implementation of a cell that may contain more cells in it.
- class wxHtmlLinkInfo

  This class stores all necessary information about hypertext links (as represented by <A> tag in HTML documents).
- class wxHtmlColourCell

  This cell changes the colour of either the background or the foreground.
- class wxHtmlWidgetCell

  wxHtmlWidgetCell is a class that provides a connection between HTML cells and widgets (an object derived from wxWindow).
- class wxHtmlWordCell

This html cell represents a single word or text fragment in the document stream.

- class wxHtmlWordWithTabsCell

  wxHtmlWordCell is a specialization for storing text fragments with embedded tab characters.

- class wxHtmlFontCell

  This cell represents a font change in the document stream.

Enumerations

- enum wxHtmlSelectionState {
  wxHTML_SEL_OUT,
  wxHTML_SEL_IN,
  wxHTML_SEL_CHANGING }
- enum {
  wxHTML_FIND_EXACT = 1,
  wxHTML_FIND_NEAREST_BEFORE = 2,
  wxHTML_FIND_NEAREST_AFTER = 4 }
- enum wxHtmlScriptMode {
  wxHTML_SCRIPT_NORMAL,
  wxHTML_SCRIPT_SUB,
  wxHTML_SCRIPT_SUP }

22.307.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxHTML_FIND_EXACT

wxHTML_FIND_NEAREST_BEFORE

wxHTML_FIND_NEAREST_AFTER

enum wxHtmlScriptMode

Enumerator

wxHTML_SCRIPT_NORMAL

wxHTML_SCRIPT_SUB

wxHTML_SCRIPT_SUP

enum wxHtmlSelectionState

Enumerator

wxHTML_SEL_OUT

wxHTML_SEL_IN

wxHTML_SEL_CHANGING

22.308 interface/wx/html/htmldefs.h File Reference

Macros

- #define wxHTML_ALIGN_LEFT 0x0000

- #define wxHTML_ALIGN_RIGHT 0x0002
- #define wxHTML_ALIGN_JUSTIFY 0x0010
- #define wxHTML_ALIGN_TOP 0x0004
- #define wxHTML_ALIGN_BOTTOM 0x0008
- #define wxHTML_ALIGN_CENTER 0x0001
- #define wxHTML_CLR_FOREGROUND 0x0001
- #define wxHTML_CLR_BACKGROUND 0x0002
- #define wxHTML_CLR_TRANSPARENT_BACKGROUND 0x0004
- #define wxHTML_UNITS_PIXELS 0x0001
- #define wxHTML_UNITS_PERCENT 0x0002
- #define wxHTML_INDENT_LEFT 0x0010
- #define wxHTML_INDENT_RIGHT 0x0020
- #define wxHTML_INDENT_TOP 0x0040
- #define wxHTML_INDENT_BOTTOM 0x0080
- #define wxHTML_INDENT_HORIZONTAL (wxHTML_INDENT_LEFT | wxHTML_INDENT_RIGHT)
- #define wxHTML_INDENT_VERTICAL (wxHTML_INDENT_TOP | wxHTML_INDENT_BOTTOM)
- #define wxHTML_INDENT_ALL (wxHTML_INDENT_VERTICAL | wxHTML_INDENT_HORIZONTAL)
- #define wxHTML_COND_ISANCHOR 1
- #define wxHTML_COND_ISIMAGEMAP 2
- #define wxHTML_COND_USER 10000

22.308.1 Macro Definition Documentation

#define wxHTML_ALIGN_BOTTOM 0x0008

#define wxHTML_ALIGN_CENTER 0x0001

#define wxHTML_ALIGN_JUSTIFY 0x0010

#define wxHTML_ALIGN_LEFT 0x0000

#define wxHTML_ALIGN_RIGHT 0x0002

#define wxHTML_ALIGN_TOP 0x0004

#define wxHTML_CLR_BACKGROUND 0x0002

#define wxHTML_CLR_FOREGROUND 0x0001

#define wxHTML_CLR_TRANSPARENT_BACKGROUND 0x0004

#define wxHTML_COND_ISANCHOR 1

#define wxHTML_COND_ISIMAGEMAP 2

#define wxHTML_COND_USER 10000

#define wxHTML_INDENT_ALL (wxHTML_INDENT_VERTICAL | wxHTML_INDENT_HORIZONTAL)

#define wxHTML_INDENT_BOTTOM 0x0080

#define wxHTML_INDENT_HORIZONTAL (wxHTML_INDENT_LEFT | wxHTML_INDENT_RIGHT)

#define wxHTML_INDENT_LEFT 0x0010

#define wxHTML_INDENT_RIGHT 0x0020

#define wxHTML_INDENT_TOP 0x0040

#define wxHTML_INDENT_VERTICAL (wxHTML_INDENT_TOP | wxHTML_INDENT_BOTTOM)

#define wxHTML_UNITS_PERCENT 0x0002

#define wxHTML_UNITS_PIXELS 0x0001

22.309 interface/wx/html/htmlfilt.h File Reference

Classes

- class wxHtmlFilter

  This class is the parent class of input filters for wxHtmlWindow.

22.310 interface/wx/html/htmlpars.h File Reference

Classes

- class wxHtmlTagHandler
- class wxHtmlParser

  *Classes derived from this handle the **generic** parsing of HTML documents: it scans the document and divide it into blocks of tags (where one block consists of beginning and ending tag and of text between these two tags).*

Enumerations

- enum wxHtmlURLType {
  wxHTML_URL_PAGE,
  wxHTML_URL_IMAGE,
  wxHTML_URL_OTHER }

22.310.1 Enumeration Type Documentation

enum **wxHtmlURLType**

Enumerator

wxHTML_URL_PAGE

wxHTML_URL_IMAGE

wxHTML_URL_OTHER

22.311 interface/wx/html/htmltag.h File Reference

Classes

- class wxHtmlTag

  This class represents a single HTML tag.

22.312 interface/wx/html/htmlwin.h File Reference

Classes

- class wxHtmlWindowInterface

  Abstract interface to a HTML rendering window (such as wxHtmlWindow or wxHtmlListBox) that is passed to wxHtmlWinParser.
- class wxHtmlWindow

  wxHtmlWindow is probably the only class you will directly use unless you want to do something special (like adding new tag handlers or MIME filters).
- class wxHtmlLinkEvent

  This event class is used for the events generated by wxHtmlWindow.
- class wxHtmlCellEvent

  This event class is used for the events generated by wxHtmlWindow.

Macros

- #define wxHW_SCROLLBAR_NEVER 0x0002
- #define wxHW_SCROLLBAR_AUTO 0x0004
- #define wxHW_NO_SELECTION 0x0008
- #define wxHW_DEFAULT_STYLE wxHW_SCROLLBAR_AUTO

Enumerations

- enum wxHtmlOpeningStatus {
  wxHTML_OPEN,
  wxHTML_BLOCK,
  wxHTML_REDIRECT }

  Enum for wxHtmlWindow::OnOpeningURL and wxHtmlWindowInterface::OnOpeningURL.

Variables

- wxEventType wxEVT_HTML_CELL_CLICKED
- wxEventType wxEVT_HTML_CELL_HOVER
- wxEventType wxEVT_HTML_LINK_CLICKED

22.312.1 Macro Definition Documentation

#define wxHW_DEFAULT_STYLE wxHW_SCROLLBAR_AUTO

#define wxHW_NO_SELECTION 0x0008

#define wxHW_SCROLLBAR_AUTO 0x0004

#define wxHW_SCROLLBAR_NEVER 0x0002

22.312.2 Enumeration Type Documentation

enum wxHtmlOpeningStatus

Enum for wxHtmlWindow::OnOpeningURL and wxHtmlWindowInterface::OnOpeningURL.

Enumerator

wxHTML_OPEN Open the requested URL.

wxHTML_BLOCK Do not open the URL.

wxHTML_REDIRECT Redirect to another URL (returned from OnOpeningURL)

22.312.3 Variable Documentation

wxEventType wxEVT_HTML_CELL_CLICKED

wxEventType wxEVT_HTML_CELL_HOVER

wxEventType wxEVT_HTML_LINK_CLICKED

22.313 interface/wx/html/htmprint.h File Reference

Classes

- class wxHtmlDCRenderer

  This class can render HTML document into a specified area of a DC.
- class wxHtmlEasyPrinting

  This class provides very simple interface to printing architecture.
- class wxHtmlPrintout

  This class serves as printout class for HTML documents.

Enumerations

- enum {
  wxPAGE_ODD,
  wxPAGE_EVEN,
  wxPAGE_ALL }

22.313.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxPAGE_ODD

wxPAGE_EVEN

wxPAGE_ALL

22.314 interface/wx/html/webkit.h File Reference

Classes

- class wxWebKitCtrl

  This control is a native wrapper around the Safari web browsing engine.
- class wxWebKitBeforeLoadEvent
- class wxWebKitStateChangedEvent
- class wxWebKitNewWindowEvent

Enumerations

- enum {
  wxWEBKIT_STATE_START = 1,
  wxWEBKIT_STATE_NEGOTIATING = 2,
  wxWEBKIT_STATE_REDIRECTING = 4,
  wxWEBKIT_STATE_TRANSFERRING = 8,
  wxWEBKIT_STATE_STOP = 16,
  wxWEBKIT_STATE_FAILED = 32 }
- enum {
  wxWEBKIT_NAV_LINK_CLICKED = 1,
  wxWEBKIT_NAV_BACK_NEXT = 2,
  wxWEBKIT_NAV_FORM_SUBMITTED = 4,
  wxWEBKIT_NAV_RELOAD = 8,
  wxWEBKIT_NAV_FORM_RESUBMITTED = 16,
  wxWEBKIT_NAV_OTHER = 32 }

Variables

- wxEventType wxEVT_WEBKIT_STATE_CHANGED
- wxEventType wxEVT_WEBKIT_BEFORE_LOAD
- wxEventType wxEVT_WEBKIT_NEW_WINDOW

22.314.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxWEBKIT_STATE_START

wxWEBKIT_STATE_NEGOTIATING

wxWEBKIT_STATE_REDIRECTING

wxWEBKIT_STATE_TRANSFERRING

wxWEBKIT_STATE_STOP

wxWEBKIT_STATE_FAILED

anonymous enum

Enumerator

wxWEBKIT_NAV_LINK_CLICKED

wxWEBKIT_NAV_BACK_NEXT

wxWEBKIT_NAV_FORM_SUBMITTED

wxWEBKIT_NAV_RELOAD

wxWEBKIT_NAV_FORM_RESUBMITTED

wxWEBKIT_NAV_OTHER

22.314.2 Variable Documentation

wxEventType wxEVT_WEBKIT_BEFORE_LOAD

wxEventType wxEVT_WEBKIT_NEW_WINDOW

wxEventType wxEVT_WEBKIT_STATE_CHANGED

22.315 interface/wx/html/winpars.h File Reference

Classes

- class wxHtmlTagsModule

  This class provides easy way of filling wxHtmlWinParser's table of tag handlers.

- class wxHtmlWinTagHandler

  This is basically wxHtmlTagHandler except that it is extended with protected member m_WParser pointing to the wxHtmlWinParser object (value of this member is identical to wxHtmlParser's m_Parser).

- class wxHtmlWinParser

  This class is derived from wxHtmlParser and its main goal is to parse HTML input so that it can be displayed in wxHtmlWindow.

22.316 interface/wx/htmllbox.h File Reference

Classes

- class wxHtmlListBox

  wxHtmlListBox is an implementation of wxVListBox which shows HTML content in the listbox rows.

- class wxSimpleHtmlListBox

  wxSimpleHtmlListBox is an implementation of wxHtmlListBox which shows HTML content in the listbox rows.

22.317 interface/wx/hyperlink.h File Reference

Classes

- class wxHyperlinkEvent

  This event class is used for the events generated by wxHyperlinkCtrl.

- class wxHyperlinkCtrl

  This class shows a static text element which links to an URL.

Macros

- #define wxHL_CONTEXTMENU 0x0001
- #define wxHL_ALIGN_LEFT 0x0002
- #define wxHL_ALIGN_RIGHT 0x0004
- #define wxHL_ALIGN_CENTRE 0x0008
- #define wxHL_DEFAULT_STYLE (wxHL_CONTEXTMENU|wxNO_BORDER|wxHL_ALIGN_CENTRE)

Variables

- wxEventType wxEVT_HYPERLINK

22.317.1 Macro Definition Documentation

#define wxHL_ALIGN_CENTRE 0x0008

#define wxHL_ALIGN_LEFT 0x0002

#define wxHL_ALIGN_RIGHT 0x0004

#define wxHL_CONTEXTMENU 0x0001

#define wxHL_DEFAULT_STYLE (wxHL_CONTEXTMENU|wxNO_BORDER|wxHL_ALIGN_CENTRE)

22.317.2 Variable Documentation

wxEventType wxEVT_HYPERLINK

22.318 interface/wx/icon.h File Reference

Classes

- class wxIcon

  An icon is a small rectangular bitmap usually used for denoting a minimized application.

Macros

- #define wxICON_SCREEN_DEPTH (-1)

  In wxIcon context this value means: "use the screen depth".

Variables

- wxIcon wxNullIcon

  An empty wxIcon.

22.318.1 Macro Definition Documentation

#define wxICON_SCREEN_DEPTH (-1)

In wxIcon context this value means: "use the screen depth".

22.318.2 Variable Documentation

wxIcon wxNullIcon

An empty wxIcon.

22.319 interface/wx/iconbndl.h File Reference

Classes

- class wxIconBundle

  This class contains multiple copies of an icon in different sizes.

Variables

- wxIconBundle wxNullIconBundle

  An empty wxIconBundle.

22.319.1 Variable Documentation

wxIconBundle wxNullIconBundle

An empty wxIconBundle.

22.320 interface/wx/iconloc.h File Reference

Classes

- class wxIconLocation

  wxIconLocation is a tiny class describing the location of an (external, i.e.

22.321 interface/wx/image.h File Reference

Classes

- class wxImageHandler

  This is the base class for implementing image file loading/saving, and image creation from data.
- class wxImage

  This class encapsulates a platform-independent image.
- class wxImage::RGBValue

  A simple class which stores red, green and blue values as 8 bit unsigned integers in the range of 0-255.
- class wxImage::HSVValue

  A simple class which stores hue, saturation and value as doubles in the range 0.0-1.0.
- class wxImageHistogram

Macros

- #define wxIMAGE_OPTION_QUALITY wxString("quality")

  Image option names.
- #define wxIMAGE_OPTION_FILENAME wxString("FileName")
- #define wxIMAGE_OPTION_RESOLUTION wxString("Resolution")
- #define wxIMAGE_OPTION_RESOLUTIONX wxString("ResolutionX")
- #define wxIMAGE_OPTION_RESOLUTIONY wxString("ResolutionY")
- #define wxIMAGE_OPTION_RESOLUTIONUNIT wxString("ResolutionUnit")
- #define wxIMAGE_OPTION_MAX_WIDTH wxString("MaxWidth")
- #define wxIMAGE_OPTION_MAX_HEIGHT wxString("MaxHeight")
- #define wxIMAGE_OPTION_ORIGINAL_WIDTH wxString("OriginalWidth")
- #define wxIMAGE_OPTION_ORIGINAL_HEIGHT wxString("OriginalHeight")
- #define wxIMAGE_OPTION_BMP_FORMAT wxString("wxBMP_FORMAT")
- #define wxIMAGE_OPTION_CUR_HOTSPOT_X wxString("HotSpotX")
- #define wxIMAGE_OPTION_CUR_HOTSPOT_Y wxString("HotSpotY")
- #define wxIMAGE_OPTION_GIF_COMMENT wxString("GifComment")
- #define wxIMAGE_OPTION_PNG_FORMAT wxString("PngFormat")

- #define wxIMAGE_OPTION_PNG_BITDEPTH wxString("PngBitDepth")
- #define wxIMAGE_OPTION_PNG_FILTER wxString("PngF")
- #define wxIMAGE_OPTION_PNG_COMPRESSION_LEVEL wxString("PngZL")
- #define wxIMAGE_OPTION_PNG_COMPRESSION_MEM_LEVEL wxString("PngZM")
- #define wxIMAGE_OPTION_PNG_COMPRESSION_STRATEGY wxString("PngZS")
- #define wxIMAGE_OPTION_PNG_COMPRESSION_BUFFER_SIZE wxString("PngZB")
- #define wxIMAGE_OPTION_TIFF_BITSPERSAMPLE wxString("BitsPerSample")
- #define wxIMAGE_OPTION_TIFF_SAMPLESPERPIXEL wxString("SamplesPerPixel")
- #define wxIMAGE_OPTION_TIFF_COMPRESSION wxString("Compression")
- #define wxIMAGE_OPTION_TIFF_PHOTOMETRIC wxString("Photometric")
- #define wxIMAGE_OPTION_TIFF_IMAGEDESCRIPTOR wxString("ImageDescriptor")

Enumerations

- enum wxImageResolution {
  wxIMAGE_RESOLUTION_NONE = 0,
  wxIMAGE_RESOLUTION_INCHES = 1,
  wxIMAGE_RESOLUTION_CM = 2 }

  Possible values for the image resolution option.
- enum wxImageResizeQuality {
  wxIMAGE_QUALITY_NEAREST,
  wxIMAGE_QUALITY_BILINEAR,
  wxIMAGE_QUALITY_BICUBIC,
  wxIMAGE_QUALITY_BOX_AVERAGE,
  wxIMAGE_QUALITY_NORMAL,
  wxIMAGE_QUALITY_HIGH }

  Image resize algorithm.
- enum wxImagePNGType {
  wxPNG_TYPE_COLOUR = 0,
  wxPNG_TYPE_GREY = 2,
  wxPNG_TYPE_GREY_RED = 3,
  wxPNG_TYPE_PALETTE = 4 }

  Possible values for PNG image type option.
- enum {
  wxBMP_24BPP = 24,
  wxBMP_8BPP = 8,
  wxBMP_8BPP_GREY = 9,
  wxBMP_8BPP_GRAY = wxBMP_8BPP_GREY,
  wxBMP_8BPP_RED = 10,
  wxBMP_8BPP_PALETTE = 11,
  wxBMP_4BPP = 4,
  wxBMP_1BPP = 1,
  wxBMP_1BPP_BW = 2 }

Functions

- void wxInitAllImageHandlers ()

  Initializes all available image handlers.

Variables

- const unsigned char wxIMAGE_ALPHA_TRANSPARENT = 0

  Constant used to indicate the alpha value conventionally defined as the complete transparency.
- const unsigned char wxIMAGE_ALPHA_OPAQUE = 0xff

Constant used to indicate the alpha value conventionally defined as the complete opacity.

- const unsigned char wxIMAGE_ALPHA_THRESHOLD = 0x80
- wxImage wxNullImage

  An instance of an empty image without an alpha channel.

22.321.1 Macro Definition Documentation

#define wxIMAGE_OPTION_BMP_FORMAT wxString("wxBMP_FORMAT")

#define wxIMAGE_OPTION_CUR_HOTSPOT_X wxString("HotSpotX")

#define wxIMAGE_OPTION_CUR_HOTSPOT_Y wxString("HotSpotY")

#define wxIMAGE_OPTION_FILENAME wxString("FileName")

#define wxIMAGE_OPTION_GIF_COMMENT wxString("GifComment")

#define wxIMAGE_OPTION_MAX_HEIGHT wxString("MaxHeight")

#define wxIMAGE_OPTION_MAX_WIDTH wxString("MaxWidth")

#define wxIMAGE_OPTION_ORIGINAL_HEIGHT wxString("OriginalHeight")

#define wxIMAGE_OPTION_ORIGINAL_WIDTH wxString("OriginalWidth")

#define wxIMAGE_OPTION_PNG_BITDEPTH wxString("PngBitDepth")

#define wxIMAGE_OPTION_PNG_COMPRESSION_BUFFER_SIZE wxString("PngZB")

#define wxIMAGE_OPTION_PNG_COMPRESSION_LEVEL wxString("PngZL")

#define wxIMAGE_OPTION_PNG_COMPRESSION_MEM_LEVEL wxString("PngZM")

#define wxIMAGE_OPTION_PNG_COMPRESSION_STRATEGY wxString("PngZS")

#define wxIMAGE_OPTION_PNG_FILTER wxString("PngF")

#define wxIMAGE_OPTION_PNG_FORMAT wxString("PngFormat")

#define wxIMAGE_OPTION_QUALITY wxString("quality")

Image option names.

#define wxIMAGE_OPTION_RESOLUTION wxString("Resolution")

#define wxIMAGE_OPTION_RESOLUTIONUNIT wxString("ResolutionUnit")

#define wxIMAGE_OPTION_RESOLUTIONX wxString("ResolutionX")

#define wxIMAGE_OPTION_RESOLUTIONY wxString("ResolutionY")

#define wxIMAGE_OPTION_TIFF_BITSPERSAMPLE wxString("BitsPerSample")

#define wxIMAGE_OPTION_TIFF_COMPRESSION wxString("Compression")

#define wxIMAGE_OPTION_TIFF_IMAGEDESCRIPTOR wxString("ImageDescriptor")

#define wxIMAGE_OPTION_TIFF_PHOTOMETRIC wxString("Photometric")

#define wxIMAGE_OPTION_TIFF_SAMPLESPERPIXEL wxString("SamplesPerPixel")

22.321.2 Enumeration Type Documentation

anonymous enum

Enumerator

wxBMP_24BPP
wxBMP_8BPP
wxBMP_8BPP_GREY
wxBMP_8BPP_GRAY
wxBMP_8BPP_RED
wxBMP_8BPP_PALETTE
wxBMP_4BPP
wxBMP_1BPP
wxBMP_1BPP_BW

enum wxImagePNGType

Possible values for PNG image type option.

See also

wxImage::GetOptionInt().

Enumerator

wxPNG_TYPE_COLOUR Colour PNG image.
wxPNG_TYPE_GREY Greyscale PNG image converted from RGB.
wxPNG_TYPE_GREY_RED Greyscale PNG image using red as grey.
wxPNG_TYPE_PALETTE Palette encoding.

enum wxImageResizeQuality

Image resize algorithm.

This is used with wxImage::Scale() and wxImage::Rescale().

Enumerator

wxIMAGE_QUALITY_NEAREST Simplest and fastest algorithm.
wxIMAGE_QUALITY_BILINEAR Compromise between wxIMAGE_QUALITY_NEAREST and wxIMAGE_↩ QUALITY_BICUBIC.
wxIMAGE_QUALITY_BICUBIC Highest quality but slowest execution time.
wxIMAGE_QUALITY_BOX_AVERAGE Use surrounding pixels to calculate an average that will be used for new pixels. This method is typically used when reducing the size of an image.
wxIMAGE_QUALITY_NORMAL Default image resizing algorithm used by wxImage::Scale(). Currently the same as wxIMAGE_QUALITY_NEAREST.
wxIMAGE_QUALITY_HIGH Best image resizing algorithm. Since version 2.9.2 this results in wxIMAGE_Q↩ UALITY_BOX_AVERAGE being used when reducing the size of the image (meaning that both the new width and height will be smaller than the original size). Otherwise wxIMAGE_QUALITY_BICUBIC is used.

enum wxImageResolution

Possible values for the image resolution option.

See also

wxImage::GetOptionInt().

Enumerator

wxIMAGE_RESOLUTION_NONE Resolution not specified.

wxIMAGE_RESOLUTION_INCHES Resolution specified in inches.

wxIMAGE_RESOLUTION_CM Resolution specified in centimetres.

22.321.3 Variable Documentation

const unsigned char wxIMAGE_ALPHA_OPAQUE = 0xff

Constant used to indicate the alpha value conventionally defined as the complete opacity.

const unsigned char wxIMAGE_ALPHA_THRESHOLD = 0x80

const unsigned char wxIMAGE_ALPHA_TRANSPARENT = 0

Constant used to indicate the alpha value conventionally defined as the complete transparency.

wxImage wxNullImage

An instance of an empty image without an alpha channel.

22.322 interface/wx/imaglist.h File Reference

Classes

- class wxImageList

  A wxImageList contains a list of images, which are stored in an unspecified form.

Macros

- #define wxIMAGELIST_DRAW_NORMAL 0x0001

  Flags for Draw.
- #define wxIMAGELIST_DRAW_TRANSPARENT 0x0002
- #define wxIMAGELIST_DRAW_SELECTED 0x0004
- #define wxIMAGELIST_DRAW_FOCUSED 0x0008

Enumerations

- enum {
  wxIMAGE_LIST_NORMAL,
  wxIMAGE_LIST_SMALL,
  wxIMAGE_LIST_STATE }

  Flag values for Set/GetImageList.

22.322.1 Macro Definition Documentation

#define wxIMAGELIST_DRAW_FOCUSED 0x0008

#define wxIMAGELIST_DRAW_NORMAL 0x0001

Flags for Draw.

#define wxIMAGELIST_DRAW_SELECTED 0x0004

#define wxIMAGELIST_DRAW_TRANSPARENT 0x0002

22.322.2 Enumeration Type Documentation

anonymous enum

Flag values for Set/GetImageList.

Enumerator

wxIMAGE_LIST_NORMAL

wxIMAGE_LIST_SMALL

wxIMAGE_LIST_STATE

22.323 interface/wx/infobar.h File Reference

Classes

- class wxInfoBar

  An info bar is a transient window shown at top or bottom of its parent window to display non-critical information to the user.

22.324 interface/wx/init.h File Reference

Classes

- class wxInitializer

  Create an object of this class on the stack to initialize/cleanup the library automatically.

Functions

- bool wxEntryStart (int &argc, wxChar ∗∗argv)

  This function can be used to perform the initialization of wxWidgets if you can't use the default initialization code for any reason.
- bool wxEntryStart (HINSTANCE hInstance, HINSTANCE hPrevInstance=NULL, char ∗pCmdLine=NULL, int nCmdShow=SW_SHOWNORMAL)

  See wxEntryStart(int&,wxChar∗∗) for more info about this function.
- void wxEntryCleanup ()

  Free resources allocated by a successful call to wxEntryStart().
- bool wxInitialize (int argc=0, wxChar ∗∗argv=NULL)

Initialize the library (may be called as many times as needed, but each call to wxInitialize() must be matched by wxUninitialize()).

- void wxUninitialize ()

  This function is for use in console (wxBase) programs only.

22.325 interface/wx/intl.h File Reference

Classes

- struct wxLanguageInfo

  Encapsulates a wxLanguage identifier together with OS-specific information related to that language.

- class wxLocale

  wxLocale class encapsulates all language-dependent settings and is a generalization of the C locale concept.

Enumerations

- enum wxLayoutDirection {
  wxLayout_Default,
  wxLayout_LeftToRight,
  wxLayout_RightToLeft }

  This is the layout direction stored in wxLanguageInfo and returned by wxApp::GetLayoutDirection(), wxWindow::GetLayoutDirection(), wxDC::GetLayoutDirection() for RTL (right-to-left) languages support.

- enum wxLocaleCategory {
  wxLOCALE_CAT_NUMBER,
  wxLOCALE_CAT_DATE,
  wxLOCALE_CAT_MONEY,
  wxLOCALE_CAT_DEFAULT }

  The category of locale settings.

- enum wxLocaleInfo {
  wxLOCALE_THOUSANDS_SEP,
  wxLOCALE_DECIMAL_POINT,
  wxLOCALE_SHORT_DATE_FMT,
  wxLOCALE_LONG_DATE_FMT,
  wxLOCALE_DATE_TIME_FMT,
  wxLOCALE_TIME_FMT }

  The values understood by wxLocale::GetInfo().

22.325.1 Enumeration Type Documentation

enum wxLayoutDirection

This is the layout direction stored in wxLanguageInfo and returned by wxApp::GetLayoutDirection(), wxWindow::GetLayoutDirection(), wxDC::GetLayoutDirection() for RTL (right-to-left) languages support.

Enumerator

wxLayout_Default

wxLayout_LeftToRight

wxLayout_RightToLeft

enum wxLocaleCategory

The category of locale settings.

See also

wxLocale::GetInfo()

Enumerator

wxLOCALE_CAT_NUMBER Number formatting.

wxLOCALE_CAT_DATE Date/time formatting.

wxLOCALE_CAT_MONEY Monetary values formatting.

wxLOCALE_CAT_DEFAULT Default category for the wxLocaleInfo value. This category can be used for values which only make sense for a single category, e.g. wxLOCALE_SHORT_DATE_FMT which can only be used with wxLOCALE_CAT_DATE. As this is the default value of the second parameter of wxLocale::GetInfo(), wxLOCALE_CAT_DATE can be omitted when asking for wxLOCALE_SHORT_DATE_FMT value.

Since

2.9.0

enum wxLocaleInfo

The values understood by wxLocale::GetInfo().

Note that for the `wxLOCALE_*_FMT` constants (the date and time formats), the strings returned by wxLocale::GetInfo() use strftime() or, equivalently, wxDateTime::Format() format. If the relevant format couldn't be determined, an empty string is returned – there is no fallback value so that the application could determine the best course of actions itself in such case.

All of these values are used with `wxLOCALE_CAT_DATE` in wxLocale::GetInfo() or, more typically, with `wxLOCALE_CAT_DEFAULT` as they only apply to a single category.

Enumerator

wxLOCALE_THOUSANDS_SEP The thousands separator. This value can be used with either wxLOCALE_CAT_NUMBER or wxLOCALE_CAT_MONEY categories.

wxLOCALE_DECIMAL_POINT The character used as decimal point. This value can be used with either wxLOCALE_CAT_NUMBER or wxLOCALE_CAT_MONEY categories.

wxLOCALE_SHORT_DATE_FMT Short date format. Notice that short and long date formats may be the same under POSIX systems currently but may, and typically are, different under MSW or OS X.

Since

2.9.0

wxLOCALE_LONG_DATE_FMT Long date format.

Since

2.9.0

wxLOCALE_DATE_TIME_FMT Date and time format.

Since

2.9.0

wxLOCALE_TIME_FMT Time format.

Since

2.9.0

22.326 interface/wx/ipcbase.h File Reference

Classes

- class wxConnectionBase

Enumerations

- enum wxIPCFormat {
  wxIPC_INVALID = 0,
  wxIPC_TEXT = 1,
  wxIPC_BITMAP = 2,
  wxIPC_METAFILE = 3,
  wxIPC_SYLK = 4,
  wxIPC_DIF = 5,
  wxIPC_TIFF = 6,
  wxIPC_OEMTEXT = 7,
  wxIPC_DIB = 8,
  wxIPC_PALETTE = 9,
  wxIPC_PENDATA = 10,
  wxIPC_RIFF = 11,
  wxIPC_WAVE = 12,
  wxIPC_UTF16TEXT = 13,
  wxIPC_ENHMETAFILE = 14,
  wxIPC_FILENAME = 15,
  wxIPC_LOCALE = 16,
  wxIPC_UTF8TEXT = 17,
  wxIPC_UTF32TEXT = 18,
  wxIPC_UNICODETEXT = wxIPC_UTF16TEXT,
  wxIPC_PRIVATE = 20,
  wxIPC_INVALID = 0,
  wxIPC_TEXT = 1,
  wxIPC_BITMAP = 2,
  wxIPC_METAFILE = 3,
  wxIPC_SYLK = 4,
  wxIPC_DIF = 5,
  wxIPC_TIFF = 6,
  wxIPC_OEMTEXT = 7,
  wxIPC_DIB = 8,
  wxIPC_PALETTE = 9,
  wxIPC_PENDATA = 10,
  wxIPC_RIFF = 11,
  wxIPC_WAVE = 12,
  wxIPC_UTF16TEXT = 13,
  wxIPC_ENHMETAFILE = 14,
  wxIPC_FILENAME = 15,
  wxIPC_LOCALE = 16,
  wxIPC_UTF8TEXT = 17,
  wxIPC_UTF32TEXT = 18,
  wxIPC_UNICODETEXT,
  wxIPC_PRIVATE = 20 }

  An enumeration for formats .

22.326.1 Enumeration Type Documentation

enum wxIPCFormat

An enumeration for formats .

Enumerator

wxIPC_INVALID

wxIPC_TEXT CF_TEXT.

wxIPC_BITMAP CF_BITMAP.

wxIPC_METAFILE CF_METAFILEPICT.

wxIPC_SYLK

wxIPC_DIF

wxIPC_TIFF

wxIPC_OEMTEXT CF_OEMTEXT.

wxIPC_DIB CF_DIB.

wxIPC_PALETTE

wxIPC_PENDATA

wxIPC_RIFF

wxIPC_WAVE

wxIPC_UTF16TEXT CF_UNICODE.

wxIPC_ENHMETAFILE

wxIPC_FILENAME CF_HDROP.

wxIPC_LOCALE

wxIPC_UTF8TEXT

wxIPC_UTF32TEXT

wxIPC_UNICODETEXT

wxIPC_PRIVATE

wxIPC_INVALID

wxIPC_TEXT

wxIPC_BITMAP

wxIPC_METAFILE

wxIPC_SYLK

wxIPC_DIF

wxIPC_TIFF

wxIPC_OEMTEXT

wxIPC_DIB

wxIPC_PALETTE

wxIPC_PENDATA

wxIPC_RIFF

wxIPC_WAVE

wxIPC_UTF16TEXT

wxIPC_ENHMETAFILE

wxIPC_FILENAME

wxIPC_LOCALE

wxIPC_UTF8TEXT

wxIPC_UTF32TEXT

wxIPC_UNICODETEXT

wxIPC_PRIVATE

22.327 interface/wx/joystick.h File Reference

Classes

- class wxJoystick

  wxJoystick allows an application to control one or more joysticks.

22.328 interface/wx/kbdstate.h File Reference

Classes

- class wxKeyboardState

  Provides methods for testing the state of the keyboard modifier keys.

22.329 interface/wx/language.h File Reference

Enumerations

- enum wxLanguage {
  wxLANGUAGE_DEFAULT,
  wxLANGUAGE_UNKNOWN,
  wxLANGUAGE_ABKHAZIAN,
  wxLANGUAGE_AFAR,
  wxLANGUAGE_AFRIKAANS,
  wxLANGUAGE_ALBANIAN,
  wxLANGUAGE_AMHARIC,
  wxLANGUAGE_ARABIC,
  wxLANGUAGE_ARABIC_ALGERIA,
  wxLANGUAGE_ARABIC_BAHRAIN,
  wxLANGUAGE_ARABIC_EGYPT,
  wxLANGUAGE_ARABIC_IRAQ,
  wxLANGUAGE_ARABIC_JORDAN,
  wxLANGUAGE_ARABIC_KUWAIT,
  wxLANGUAGE_ARABIC_LEBANON,
  wxLANGUAGE_ARABIC_LIBYA,
  wxLANGUAGE_ARABIC_MOROCCO,
  wxLANGUAGE_ARABIC_OMAN,
  wxLANGUAGE_ARABIC_QATAR,
  wxLANGUAGE_ARABIC_SAUDI_ARABIA,
  wxLANGUAGE_ARABIC_SUDAN,
  wxLANGUAGE_ARABIC_SYRIA,
  wxLANGUAGE_ARABIC_TUNISIA,
  wxLANGUAGE_ARABIC_UAE,
  wxLANGUAGE_ARABIC_YEMEN,
  wxLANGUAGE_ARMENIAN,
  wxLANGUAGE_ASSAMESE,
  wxLANGUAGE_ASTURIAN,
  wxLANGUAGE_AYMARA,
  wxLANGUAGE_AZERI,
  wxLANGUAGE_AZERI_CYRILLIC,
  wxLANGUAGE_AZERI_LATIN,
  wxLANGUAGE_BASHKIR,
  wxLANGUAGE_BASQUE,
  wxLANGUAGE_BELARUSIAN,
  wxLANGUAGE_BENGALI,
  wxLANGUAGE_BHUTANI,
  wxLANGUAGE_BIHARI,
  wxLANGUAGE_BISLAMA,
  wxLANGUAGE_BOSNIAN,
  wxLANGUAGE_BRETON,
  wxLANGUAGE_BULGARIAN,
  wxLANGUAGE_BURMESE,
  wxLANGUAGE_CATALAN,
  wxLANGUAGE_CHINESE,
  wxLANGUAGE_CHINESE_SIMPLIFIED,
  wxLANGUAGE_CHINESE_TRADITIONAL,
  wxLANGUAGE_CHINESE_HONGKONG,
  wxLANGUAGE_CHINESE_MACAU,
  wxLANGUAGE_CHINESE_SINGAPORE,
  wxLANGUAGE_CHINESE_TAIWAN,
  wxLANGUAGE_CORSICAN,
  wxLANGUAGE_CROATIAN,
  wxLANGUAGE_CZECH,
  wxLANGUAGE_DANISH,
  wxLANGUAGE_DUTCH,
  wxLANGUAGE_DUTCH_BELGIAN,
  wxLANGUAGE_ENGLISH,
  wxLANGUAGE_ENGLISH_UK,
  wxLANGUAGE_ENGLISH_US,
  wxLANGUAGE_ENGLISH_AUSTRALIA,
  wxLANGUAGE_ENGLISH_BELIZE,

The languages supported by wxLocale.

22.329.1 Enumeration Type Documentation

enum wxLanguage

The languages supported by wxLocale.

This enum is generated by misc/languages/genlang.py When making changes, please put them into misc/languages/langtabl.txt

Enumerator

wxLANGUAGE_DEFAULT User's default/preferred language as got from OS.

wxLANGUAGE_UNKNOWN Unknown language, returned if wxLocale::GetSystemLanguage fails.

wxLANGUAGE_ABKHAZIAN

wxLANGUAGE_AFAR

wxLANGUAGE_AFRIKAANS

wxLANGUAGE_ALBANIAN

wxLANGUAGE_AMHARIC

wxLANGUAGE_ARABIC

wxLANGUAGE_ARABIC_ALGERIA

wxLANGUAGE_ARABIC_BAHRAIN

wxLANGUAGE_ARABIC_EGYPT

wxLANGUAGE_ARABIC_IRAQ

wxLANGUAGE_ARABIC_JORDAN

wxLANGUAGE_ARABIC_KUWAIT

wxLANGUAGE_ARABIC_LEBANON

wxLANGUAGE_ARABIC_LIBYA

wxLANGUAGE_ARABIC_MOROCCO

wxLANGUAGE_ARABIC_OMAN

wxLANGUAGE_ARABIC_QATAR

wxLANGUAGE_ARABIC_SAUDI_ARABIA

wxLANGUAGE_ARABIC_SUDAN

wxLANGUAGE_ARABIC_SYRIA

wxLANGUAGE_ARABIC_TUNISIA

wxLANGUAGE_ARABIC_UAE

wxLANGUAGE_ARABIC_YEMEN

wxLANGUAGE_ARMENIAN

wxLANGUAGE_ASSAMESE

wxLANGUAGE_ASTURIAN

wxLANGUAGE_AYMARA

wxLANGUAGE_AZERI

wxLANGUAGE_AZERI_CYRILLIC

wxLANGUAGE_AZERI_LATIN

wxLANGUAGE_BASHKIR

wxLANGUAGE_BASQUE

wxLANGUAGE_BELARUSIAN

wxLANGUAGE_BENGALI
wxLANGUAGE_BHUTANI
wxLANGUAGE_BIHARI
wxLANGUAGE_BISLAMA
wxLANGUAGE_BOSNIAN
wxLANGUAGE_BRETON
wxLANGUAGE_BULGARIAN
wxLANGUAGE_BURMESE
wxLANGUAGE_CATALAN
wxLANGUAGE_CHINESE
wxLANGUAGE_CHINESE_SIMPLIFIED
wxLANGUAGE_CHINESE_TRADITIONAL
wxLANGUAGE_CHINESE_HONGKONG
wxLANGUAGE_CHINESE_MACAU
wxLANGUAGE_CHINESE_SINGAPORE
wxLANGUAGE_CHINESE_TAIWAN
wxLANGUAGE_CORSICAN
wxLANGUAGE_CROATIAN
wxLANGUAGE_CZECH
wxLANGUAGE_DANISH
wxLANGUAGE_DUTCH
wxLANGUAGE_DUTCH_BELGIAN
wxLANGUAGE_ENGLISH
wxLANGUAGE_ENGLISH_UK
wxLANGUAGE_ENGLISH_US
wxLANGUAGE_ENGLISH_AUSTRALIA
wxLANGUAGE_ENGLISH_BELIZE
wxLANGUAGE_ENGLISH_BOTSWANA
wxLANGUAGE_ENGLISH_CANADA
wxLANGUAGE_ENGLISH_CARIBBEAN
wxLANGUAGE_ENGLISH_DENMARK
wxLANGUAGE_ENGLISH_EIRE
wxLANGUAGE_ENGLISH_JAMAICA
wxLANGUAGE_ENGLISH_NEW_ZEALAND
wxLANGUAGE_ENGLISH_PHILIPPINES
wxLANGUAGE_ENGLISH_SOUTH_AFRICA
wxLANGUAGE_ENGLISH_TRINIDAD
wxLANGUAGE_ENGLISH_ZIMBABWE
wxLANGUAGE_ESPERANTO
wxLANGUAGE_ESTONIAN
wxLANGUAGE_FAEROESE
wxLANGUAGE_FARSI
wxLANGUAGE_FIJI
wxLANGUAGE_FINNISH
wxLANGUAGE_FRENCH

wxLANGUAGE_FRENCH_BELGIAN
wxLANGUAGE_FRENCH_CANADIAN
wxLANGUAGE_FRENCH_LUXEMBOURG
wxLANGUAGE_FRENCH_MONACO
wxLANGUAGE_FRENCH_SWISS
wxLANGUAGE_FRISIAN
wxLANGUAGE_GALICIAN
wxLANGUAGE_GEORGIAN
wxLANGUAGE_GERMAN
wxLANGUAGE_GERMAN_AUSTRIAN
wxLANGUAGE_GERMAN_BELGIUM
wxLANGUAGE_GERMAN_LIECHTENSTEIN
wxLANGUAGE_GERMAN_LUXEMBOURG
wxLANGUAGE_GERMAN_SWISS
wxLANGUAGE_GREEK
wxLANGUAGE_GREENLANDIC
wxLANGUAGE_GUARANI
wxLANGUAGE_GUJARATI
wxLANGUAGE_HAUSA
wxLANGUAGE_HEBREW
wxLANGUAGE_HINDI
wxLANGUAGE_HUNGARIAN
wxLANGUAGE_ICELANDIC
wxLANGUAGE_INDONESIAN
wxLANGUAGE_INTERLINGUA
wxLANGUAGE_INTERLINGUE
wxLANGUAGE_INUKTITUT
wxLANGUAGE_INUPIAK
wxLANGUAGE_IRISH
wxLANGUAGE_ITALIAN
wxLANGUAGE_ITALIAN_SWISS
wxLANGUAGE_JAPANESE
wxLANGUAGE_JAVANESE
wxLANGUAGE_KABYLE
wxLANGUAGE_KANNADA
wxLANGUAGE_KASHMIRI
wxLANGUAGE_KASHMIRI_INDIA
wxLANGUAGE_KAZAKH
wxLANGUAGE_KERNEWEK
wxLANGUAGE_KHMER
wxLANGUAGE_KINYARWANDA
wxLANGUAGE_KIRGHIZ
wxLANGUAGE_KIRUNDI
wxLANGUAGE_KONKANI
wxLANGUAGE_KOREAN

wxLANGUAGE_KURDISH
wxLANGUAGE_LAOTHIAN
wxLANGUAGE_LATIN
wxLANGUAGE_LATVIAN
wxLANGUAGE_LINGALA
wxLANGUAGE_LITHUANIAN
wxLANGUAGE_MACEDONIAN
wxLANGUAGE_MALAGASY
wxLANGUAGE_MALAY
wxLANGUAGE_MALAYALAM
wxLANGUAGE_MALAY_BRUNEI_DARUSSALAM
wxLANGUAGE_MALAY_MALAYSIA
wxLANGUAGE_MALTESE
wxLANGUAGE_MANIPURI
wxLANGUAGE_MAORI
wxLANGUAGE_MARATHI
wxLANGUAGE_MOLDAVIAN
wxLANGUAGE_MONGOLIAN
wxLANGUAGE_NAURU
wxLANGUAGE_NEPALI
wxLANGUAGE_NEPALI_INDIA
wxLANGUAGE_NORWEGIAN_BOKMAL
wxLANGUAGE_NORWEGIAN_NYNORSK
wxLANGUAGE_OCCITAN
wxLANGUAGE_ORIYA
wxLANGUAGE_OROMO
wxLANGUAGE_PASHTO
wxLANGUAGE_POLISH
wxLANGUAGE_PORTUGUESE
wxLANGUAGE_PORTUGUESE_BRAZILIAN
wxLANGUAGE_PUNJABI
wxLANGUAGE_QUECHUA
wxLANGUAGE_RHAETO_ROMANCE
wxLANGUAGE_ROMANIAN
wxLANGUAGE_RUSSIAN
wxLANGUAGE_RUSSIAN_UKRAINE
wxLANGUAGE_SAMI
wxLANGUAGE_SAMOAN
wxLANGUAGE_SANGHO
wxLANGUAGE_SANSKRIT
wxLANGUAGE_SCOTS_GAELIC
wxLANGUAGE_SERBIAN
wxLANGUAGE_SERBIAN_CYRILLIC
wxLANGUAGE_SERBIAN_LATIN
wxLANGUAGE_SERBO_CROATIAN

wxLANGUAGE_SESOTHO
wxLANGUAGE_SETSWANA
wxLANGUAGE_SHONA
wxLANGUAGE_SINDHI
wxLANGUAGE_SINHALESE
wxLANGUAGE_SISWATI
wxLANGUAGE_SLOVAK
wxLANGUAGE_SLOVENIAN
wxLANGUAGE_SOMALI
wxLANGUAGE_SPANISH
wxLANGUAGE_SPANISH_ARGENTINA
wxLANGUAGE_SPANISH_BOLIVIA
wxLANGUAGE_SPANISH_CHILE
wxLANGUAGE_SPANISH_COLOMBIA
wxLANGUAGE_SPANISH_COSTA_RICA
wxLANGUAGE_SPANISH_DOMINICAN_REPUBLIC
wxLANGUAGE_SPANISH_ECUADOR
wxLANGUAGE_SPANISH_EL_SALVADOR
wxLANGUAGE_SPANISH_GUATEMALA
wxLANGUAGE_SPANISH_HONDURAS
wxLANGUAGE_SPANISH_MEXICAN
wxLANGUAGE_SPANISH_MODERN
wxLANGUAGE_SPANISH_NICARAGUA
wxLANGUAGE_SPANISH_PANAMA
wxLANGUAGE_SPANISH_PARAGUAY
wxLANGUAGE_SPANISH_PERU
wxLANGUAGE_SPANISH_PUERTO_RICO
wxLANGUAGE_SPANISH_URUGUAY
wxLANGUAGE_SPANISH_US
wxLANGUAGE_SPANISH_VENEZUELA
wxLANGUAGE_SUNDANESE
wxLANGUAGE_SWAHILI
wxLANGUAGE_SWEDISH
wxLANGUAGE_SWEDISH_FINLAND
wxLANGUAGE_TAGALOG
wxLANGUAGE_TAJIK
wxLANGUAGE_TAMIL
wxLANGUAGE_TATAR
wxLANGUAGE_TELUGU
wxLANGUAGE_THAI
wxLANGUAGE_TIBETAN
wxLANGUAGE_TIGRINYA
wxLANGUAGE_TONGA
wxLANGUAGE_TSONGA
wxLANGUAGE_TURKISH

wxLANGUAGE_TURKMEN

wxLANGUAGE_TWI

wxLANGUAGE_UIGHUR

wxLANGUAGE_UKRAINIAN

wxLANGUAGE_URDU

wxLANGUAGE_URDU_INDIA

wxLANGUAGE_URDU_PAKISTAN

wxLANGUAGE_UZBEK

wxLANGUAGE_UZBEK_CYRILLIC

wxLANGUAGE_UZBEK_LATIN

wxLANGUAGE_VALENCIAN

wxLANGUAGE_VIETNAMESE

wxLANGUAGE_VOLAPUK

wxLANGUAGE_WELSH

wxLANGUAGE_WOLOF

wxLANGUAGE_XHOSA

wxLANGUAGE_YIDDISH

wxLANGUAGE_YORUBA

wxLANGUAGE_ZHUANG

wxLANGUAGE_ZULU

wxLANGUAGE_USER_DEFINED For custom, user-defined languages.

wxLANGUAGE_CAMBODIAN Obsolete synonym.

22.330 interface/wx/layout.h File Reference

Classes

- class wxIndividualLayoutConstraint
- class wxLayoutConstraints

Enumerations

- enum wxEdge {
  wxLeft,
  wxTop,
  wxRight,
  wxBottom,
  wxWidth,
  wxHeight,
  wxCentre,
  wxCenter = wxCentre,
  wxCentreX,
  wxCentreY }

- enum wxRelationship {
  wxUnconstrained,
  wxAsIs,
  wxPercentOf,
  wxAbove,
  wxBelow,
  wxLeftOf,
  wxRightOf,
  wxSameAs,
  wxAbsolute }

Variables

- const int wxLAYOUT_DEFAULT_MARGIN = 0

22.330.1 Enumeration Type Documentation

enum wxEdge

Enumerator

wxLeft

wxTop

wxRight

wxBottom

wxWidth

wxHeight

wxCentre

wxCenter

wxCentreX

wxCentreY

enum wxRelationship

Enumerator

wxUnconstrained

wxAsIs

wxPercentOf

wxAbove

wxBelow

wxLeftOf

wxRightOf

wxSameAs

wxAbsolute

22.330.2 Variable Documentation

const int wxLAYOUT_DEFAULT_MARGIN = 0

22.331 interface/wx/laywin.h File Reference

Classes

- class wxLayoutAlgorithm

  wxLayoutAlgorithm implements layout of subwindows in MDI or SDI frames.
- class wxSashLayoutWindow

  wxSashLayoutWindow responds to OnCalculateLayout events generated by wxLayoutAlgorithm.
- class wxQueryLayoutInfoEvent

  This event is sent when wxLayoutAlgorithm wishes to get the size, orientation and alignment of a window.
- class wxCalculateLayoutEvent

  This event is sent by wxLayoutAlgorithm to calculate the amount of the remaining client area that the window should occupy.

Enumerations

- enum wxLayoutOrientation {
  wxLAYOUT_HORIZONTAL,
  wxLAYOUT_VERTICAL }

  Enumeration used by wxLayoutAlgorithm.
- enum wxLayoutAlignment {
  wxLAYOUT_NONE,
  wxLAYOUT_TOP,
  wxLAYOUT_LEFT,
  wxLAYOUT_RIGHT,
  wxLAYOUT_BOTTOM }

  Enumeration used by wxLayoutAlgorithm.

Variables

- wxEventType wxEVT_QUERY_LAYOUT_INFO
- wxEventType wxEVT_CALCULATE_LAYOUT

22.331.1 Enumeration Type Documentation

enum wxLayoutAlignment

Enumeration used by wxLayoutAlgorithm.

Enumerator

wxLAYOUT_NONE

wxLAYOUT_TOP

wxLAYOUT_LEFT

wxLAYOUT_RIGHT

wxLAYOUT_BOTTOM

enum wxLayoutOrientation

Enumeration used by wxLayoutAlgorithm.

Enumerator

wxLAYOUT_HORIZONTAL

wxLAYOUT_VERTICAL

22.331.2 Variable Documentation

wxEventType wxEVT_CALCULATE_LAYOUT

wxEventType wxEVT_QUERY_LAYOUT_INFO

22.332 interface/wx/link.h File Reference

Macros

- #define wxFORCE_LINK_THIS_MODULE(moduleName)

  This macro can be used in conjunction with the wxFORCE_LINK_MODULE() macro to force the linker to include in its output a specific object file.
- #define wxFORCE_LINK_MODULE(moduleName)

  This macro can be used in conjunction with the wxFORCE_LINK_THIS_MODULE() macro to force the linker to include in its output a specific object file.

22.333 interface/wx/list.h File Reference

Classes

- class wxList< T >

  The wxList<T> class provides linked list functionality.
- class wxNode< T >

  wxNode<T> is the node structure used in linked lists (see wxList) and derived classes.

22.334 interface/wx/listbook.h File Reference

Classes

- class wxListbook

  wxListbook is a class similar to wxNotebook but which uses a wxListCtrl to show the labels instead of the tabs.

Macros

- #define wxLB_DEFAULT wxBK_DEFAULT
- #define wxLB_TOP wxBK_TOP
- #define wxLB_BOTTOM wxBK_BOTTOM
- #define wxLB_LEFT wxBK_LEFT
- #define wxLB_RIGHT wxBK_RIGHT
- #define wxLB_ALIGN_MASK wxBK_ALIGN_MASK

Variables

- wxEventType wxEVT_LISTBOOK_PAGE_CHANGED
- wxEventType wxEVT_LISTBOOK_PAGE_CHANGING

22.334.1 Macro Definition Documentation

#define wxLB_ALIGN_MASK wxBK_ALIGN_MASK

#define wxLB_BOTTOM wxBK_BOTTOM

#define wxLB_DEFAULT wxBK_DEFAULT

#define wxLB_LEFT wxBK_LEFT

#define wxLB_RIGHT wxBK_RIGHT

#define wxLB_TOP wxBK_TOP

22.334.2 Variable Documentation

wxEventType wxEVT_LISTBOOK_PAGE_CHANGED

wxEventType wxEVT_LISTBOOK_PAGE_CHANGING

22.335 interface/wx/listbox.h File Reference

Classes

- class wxListBox

  A listbox is used to select one or more of a list of strings.

22.336 interface/wx/longlong.h File Reference

Classes

- class wxLongLong

  This class represents a signed 64 bit long number.
- class wxULongLong

  This class represents an unsigned 64 bit long number.

Macros

- #define wxLongLongFmtSpec

  This macro is defined to contain the `printf()` format specifier using which 64 bit integer numbers (i.e.

Functions

- wxLongLong_t wxLL (number)

  This macro is defined for the platforms with a native 64 bit integer type and allow the use of 64 bit compile time constants:

- wxLongLong_t wxULL (number)

  This macro is defined for the platforms with a native 64 bit integer type and allow the use of 64 bit compile time constants:

22.337 interface/wx/math.h File Reference

Functions

- int wxFinite (double x)

  Returns a non-zero value if x is neither infinite nor NaN (not a number), returns 0 otherwise.
- unsigned int wxGCD (unsigned int u, unsigned int v)

  Returns the greatest common divisor of the two given numbers.
- bool wxIsNaN (double x)

  Returns a non-zero value if x is NaN (not a number), returns 0 otherwise.
- wxFloat64 wxConvertFromIeeeExtended (const wxInt8 *bytes)

  Converts the given array of 10 bytes (corresponding to 80 bits) to a float number according to the IEEE floating point standard format (aka IEEE standard 754).
- void wxConvertToIeeeExtended (wxFloat64 num, wxInt8 *bytes)

  Converts the given floating number num in a sequence of 10 bytes which are stored in the given array bytes (which must be large enough) according to the IEEE floating point standard format (aka IEEE standard 754).
- double wxDegToRad (double deg)

  Convert degrees to radians.
- double wxRadToDeg (double rad)

  Convert radians to degrees.
- int wxRound (double x)

  Small wrapper around round().
- bool wxIsSameDouble (double x, double y)

  Returns true if both double values are identical.
- bool wxIsNullDouble (double x)

  Return true of x is exactly zero.

22.338 interface/wx/mdi.h File Reference

Classes

- class wxMDIClientWindow

  An MDI client window is a child of wxMDIParentFrame, and manages zero or more wxMDIChildFrame objects.
- class wxMDIParentFrame

  An MDI (Multiple Document Interface) parent frame is a window which can contain MDI child frames in its client area which emulates the full desktop.
- class wxMDIChildFrame

  An MDI child frame is a frame that can only exist inside a wxMDIClientWindow, which is itself a child of wxMDI↩ ParentFrame.

22.339 interface/wx/mediactrl.h File Reference

Classes

- class wxMediaEvent

Event wxMediaCtrl uses.

- class wxMediaCtrl

  wxMediaCtrl is a class for displaying types of media, such as videos, audio files, natively through native codecs.

Enumerations

- enum wxMediaState {
  wxMEDIASTATE_STOPPED,
  wxMEDIASTATE_PAUSED,
  wxMEDIASTATE_PLAYING }

  Describes the current state of the media.
- enum wxMediaCtrlPlayerControls {
  wxMEDIACTRLPLAYERCONTROLS_NONE = 0,
  wxMEDIACTRLPLAYERCONTROLS_STEP = 1 << 0,
  wxMEDIACTRLPLAYERCONTROLS_VOLUME = 1 << 1,
  wxMEDIACTRLPLAYERCONTROLS_DEFAULT }

Variables

- wxEventType wxEVT_MEDIA_LOADED
- wxEventType wxEVT_MEDIA_STOP
- wxEventType wxEVT_MEDIA_FINISHED
- wxEventType wxEVT_MEDIA_STATECHANGED
- wxEventType wxEVT_MEDIA_PLAY
- wxEventType wxEVT_MEDIA_PAUSE

22.339.1 Enumeration Type Documentation

enum wxMediaCtrlPlayerControls

Enumerator

wxMEDIACTRLPLAYERCONTROLS_NONE No controls. return wxMediaCtrl to its default state.

wxMEDIACTRLPLAYERCONTROLS_STEP Step controls like fastforward, step one frame etc.

wxMEDIACTRLPLAYERCONTROLS_VOLUME Volume controls like the speaker icon, volume slider, etc.

wxMEDIACTRLPLAYERCONTROLS_DEFAULT Default controls for the toolkit. Currently a combination for `wxMEDIACTRLPLAYERCONTROLS_STEP` and `wxMEDIACTRLPLAYERCONTROLS_VOLUME`.

enum wxMediaState

Describes the current state of the media.

See also

wxMediaCtrl::GetState()

Enumerator

wxMEDIASTATE_STOPPED No media is being currently played.

wxMEDIASTATE_PAUSED Current media is paused.

wxMEDIASTATE_PLAYING There is media currently playing.

22.339.2 Variable Documentation

wxEventType wxEVT_MEDIA_FINISHED

wxEventType wxEVT_MEDIA_LOADED

wxEventType wxEVT_MEDIA_PAUSE

wxEventType wxEVT_MEDIA_PLAY

wxEventType wxEVT_MEDIA_STATECHANGED

wxEventType wxEVT_MEDIA_STOP

22.340 interface/wx/memory.h File Reference

Classes

- class wxDebugContext

  A class for performing various debugging and memory tracing operations.

Macros

- #define WXTRACE(format,...)
- #define WXTRACELEVEL(level, format,...)

Functions

- void wxTrace (const wxString &format,...)
- void wxTraceLevel (int level, const wxString &format,...)

22.341 interface/wx/menu.h File Reference

Classes

- class wxMenuBar

  A menu bar is a series of menus accessible from the top of a frame.

- class wxMenu

  A menu is a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu).

22.342 interface/wx/menuitem.h File Reference

Classes

- class wxMenuItem

  A menu item represents an item in a menu.

22.343 interface/wx/metafile.h File Reference

Classes

- class wxMetafileDC

  *This is a type of device context that allows a metafile object to be created (Windows only), and has most of the characteristics of a normal **wxDC**.*
- class wxMetafile

  *A **wxMetafile** represents the MS Windows metafile object, so metafile operations have no effect in X.*

Functions

- bool wxMakeMetafilePlaceable (const wxString &filename, int minX, int minY, int maxX, int maxY, float scale=1.0)

  Given a filename for an existing, valid metafile (as constructed using wxMetafileDC) makes it into a placeable metafile by prepending a header containing the given bounding box.

22.344 interface/wx/mimetype.h File Reference

Classes

- class wxMimeTypesManager

  This class allows the application to retrieve information about all known MIME types from a system-specific location and the filename extensions to the MIME types and vice versa.
- class wxFileType

  This class holds information about a given file type.
- class wxFileType::MessageParameters

  Class representing message parameters.
- class wxFileTypeInfo

  Container of information about wxFileType.

Variables

- wxMimeTypesManager * wxTheMimeTypesManager

  The global wxMimeTypesManager instance.

22.344.1 Variable Documentation

wxMimeTypesManager* wxTheMimeTypesManager

The global wxMimeTypesManager instance.

22.345 interface/wx/minifram.h File Reference

Classes

- class wxMiniFrame

  A miniframe is a frame with a small title bar.

22.346 interface/wx/modalhook.h File Reference

Classes

- class wxModalDialogHook

  Allows to intercept all modal dialog calls.

22.347 interface/wx/module.h File Reference

Classes

- class wxModule

  The module system is a very simple mechanism to allow applications (and parts of wxWidgets itself) to define initialization and cleanup functions that are automatically called on wxWidgets startup and exit.

22.348 interface/wx/mousemanager.h File Reference

Classes

- class wxMouseEventsManager

  Helper for handling mouse input events in windows containing multiple items.

22.349 interface/wx/mousestate.h File Reference

Classes

- class wxMouseState

  Represents the mouse state.

Enumerations

- enum wxMouseButton {
  wxMOUSE_BTN_ANY = -1,
  wxMOUSE_BTN_NONE = 0,
  wxMOUSE_BTN_LEFT = 1,
  wxMOUSE_BTN_MIDDLE = 2,
  wxMOUSE_BTN_RIGHT = 3,
  wxMOUSE_BTN_AUX1 = 4,
  wxMOUSE_BTN_AUX2 = 5,
  wxMOUSE_BTN_MAX }

  Symbolic names for the mouse buttons.

22.349.1 Enumeration Type Documentation

enum wxMouseButton

Symbolic names for the mouse buttons.

Enumerator

wxMOUSE_BTN_ANY Any mouse button, means to check for any button being pressed for example.

wxMOUSE_BTN_NONE None of the mouse buttons.

wxMOUSE_BTN_LEFT Left mouse button.

wxMOUSE_BTN_MIDDLE Middle mouse button.

wxMOUSE_BTN_RIGHT Right mouse button.

wxMOUSE_BTN_AUX1 First additional mouse button.

wxMOUSE_BTN_AUX2 Second additional mouse button.

wxMOUSE_BTN_MAX

22.350 interface/wx/msgdlg.h File Reference

Classes

- class wxMessageDialog

  This class represents a dialog that shows a single or multi-line message, with a choice of OK, Yes, No and Cancel buttons.
- class wxMessageDialog::ButtonLabel

  Helper class allowing to use either stock id or string labels.

Functions

- int wxMessageBox (const wxString &message, const wxString &caption=wxMessageBoxCaptionStr, int style=wxOK|wxCENTRE, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord)

  Show a general purpose message dialog.

Variables

- const char wxMessageBoxCaptionStr [] = "Message"

  Default message box caption string.

22.350.1 Variable Documentation

const char wxMessageBoxCaptionStr[ ] = "Message"

Default message box caption string.

22.351 interface/wx/msgout.h File Reference

Classes

- class wxMessageOutput

  Simple class allowing to write strings to various output channels.
- class wxMessageOutputStderr

  Output messages to stderr or another STDIO file stream.
- class wxMessageOutputBest

  Output messages in the best possible way.
- class wxMessageOutputDebug

  Output messages to the system debug output channel.
- class wxMessageOutputMessageBox

  Output messages by showing them in a message box.

Enumerations

- enum wxMessageOutputFlags {
  wxMSGOUT_PREFER_STDERR = 0,
  wxMSGOUT_PREFER_MSGBOX = 1 }
  Flags used with wxMessageOutputBest.

22.351.1 Enumeration Type Documentation

enum wxMessageOutputFlags

Flags used with wxMessageOutputBest.

See wxMessageOutputBest::wxMessageOutputBest().

Enumerator

wxMSGOUT_PREFER_STDERR use stderr if available (this is the default)

wxMSGOUT_PREFER_MSGBOX always use message box if available

22.352 interface/wx/msgqueue.h File Reference

Classes

- class wxMessageQueue< T >
  wxMessageQueue allows passing messages between threads.

Enumerations

- enum wxMessageQueueError {
  wxMSGQUEUE_NO_ERROR = 0,
  wxMSGQUEUE_TIMEOUT,
  wxMSGQUEUE_MISC_ERROR }
  Error codes for wxMessageQueue<> operations.

22.353 interface/wx/mstream.h File Reference

Classes

- class wxMemoryOutputStream
  This class allows to use all methods taking a wxOutputStream reference to write to in-memory data.
- class wxMemoryInputStream
  This class allows to use all methods taking a wxInputStream reference to read in-memory data.

22.354 interface/wx/msw/ole/activex.h File Reference

Classes

- class wxActiveXEvent
  An event class for handling ActiveX events passed from wxActiveXContainer.
- class wxActiveXContainer
  wxActiveXContainer is a host for an ActiveX control on Windows (and as such is a platform-specific class).

22.355 interface/wx/msw/ole/automtn.h File Reference

Classes

- class wxVariantDataCurrency

  This class represents a thin wrapper for Microsoft Windows CURRENCY type.
- class wxVariantDataErrorCode

  This class represents a thin wrapper for Microsoft Windows SCODE type (which is the same as HRESULT).
- class wxVariantDataSafeArray

  This class represents a thin wrapper for Microsoft Windows SAFEARRAY type.
- class wxAutomationObject

  *The **wxAutomationObject** class represents an OLE automation object containing a single data member, an IDispatch pointer.*

Enumerations

- enum wxAutomationInstanceFlags {
  wxAutomationInstance_UseExistingOnly = 0,
  wxAutomationInstance_CreateIfNeeded = 1,
  wxAutomationInstance_SilentIfNone = 2 }

  Automation object creation flags.
- enum wxOleConvertVariantFlags {
  wxOleConvertVariant_Default = 0,
  wxOleConvertVariant_ReturnSafeArrays = 1 }

  Flags used for conversions between wxVariant and OLE VARIANT.

22.355.1 Enumeration Type Documentation

enum wxAutomationInstanceFlags

Automation object creation flags.

These flags can be used with wxAutomationObject::GetInstance().

Since

2.9.2

Enumerator

wxAutomationInstance_UseExistingOnly Only use the existing instance, never create a new one. This flag can be used to forbid the creation of a new instance if none is currently running.

wxAutomationInstance_CreateIfNeeded Create a new instance if there are no existing ones. This flag corresponds to the default behaviour of wxAutomationObject::GetInstance() and means that if getting an existing instance failed, we should call wxAutomationObject::CreateInstance() to create a new one.

wxAutomationInstance_SilentIfNone Do not show an error message if no existing instance is currently running. All other errors will still be reported as usual.

enum wxOleConvertVariantFlags

Flags used for conversions between wxVariant and OLE VARIANT.

These flags are used by wxAutomationObject for its wxConvertOleToVariant() calls. They can be obtained by wxAutomationObject::GetConvertVariantFlags() and set by wxAutomationObject::SetConvertVariantFlags().

Since

3.0

Include file:

```
#include <wx/msw/ole/oleutils.h>
```

Enumerator

wxOleConvertVariant_Default Default value.

wxOleConvertVariant_ReturnSafeArrays If this flag is used, SAFEARRAYs contained in OLE VARIANTs will be returned as wxVariants with wxVariantDataSafeArray type instead of wxVariants with the list type containing the (flattened) SAFEARRAY's elements.

22.356 interface/wx/msw/regconf.h File Reference

Classes

- class wxRegConfig

  wxRegConfig implements the wxConfigBase interface for storing and retrieving configuration information using Windows registry.

22.357 interface/wx/msw/registry.h File Reference

Classes

- class wxRegKey

  wxRegKey is a class representing the Windows registry (it is only available under Windows).

22.358 interface/wx/nonownedwnd.h File Reference

Classes

- class wxNonOwnedWindow

  Common base class for all non-child windows.

Macros

- #define wxFRAME_SHAPED 0x0010

  Styles that can be used with any wxNonOwnedWindow:

22.358.1 Macro Definition Documentation

#define wxFRAME_SHAPED 0x0010

Styles that can be used with any wxNonOwnedWindow:

22.359 interface/wx/notebook.h File Reference

Classes

- class wxNotebook

  This class represents a notebook control, which manages multiple windows with associated tabs.

Macros

- #define wxNB_DEFAULT wxBK_DEFAULT
- #define wxNB_TOP wxBK_TOP
- #define wxNB_BOTTOM wxBK_BOTTOM
- #define wxNB_LEFT wxBK_LEFT
- #define wxNB_RIGHT wxBK_RIGHT
- #define wxNB_FIXEDWIDTH 0x0100
- #define wxNB_MULTILINE 0x0200
- #define wxNB_NOPAGETHEME 0x0400
- #define wxNB_FLAT 0x0800

Enumerations

- enum {
  wxNB_HITTEST_NOWHERE = wxBK_HITTEST_NOWHERE,
  wxNB_HITTEST_ONICON = wxBK_HITTEST_ONICON,
  wxNB_HITTEST_ONLABEL = wxBK_HITTEST_ONLABEL,
  wxNB_HITTEST_ONITEM = wxBK_HITTEST_ONITEM,
  wxNB_HITTEST_ONPAGE = wxBK_HITTEST_ONPAGE }

Variables

- wxEventType wxEVT_NOTEBOOK_PAGE_CHANGED
- wxEventType wxEVT_NOTEBOOK_PAGE_CHANGING

22.359.1 Macro Definition Documentation

#define wxNB_BOTTOM wxBK_BOTTOM

#define wxNB_DEFAULT wxBK_DEFAULT

#define wxNB_FIXEDWIDTH 0x0100

#define wxNB_FLAT 0x0800

#define wxNB_LEFT wxBK_LEFT

#define wxNB_MULTILINE 0x0200

#define wxNB_NOPAGETHEME 0x0400

#define wxNB_RIGHT wxBK_RIGHT

#define wxNB_TOP wxBK_TOP

22.359.2 Enumeration Type Documentation

anonymous enum

Enumerator

wxNB_HITTEST_NOWHERE

wxNB_HITTEST_ONICON

wxNB_HITTEST_ONLABEL

wxNB_HITTEST_ONITEM

wxNB_HITTEST_ONPAGE

22.359.3 Variable Documentation

wxEventType wxEVT_NOTEBOOK_PAGE_CHANGED

wxEventType wxEVT_NOTEBOOK_PAGE_CHANGING

22.360 interface/wx/notifmsg.h File Reference

Classes

- class wxNotificationMessage

  This class allows to show the user a message non intrusively.

22.361 interface/wx/numdlg.h File Reference

Functions

- long wxGetNumberFromUser (const wxString &message, const wxString &prompt, const wxString &caption, long value, long min=0, long max=100, wxWindow *parent=NULL, const wxPoint &pos=wxDefaultPosition)

  Shows a dialog asking the user for numeric input.

22.362 interface/wx/numformatter.h File Reference

Classes

- class wxNumberFormatter

  Helper class for formatting and parsing numbers with thousands separators.

22.363 interface/wx/object.h File Reference

Classes

- class wxRefCounter

  This class is used to manage reference-counting providing a simple interface and a counter.
- class wxObject

  This is the root class of many of the wxWidgets classes.
- class wxClassInfo

This class stores meta-information about classes.

- class wxObjectDataPtr< T >

  This is an helper template class primarily written to avoid memory leaks because of missing calls to wxRefCounter::DecRef() and wxObjectRefData::DecRef().

Macros

- #define wxCLASSINFO(className)

  Returns a pointer to the wxClassInfo object associated with this class.
- #define wxDECLARE_ABSTRACT_CLASS(className)

  Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically.
- #define wxDECLARE_DYNAMIC_CLASS(className)

  Used inside a class declaration to make the class known to wxWidgets RTTI system and also declare that the objects of this class should be dynamically creatable from run-time type information.
- #define wxDECLARE_CLASS(className)

  Used inside a class declaration to declare that the class should be made known to the class hierarchy, but objects of this class cannot be created dynamically.
- #define wxIMPLEMENT_ABSTRACT_CLASS(className, baseClassName)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information.
- #define wxIMPLEMENT_ABSTRACT_CLASS2(className, baseClassName1, baseClassName2)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes.
- #define wxIMPLEMENT_DYNAMIC_CLASS(className, baseClassName)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.
- #define wxIMPLEMENT_DYNAMIC_CLASS2(className, baseClassName1, baseClassName2)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.
- #define wxIMPLEMENT_CLASS(className, baseClassName)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information, and whose instances can be created dynamically.
- #define wxIMPLEMENT_CLASS2(className, baseClassName1, baseClassName2)

  Used in a C++ implementation file to complete the declaration of a class that has run-time type information and two base classes, and whose instances can be created dynamically.
- #define wx_const_cast(T, x)

  Same as `const_cast<T>(x)` if the compiler supports const cast or (T)x for old compilers.
- #define wx_reinterpret_cast(T, x)

  Same as `reinterpret_cast<T>(x)` if the compiler supports reinterpret cast or (T)x for old compilers.
- #define wx_static_cast(T, x)

  Same as `static_cast<T>(x)` if the compiler supports static cast or (T)x for old compilers.
- #define wx_truncate_cast(T, x)

  This case doesn't correspond to any standard cast but exists solely to make casts which possibly result in a truncation of an integer value more readable.
- #define wxConstCast(ptr, classname)

  This macro expands into `const_cast<classname *>(ptr)` if the compiler supports const_cast or into an old, C-style cast, otherwise.
- #define wxDynamicCast(ptr, classname)

  *This macro returns the pointer ptr cast to the type classname * if the pointer is of this type (the check is done during the run-time) or NULL otherwise.*
- #define wxDynamicCastThis(classname)

  This macro is equivalent to `wxDynamicCast(this, classname)` but the latter provokes spurious compilation warnings from some compilers (because it tests whether `this` pointer is non-NULL which is always true), so this macro should be used to avoid them.

- #define wxStaticCast(ptr, classname)

  This macro checks that the cast is valid in debug mode (an assert failure will result if wxDynamicCast(ptr, classname) == NULL) and then returns the result of executing an equivalent of `static_cast<classname *>(ptr)`.

- #define WXDEBUG_NEW(arg)

  This is defined in debug mode to be call the redefined new operator with filename and line number arguments.

Typedefs

- typedef wxRefCounter wxObjectRefData

Functions

- wxObject * wxCreateDynamicObject (const wxString &className)

  Creates and returns an object of the given class, if the class has been registered with the dynamic class system using DECLARE...

22.363.1 Typedef Documentation

typedef wxRefCounter wxObjectRefData

22.364 interface/wx/odcombo.h File Reference

Classes

- class wxOwnerDrawnComboBox

  wxOwnerDrawnComboBox is a combobox with owner-drawn list items.

Enumerations

- enum wxOwnerDrawnComboBoxPaintingFlags {
  wxODCB_PAINTING_CONTROL = 0x0001,
  wxODCB_PAINTING_SELECTED = 0x0002 }
- enum {
  wxODCB_DCLICK_CYCLES = wxCC_SPECIAL_DCLICK,
  wxODCB_STD_CONTROL_PAINT = 0x1000 }

  New window styles for wxOwnerDrawnComboBox.

22.364.1 Enumeration Type Documentation

anonymous enum

New window styles for wxOwnerDrawnComboBox.

Enumerator

wxODCB_DCLICK_CYCLES Double-clicking cycles item if wxCB_READONLY is also used.

wxODCB_STD_CONTROL_PAINT If used, control itself is not custom paint using callback. Even if this is not used, writable combo is never custom paint until SetCustomPaintWidth is called

enum wxOwnerDrawnComboBoxPaintingFlags

Enumerator

wxODCB_PAINTING_CONTROL Combo control is being painted, instead of a list item. Argument item may be `wxNOT_FOUND` in this case.

wxODCB_PAINTING_SELECTED An item with selection background is being painted. DC text colour should already be correct.

22.365 interface/wx/overlay.h File Reference

Classes

- class wxOverlay

  Creates an overlay over an existing window, allowing for manipulations like rubberbanding, etc.
- class wxDCOverlay

  Connects an overlay with a drawing DC.

22.366 interface/wx/palette.h File Reference

Classes

- class wxPalette

  A palette is a table that maps pixel values to RGB colours.

Variables

- wxPalette wxNullPalette

  An empty palette.

22.366.1 Variable Documentation

wxPalette wxNullPalette

An empty palette.

22.367 interface/wx/panel.h File Reference

Classes

- class wxPanel

  A panel is a window on which controls are placed.

22.368 interface/wx/ribbon/panel.h File Reference

Classes

- class wxRibbonPanelEvent

Event used to indicate various actions relating to a wxRibbonPanel.

- class wxRibbonPanel

  Serves as a container for a group of (ribbon) controls.

22.369 interface/wx/pen.h File Reference

Classes

- class wxPen

  A pen is a drawing tool for drawing outlines.

- class wxPenList

  There is only one instance of this class: wxThePenList.

Enumerations

- enum wxPenStyle {
  wxPENSTYLE_INVALID = -1,
  wxPENSTYLE_SOLID,
  wxPENSTYLE_DOT,
  wxPENSTYLE_LONG_DASH,
  wxPENSTYLE_SHORT_DASH,
  wxPENSTYLE_DOT_DASH,
  wxPENSTYLE_USER_DASH,
  wxPENSTYLE_TRANSPARENT,
  wxPENSTYLE_STIPPLE_MASK_OPAQUE,
  wxPENSTYLE_STIPPLE_MASK,
  wxPENSTYLE_STIPPLE,
  wxPENSTYLE_BDIAGONAL_HATCH,
  wxPENSTYLE_CROSSDIAG_HATCH,
  wxPENSTYLE_FDIAGONAL_HATCH,
  wxPENSTYLE_CROSS_HATCH,
  wxPENSTYLE_HORIZONTAL_HATCH,
  wxPENSTYLE_VERTICAL_HATCH,
  wxPENSTYLE_FIRST_HATCH,
  wxPENSTYLE_LAST_HATCH }

  The possible styles for a wxPen.

- enum wxPenJoin {
  wxJOIN_INVALID = -1,
  wxJOIN_BEVEL = 120,
  wxJOIN_MITER,
  wxJOIN_ROUND }

  The possible join values of a wxPen.

- enum wxPenCap {
  wxCAP_INVALID = -1,
  wxCAP_ROUND = 130,
  wxCAP_PROJECTING,
  wxCAP_BUTT }

  The possible cap values of a wxPen.

Variables

- wxPen wxNullPen

  An empty pen.

- wxPen * wxRED_PEN
  Red pen.
- wxPen * wxBLUE_PEN
  Blue pen.
- wxPen * wxCYAN_PEN
  Cyan pen.
- wxPen * wxGREEN_PEN
  Green pen.
- wxPen * wxYELLOW_PEN
  Yellow pen.
- wxPen * wxBLACK_PEN
  Black pen.
- wxPen * wxWHITE_PEN
  White pen.
- wxPen * wxTRANSPARENT_PEN
  Transparent pen.
- wxPen * wxBLACK_DASHED_PEN
  Black dashed pen.
- wxPen * wxGREY_PEN
  Grey pen.
- wxPen * wxMEDIUM_GREY_PEN
  Medium-grey pen.
- wxPen * wxLIGHT_GREY_PEN
  Light-grey pen.
- wxPenList * wxThePenList
  The global list of wxPen objects ready to be re-used (for better performances).

22.369.1 Enumeration Type Documentation

enum wxPenCap

The possible cap values of a wxPen.

Todo use wxPENCAP_ prefix

Enumerator

wxCAP_INVALID
wxCAP_ROUND
wxCAP_PROJECTING
wxCAP_BUTT

enum wxPenJoin

The possible join values of a wxPen.

Todo use wxPENJOIN_ prefix

Enumerator

wxJOIN_INVALID
wxJOIN_BEVEL
wxJOIN_MITER
wxJOIN_ROUND

enum wxPenStyle

The possible styles for a wxPen.

Note that hatched pen styles are not supported by X11-based ports, including wxGTK.

Enumerator

wxPENSTYLE_INVALID

wxPENSTYLE_SOLID Solid style.

wxPENSTYLE_DOT Dotted style.

wxPENSTYLE_LONG_DASH Long dashed style.

wxPENSTYLE_SHORT_DASH Short dashed style.

wxPENSTYLE_DOT_DASH Dot and dash style.

wxPENSTYLE_USER_DASH Use the user dashes: see wxPen::SetDashes.

wxPENSTYLE_TRANSPARENT No pen is used.

wxPENSTYLE_STIPPLE_MASK_OPAQUE **Todo** WHAT's this?

wxPENSTYLE_STIPPLE_MASK **Todo** WHAT's this?

wxPENSTYLE_STIPPLE Use the stipple bitmap.

wxPENSTYLE_BDIAGONAL_HATCH Backward diagonal hatch.

wxPENSTYLE_CROSSDIAG_HATCH Cross-diagonal hatch.

wxPENSTYLE_FDIAGONAL_HATCH Forward diagonal hatch.

wxPENSTYLE_CROSS_HATCH Cross hatch.

wxPENSTYLE_HORIZONTAL_HATCH Horizontal hatch.

wxPENSTYLE_VERTICAL_HATCH Vertical hatch.

wxPENSTYLE_FIRST_HATCH First of the hatch styles (inclusive).

wxPENSTYLE_LAST_HATCH Last of the hatch styles (inclusive).

22.369.2 Variable Documentation

wxPen* wxBLACK_DASHED_PEN

Black dashed pen.

Except for the color and for the `wxPENSTYLE_SHORT_DASH` it has all standard attributes (1-pixel width, `wxCAP_ROUND` style, etc...).

wxPen* wxBLACK_PEN

Black pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxBLUE_PEN

Blue pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxCYAN_PEN

Cyan pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxGREEN_PEN

Green pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxGREY_PEN

Grey pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxLIGHT_GREY_PEN

Light-grey pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxMEDIUM_GREY_PEN

Medium-grey pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen wxNullPen

An empty pen.

wxPen::IsOk() always returns false for this object.

wxPen* wxRED_PEN

Red pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPenList* wxThePenList

The global list of wxPen objects ready to be re-used (for better performances).

wxPen* wxTRANSPARENT_PEN

Transparent pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxWHITE_PEN

White pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

wxPen* wxYELLOW_PEN

Yellow pen.

Except for the color it has all standard attributes (1-pixel width, `wxPENSTYLE_SOLID` and `wxCAP_ROUND` styles, etc...).

22.370 interface/wx/persist.h File Reference

Classes

- class wxPersistenceManager

  Provides support for automatically saving and restoring object properties to persistent storage.
- class wxPersistentObject

  Base class for persistent object adapters.

Functions

- template<class T >
  wxPersistentObject * wxCreatePersistentObject (T *obj)

  Function used to create the correct persistent adapter for the given type of objects.
- template<class T >
  bool wxPersistentRegisterAndRestore (T *obj, const wxString &name=wxString())

  A shorter synonym for wxPersistenceManager::RegisterAndRestore().

22.370.1 Function Documentation

template<class T > wxPersistentObject* wxCreatePersistentObject (T * *obj*)

Function used to create the correct persistent adapter for the given type of objects.

To be precise, there is no such template function definition but there are overloads of wxCreatePersistentObject() taking different object types for all wxWidgets classes supporting persistence. And you may also define your own overloads to integrate your custom classes with wxWidgets persistence framework.

See also

Defining Custom Persistent Windows

Include file:

```
#include <wx/persist.h>
```

template<class T > bool wxPersistentRegisterAndRestore (T * *obj,* const wxString & *name* = wxString ())

A shorter synonym for wxPersistenceManager::RegisterAndRestore().

This function simply calls wxPersistenceManager::RegisterAndRestore() but using it results in slightly shorter code as it calls wxPersistenceManager::Get() internally. As an additional convenience, this function can also set the window name.

Parameters

obj	wxWindow-derived object to register with persistence manager and to try to restore the settings for.
name	If not empty, *obj* name is changed to the provided value before registering it.

Returns

true if the settings were restored or false otherwise (this will always be the case when the program runs for the first time, for example).

Since

2.9.0, *name* is new in 2.9.1.

Include file:

```
#include <wx/persist.h>
```

22.371 interface/wx/persist/toplevel.h File Reference

Classes

- class wxPersistentTLW

  Persistence adapter for wxTopLevelWindow.

Functions

- wxPersistentObject * wxCreatePersistentObject (wxTopLevelWindow *book)

  Overload allowing persistence adapter creation for wxTopLevelWindow-derived objects.

22.371.1 Function Documentation

wxPersistentObject* wxCreatePersistentObject (wxTopLevelWindow * *book*)

Overload allowing persistence adapter creation for wxTopLevelWindow-derived objects.

22.372 interface/wx/toplevel.h File Reference

Classes

- class wxTopLevelWindow

  wxTopLevelWindow is a common base class for wxDialog and wxFrame.

Macros

- #define wxDEFAULT_FRAME_STYLE

Enumerations

- enum {
  wxUSER_ATTENTION_INFO = 1,
  wxUSER_ATTENTION_ERROR = 2 }

  Styles used with wxTopLevelWindow::RequestUserAttention().
- enum {
  wxFULLSCREEN_NOMENUBAR = 0x0001,
  wxFULLSCREEN_NOTOOLBAR = 0x0002,
  wxFULLSCREEN_NOSTATUSBAR = 0x0004,
  wxFULLSCREEN_NOBORDER = 0x0008,
  wxFULLSCREEN_NOCAPTION = 0x0010,
  wxFULLSCREEN_ALL }

  Styles used with wxTopLevelWindow::ShowFullScreen().

22.372.1 Macro Definition Documentation

#define wxDEFAULT_FRAME_STYLE

Value:

```
(wxSYSTEM_MENU |           \
                           wxRESIZE_BORDER |
      \                    wxMINIMIZE_BOX |
      \                    wxMAXIMIZE_BOX |
      \                    wxCLOSE_BOX |
      \                    wxCAPTION |
      \                    wxCLIP_CHILDREN)
```

22.372.2 Enumeration Type Documentation

anonymous enum

Styles used with wxTopLevelWindow::RequestUserAttention().

Enumerator

wxUSER_ATTENTION_INFO Requests user attention,.

wxUSER_ATTENTION_ERROR Results in a more drastic action.

anonymous enum

Styles used with wxTopLevelWindow::ShowFullScreen().

Enumerator

wxFULLSCREEN_NOMENUBAR Don't display the menu bar.

wxFULLSCREEN_NOTOOLBAR Don't display toolbar bars.

wxFULLSCREEN_NOSTATUSBAR Don't display the status bar.

wxFULLSCREEN_NOBORDER Don't display any border.

wxFULLSCREEN_NOCAPTION Don't display a caption.

wxFULLSCREEN_ALL Combination of all above, will display the least possible.

22.373 interface/wx/persist/treebook.h File Reference

Classes

- class wxPersistentTreeBookCtrl

  Persistence adapter for wxTreebook.

Functions

- wxPersistentObject * wxCreatePersistentObject (wxTreebook *book)

  Overload allowing persistence adapter creation for wxTreebook objects.

22.373.1 Function Documentation

wxPersistentObject* wxCreatePersistentObject (wxTreebook * *book*)

Overload allowing persistence adapter creation for wxTreebook objects.

22.374 interface/wx/treebook.h File Reference

Classes

- class wxTreebook

  This class is an extension of the wxNotebook class that allows a tree structured set of pages to be shown in a control.

Variables

- wxEventType wxEVT_TREEBOOK_PAGE_CHANGED
- wxEventType wxEVT_TREEBOOK_PAGE_CHANGING
- wxEventType wxEVT_TREEBOOK_NODE_COLLAPSED
- wxEventType wxEVT_TREEBOOK_NODE_EXPANDED

22.374.1 Variable Documentation

wxEventType wxEVT_TREEBOOK_NODE_COLLAPSED

wxEventType wxEVT_TREEBOOK_NODE_EXPANDED

wxEventType wxEVT_TREEBOOK_PAGE_CHANGED

wxEventType wxEVT_TREEBOOK_PAGE_CHANGING

22.375 interface/wx/persist/window.h File Reference

Classes

- class wxPersistentWindow< T >

  Base class for persistent windows.

22.376 interface/wx/window.h File Reference

Classes

- struct wxVisualAttributes

  Struct containing all the visual attributes of a control.
- class wxWindow

  wxWindow is the base class for all windows and represents any visible object on screen.
- class wxWindow::ChildrenRepositioningGuard

  Helper for ensuring EndRepositioningChildren() is called correctly.

Enumerations

- enum wxShowEffect {
  wxSHOW_EFFECT_NONE,
  wxSHOW_EFFECT_ROLL_TO_LEFT,
  wxSHOW_EFFECT_ROLL_TO_RIGHT,
  wxSHOW_EFFECT_ROLL_TO_TOP,
  wxSHOW_EFFECT_ROLL_TO_BOTTOM,
  wxSHOW_EFFECT_SLIDE_TO_LEFT,
  wxSHOW_EFFECT_SLIDE_TO_RIGHT,
  wxSHOW_EFFECT_SLIDE_TO_TOP,
  wxSHOW_EFFECT_SLIDE_TO_BOTTOM,
  wxSHOW_EFFECT_BLEND,
  wxSHOW_EFFECT_EXPAND,
  wxSHOW_EFFECT_MAX }

  Valid values for wxWindow::ShowWithEffect() and wxWindow::HideWithEffect().
- enum { wxSEND_EVENT_POST = 1 }

  flags for SendSizeEvent()
- enum wxWindowVariant {
  wxWINDOW_VARIANT_NORMAL,
  wxWINDOW_VARIANT_SMALL,
  wxWINDOW_VARIANT_MINI,
  wxWINDOW_VARIANT_LARGE,
  wxWINDOW_VARIANT_MAX }

  Different window variants, on platforms like eg mac uses different rendering sizes.

Functions

- wxWindow * wxFindWindowAtPointer (wxPoint &pt)

  Find the deepest window at the mouse pointer position, returning the window and current pointer position in screen coordinates.
- wxWindow * wxGetActiveWindow ()

  Gets the currently active window (implemented for MSW and GTK only currently, always returns NULL in the other ports).
- wxWindow * wxGetTopLevelParent (wxWindow *window)

  Returns the first top level parent of the given window, or in other words, the frame or dialog containing it, or NULL.

22.376.1 Enumeration Type Documentation

anonymous enum

flags for SendSizeEvent()

Enumerator

wxSEND_EVENT_POST

enum wxShowEffect

Valid values for wxWindow::ShowWithEffect() and wxWindow::HideWithEffect().

Enumerator

wxSHOW_EFFECT_NONE No effect, equivalent to normal wxWindow::Show() or Hide() call.

Since

2.9.1

wxSHOW_EFFECT_ROLL_TO_LEFT Roll window to the left.

wxSHOW_EFFECT_ROLL_TO_RIGHT Roll window to the right.

wxSHOW_EFFECT_ROLL_TO_TOP Roll window to the top.

wxSHOW_EFFECT_ROLL_TO_BOTTOM Roll window to the bottom.

wxSHOW_EFFECT_SLIDE_TO_LEFT Slide window to the left.

wxSHOW_EFFECT_SLIDE_TO_RIGHT Slide window to the right.

wxSHOW_EFFECT_SLIDE_TO_TOP Slide window to the top.

wxSHOW_EFFECT_SLIDE_TO_BOTTOM Slide window to the bottom.

wxSHOW_EFFECT_BLEND Fade in or out effect.

wxSHOW_EFFECT_EXPAND Expanding or collapsing effect.

wxSHOW_EFFECT_MAX

enum wxWindowVariant

Different window variants, on platforms like eg mac uses different rendering sizes.

Enumerator

wxWINDOW_VARIANT_NORMAL Normal size.

wxWINDOW_VARIANT_SMALL Smaller size (about 25 % smaller than normal)

wxWINDOW_VARIANT_MINI Mini size (about 33 % smaller than normal)

wxWINDOW_VARIANT_LARGE Large size (about 25 % larger than normal)

wxWINDOW_VARIANT_MAX

22.377 interface/wx/pickerbase.h File Reference

Classes

- class wxPickerBase

  Base abstract class for all pickers which support an auxiliary text control.

Macros

- #define wxPB_USE_TEXTCTRL 0x0002
- #define wxPB_SMALL 0x8000

22.377.1 Macro Definition Documentation

#define wxPB_SMALL 0x8000

#define wxPB_USE_TEXTCTRL 0x0002

22.378 interface/wx/platform.h File Reference

Macros

- #define wxCHECK_GCC_VERSION(major, minor)

  Returns true if the compiler being used is GNU C++ and its version is at least major.minor or greater.

- #define wxCHECK_SUNCC_VERSION(major, minor)

  Returns true if the compiler being used is Sun CC Pro and its version is at least major.minor or greater.

- #define wxCHECK_VISUALC_VERSION(major)

  Returns true if the compiler being used is Visual C++ and its version is at least major or greater.

- #define wxCHECK_W32API_VERSION(major, minor)

  Returns true if the version of w32api headers used is major.minor or greater.

22.379 interface/wx/platinfo.h File Reference

Classes

- struct wxLinuxDistributionInfo

  A structure containing information about a Linux distribution as returned by the `lsb_release` utility.

- class wxPlatformInfo

  This class holds information about the operating system, the toolkit and the basic architecture of the machine where the application is currently running.

Enumerations

- enum wxOperatingSystemId {
  wxOS_UNKNOWN = 0,
  wxOS_MAC_OS = 1 << 0,
  wxOS_MAC_OSX_DARWIN = 1 << 1,
  wxOS_MAC = wxOS_MAC_OS|wxOS_MAC_OSX_DARWIN,
  wxOS_WINDOWS_9X = 1 << 2,
  wxOS_WINDOWS_NT = 1 << 3,
  wxOS_WINDOWS_MICRO = 1 << 4,
  wxOS_WINDOWS_CE = 1 << 5,
  wxOS_WINDOWS,
  wxOS_UNIX_LINUX = 1 << 6,
  wxOS_UNIX_FREEBSD = 1 << 7,
  wxOS_UNIX_OPENBSD = 1 << 8,
  wxOS_UNIX_NETBSD = 1 << 9,
  wxOS_UNIX_SOLARIS = 1 << 10,
  wxOS_UNIX_AIX = 1 << 11,
  wxOS_UNIX_HPUX = 1 << 12,
  wxOS_UNIX,
  wxOS_DOS = 1 << 15,
  wxOS_OS2 = 1 << 16 }

  The following are the operating systems which are recognized by wxWidgets and whose version can be detected at run-time.

- enum wxPortId {
  wxPORT_UNKNOWN = 0,
  wxPORT_BASE = 1 << 0,
  wxPORT_MSW = 1 << 1,
  wxPORT_MOTIF = 1 << 2,
  wxPORT_GTK = 1 << 3,
  wxPORT_DFB = 1 << 4,
  wxPORT_X11 = 1 << 5,
  wxPORT_OS2 = 1 << 6,
  wxPORT_MAC = 1 << 7,
  wxPORT_COCOA = 1 << 8,
  wxPORT_WINCE = 1 << 9 }

  The list of wxWidgets ports.

- enum wxArchitecture {
  wxARCH_INVALID = -1,
  wxARCH_32,
  wxARCH_64,
  wxARCH_MAX }

  The architecture of the operating system (regardless of the build environment of wxWidgets library - see wxIs↩ Platform64Bit() documentation for more info).

- enum wxEndianness {
  wxENDIAN_INVALID = -1,
  wxENDIAN_BIG,
  wxENDIAN_LITTLE,
  wxENDIAN_PDP,
  wxENDIAN_MAX }

  The endian-ness of the machine.

22.379.1 Enumeration Type Documentation

enum wxArchitecture

The architecture of the operating system (regardless of the build environment of wxWidgets library - see wxIs↩ Platform64Bit() documentation for more info).

Enumerator

wxARCH_INVALID returned on error

wxARCH_32 32 bit

wxARCH_64 64 bit

wxARCH_MAX

enum wxEndianness

The endian-ness of the machine.

Enumerator

wxENDIAN_INVALID returned on error

wxENDIAN_BIG 4321

wxENDIAN_LITTLE 1234

wxENDIAN_PDP 3412

wxENDIAN_MAX

enum wxOperatingSystemId

The following are the operating systems which are recognized by wxWidgets and whose version can be detected at run-time.

The values of the constants are chosen so that they can be combined as flags; this allows to check for operating system families like e.g. `wxOS_MAC` and `wxOS_UNIX`.

Note that you can obtain more detailed information about the current OS version in use by checking the major and minor version numbers returned by wxGetOsVersion() or by wxPlatformInfo::GetOSMajorVersion(), wxPlatform↩ Info::GetOSMinorVersion().

Enumerator

wxOS_UNKNOWN returned on error

wxOS_MAC_OS Apple Mac OS 8/9/X with Mac paths.

wxOS_MAC_OSX_DARWIN Apple Mac OS X with Unix paths.

wxOS_MAC A combination of all `wxOS_MAC_*` values previously listed.

wxOS_WINDOWS_9X Windows 9x family (95/98/ME)

wxOS_WINDOWS_NT Windows NT family (NT/2000/XP/Vista/7)

wxOS_WINDOWS_MICRO MicroWindows.

wxOS_WINDOWS_CE Windows CE (Window Mobile)

wxOS_WINDOWS A combination of all `wxOS_WINDOWS_*` values previously listed.

wxOS_UNIX_LINUX Linux.

wxOS_UNIX_FREEBSD FreeBSD.

wxOS_UNIX_OPENBSD OpenBSD.

wxOS_UNIX_NETBSD NetBSD.

wxOS_UNIX_SOLARIS SunOS.

wxOS_UNIX_AIX AIX.

wxOS_UNIX_HPUX HP/UX.

wxOS_UNIX A combination of all `wxOS_UNIX_*` values previously listed.

wxOS_DOS Microsoft DOS.

wxOS_OS2 OS/2.

enum wxPortId

The list of wxWidgets ports.

Some of them can be used with more than a single (native) toolkit; e.g. wxWinCE port sources can be used with smartphones, pocket PCs and handheld devices SDKs.

Enumerator

wxPORT_UNKNOWN returned on error

wxPORT_BASE wxBase, no native toolkit used

wxPORT_MSW wxMSW, native toolkit is Windows API

wxPORT_MOTIF wxMotif, using [Open]Motif or Lesstif

wxPORT_GTK wxGTK, using GTK+ 1.x, 2.x, GPE or Maemo

wxPORT_DFB wxDFB, using wxUniversal

wxPORT_X11 wxX11, using wxUniversal

wxPORT_OS2 wxOS2, using OS/2 Presentation Manager

wxPORT_MAC wxMac, using Carbon or Classic Mac API

wxPORT_COCOA wxCocoa, using Cocoa NextStep/Mac API

wxPORT_WINCE wxWinCE, toolkit is WinCE SDK API

22.380 interface/wx/popupwin.h File Reference

Classes

- class wxPopupWindow

  A special kind of top level window used for popup menus, combobox popups and such.

- class wxPopupTransientWindow

  A wxPopupWindow which disappears automatically when the user clicks mouse outside it or if it loses focus in any other way.

22.381 interface/wx/position.h File Reference

Classes

- class wxPosition

  This class represents the position of an item in any kind of grid of rows and columns such as wxGridBagSizer, or wxHVScrolledWindow.

22.382 interface/wx/power.h File Reference

Classes

- class wxPowerEvent

  The power events are generated when the system power state changes, e.g.
- class wxPowerResource

  Helper functions for acquiring and releasing the given power resource.
- class wxPowerResourceBlocker

  Helper RAII class ensuring that power resources are released.

Enumerations

- enum wxPowerType {
  wxPOWER_SOCKET,
  wxPOWER_BATTERY,
  wxPOWER_UNKNOWN }
- enum wxBatteryState {
  wxBATTERY_NORMAL_STATE,
  wxBATTERY_LOW_STATE,
  wxBATTERY_CRITICAL_STATE,
  wxBATTERY_SHUTDOWN_STATE,
  wxBATTERY_UNKNOWN_STATE }
- enum wxPowerResourceKind {
  wxPOWER_RESOURCE_SCREEN,
  wxPOWER_RESOURCE_SYSTEM }

  Possible power resources that can be locked by wxPowerResourceBlocker.

Variables

- wxEventType wxEVT_POWER_SUSPENDING
- wxEventType wxEVT_POWER_SUSPENDED
- wxEventType wxEVT_POWER_SUSPEND_CANCEL
- wxEventType wxEVT_POWER_RESUME

22.382.1 Enumeration Type Documentation

enum wxBatteryState

Enumerator

wxBATTERY_NORMAL_STATE

wxBATTERY_LOW_STATE

wxBATTERY_CRITICAL_STATE

wxBATTERY_SHUTDOWN_STATE

wxBATTERY_UNKNOWN_STATE

enum wxPowerResourceKind

Possible power resources that can be locked by wxPowerResourceBlocker.

Since

3.1.0

Enumerator

wxPOWER_RESOURCE_SCREEN Use to prevent automatic display power off.

wxPOWER_RESOURCE_SYSTEM Use to prevent automatic system suspend.

enum wxPowerType

Enumerator

wxPOWER_SOCKET

wxPOWER_BATTERY

wxPOWER_UNKNOWN

22.382.2 Variable Documentation

wxEventType wxEVT_POWER_RESUME

wxEventType wxEVT_POWER_SUSPEND_CANCEL

wxEventType wxEVT_POWER_SUSPENDED

wxEventType wxEVT_POWER_SUSPENDING

22.383 interface/wx/preferences.h File Reference

Classes

- class wxPreferencesEditor

  Manage preferences dialog.
- class wxPreferencesPage

  One page of preferences dialog.
- class wxStockPreferencesPage

  Specialization of wxPreferencesPage useful for certain commonly used preferences page.

22.384 interface/wx/print.h File Reference

Classes

- class wxPreviewControlBar

  This is the default implementation of the preview control bar, a panel with buttons and a zoom control.
- class wxPreviewCanvas

  A preview canvas is the default canvas used by the print preview system to display the preview.
- class wxPreviewFrame

  This class provides the default method of managing the print preview interface.
- class wxPrintPreview

  Objects of this class manage the print preview process.
- class wxPrinter

This class represents the Windows or PostScript printer, and is the vehicle through which printing may be launched by an application.

- class wxPrintout

  This class encapsulates the functionality of printing out an application document.

- class wxPrintAbortDialog

  The dialog created by default by the print framework that enables aborting the printing process.

Macros

- #define wxPREVIEW_PRINT 1
- #define wxPREVIEW_PREVIOUS 2
- #define wxPREVIEW_NEXT 4
- #define wxPREVIEW_ZOOM 8
- #define wxPREVIEW_FIRST 16
- #define wxPREVIEW_LAST 32
- #define wxPREVIEW_GOTO 64
- #define wxPREVIEW_DEFAULT
- #define wxID_PREVIEW_CLOSE 1
- #define wxID_PREVIEW_NEXT 2
- #define wxID_PREVIEW_PREVIOUS 3
- #define wxID_PREVIEW_PRINT 4
- #define wxID_PREVIEW_ZOOM 5
- #define wxID_PREVIEW_FIRST 6
- #define wxID_PREVIEW_LAST 7
- #define wxID_PREVIEW_GOTO 8
- #define wxID_PREVIEW_ZOOM_IN 9
- #define wxID_PREVIEW_ZOOM_OUT 10

Enumerations

- enum wxPrinterError {
  wxPRINTER_NO_ERROR = 0,
  wxPRINTER_CANCELLED,
  wxPRINTER_ERROR }
- enum wxPreviewFrameModalityKind {
  wxPreviewFrame_AppModal,
  wxPreviewFrame_WindowModal,
  wxPreviewFrame_NonModal }

  Preview frame modality kind.

22.384.1 Macro Definition Documentation

#define wxID_PREVIEW_CLOSE 1

#define wxID_PREVIEW_FIRST 6

#define wxID_PREVIEW_GOTO 8

#define wxID_PREVIEW_LAST 7

#define wxID_PREVIEW_NEXT 2

#define wxID_PREVIEW_PREVIOUS 3

#define wxID_PREVIEW_PRINT 4

#define wxID_PREVIEW_ZOOM 5

#define wxID_PREVIEW_ZOOM_IN 9

#define wxID_PREVIEW_ZOOM_OUT 10

#define wxPREVIEW_DEFAULT

Value:

```
(wxPREVIEW_PREVIOUS|wxPREVIEW_NEXT|
      wxPREVIEW_ZOOM\
                          |wxPREVIEW_FIRST|wxPREVIEW_GOTO|
      wxPREVIEW_LAST)
```

#define wxPREVIEW_FIRST 16

#define wxPREVIEW_GOTO 64

#define wxPREVIEW_LAST 32

#define wxPREVIEW_NEXT 4

#define wxPREVIEW_PREVIOUS 2

#define wxPREVIEW_PRINT 1

#define wxPREVIEW_ZOOM 8

22.384.2 Enumeration Type Documentation

enum **wxPreviewFrameModalityKind**

Preview frame modality kind.

The elements of this enum can be used with wxPreviewFrame::Initialize() to indicate how should the preview frame be shown.

Since

2.9.2

Enumerator

wxPreviewFrame_AppModal Disable all the other top level windows while the preview frame is shown. This is the default behaviour.

wxPreviewFrame_WindowModal Disable only the parent window while the preview frame is shown.

wxPreviewFrame_NonModal Show the preview frame non-modally and don't disable any other windows.

enum **wxPrinterError**

Enumerator

wxPRINTER_NO_ERROR

wxPRINTER_CANCELLED

wxPRINTER_ERROR

22.385 interface/wx/printdlg.h File Reference

Classes

- class wxPrintDialog

  This class represents the print and print setup common dialogs.
- class wxPageSetupDialog

  This class represents the page setup common dialog.

22.386 interface/wx/process.h File Reference

Classes

- class wxProcess

  The objects of this class are used in conjunction with the wxExecute() function.
- class wxProcessEvent

  A process event is sent to the wxEvtHandler specified to wxProcess when a process is terminated.

Variables

- wxEventType wxEVT_END_PROCESS

22.386.1 Variable Documentation

wxEventType wxEVT_END_PROCESS

22.387 interface/wx/progdlg.h File Reference

Classes

- class wxGenericProgressDialog

  This class represents a dialog that shows a short message and a progress bar.
- class wxProgressDialog

  If supported by the platform this class will provide the platform's native progress dialog, else it will simply be the wxGenericProgressDialog.

Macros

- #define wxPD_CAN_ABORT 0x0001
- #define wxPD_APP_MODAL 0x0002
- #define wxPD_AUTO_HIDE 0x0004
- #define wxPD_ELAPSED_TIME 0x0008
- #define wxPD_ESTIMATED_TIME 0x0010
- #define wxPD_SMOOTH 0x0020
- #define wxPD_REMAINING_TIME 0x0040
- #define wxPD_CAN_SKIP 0x0080

22.387.1 Macro Definition Documentation

#define wxPD_APP_MODAL 0x0002

#define wxPD_AUTO_HIDE 0x0004

#define wxPD_CAN_ABORT 0x0001

#define wxPD_CAN_SKIP 0x0080

#define wxPD_ELAPSED_TIME 0x0008

#define wxPD_ESTIMATED_TIME 0x0010

#define wxPD_REMAINING_TIME 0x0040

#define wxPD_SMOOTH 0x0020

22.388 interface/wx/propdlg.h File Reference

Classes

- class wxPropertySheetDialog

  This class represents a property sheet dialog: a tabbed dialog for showing settings.

Enumerations

- enum wxPropertySheetDialogFlags {
  wxPROPSHEET_DEFAULT = 0x0001,
  wxPROPSHEET_NOTEBOOK = 0x0002,
  wxPROPSHEET_TOOLBOOK = 0x0004,
  wxPROPSHEET_CHOICEBOOK = 0x0008,
  wxPROPSHEET_LISTBOOK = 0x0010,
  wxPROPSHEET_BUTTONTOOLBOOK = 0x0020,
  wxPROPSHEET_TREEBOOK = 0x0040,
  wxPROPSHEET_SHRINKTOFIT = 0x0100 }

  Values used by wxPropertySheetDialog::SetSheetStyle.

22.388.1 Enumeration Type Documentation

enum **wxPropertySheetDialogFlags**

Values used by wxPropertySheetDialog::SetSheetStyle.

Enumerator

wxPROPSHEET_DEFAULT Uses the default look and feel for the controller window, normally a notebook except on Smartphone where a choice control is used.

wxPROPSHEET_NOTEBOOK Uses a notebook for the controller window.

wxPROPSHEET_TOOLBOOK Uses a toolbook for the controller window.

wxPROPSHEET_CHOICEBOOK Uses a choicebook for the controller window.

wxPROPSHEET_LISTBOOK Uses a listbook for the controller window.

wxPROPSHEET_BUTTONTOOLBOOK Uses a button toolbox for the controller window.

wxPROPSHEET_TREEBOOK Uses a treebook for the controller window.

wxPROPSHEET_SHRINKTOFIT Shrinks the dialog window to fit the currently selected page (common behaviour for property sheets on Mac OS X).

22.389 interface/wx/propgrid/editors.h File Reference

Classes

- class wxPGEditor

  Base class for custom wxPropertyGrid editors.
- class wxPGMultiButton

  This class can be used to have multiple buttons in a property editor.

22.390 interface/wx/propgrid/manager.h File Reference

Classes

- class wxPropertyGridPage

  Holder of property grid page information.
- class wxPropertyGridManager

  wxPropertyGridManager is an efficient multi-page version of wxPropertyGrid, which can optionally have toolbar for mode and page selection, a help text box, and a header.

22.391 interface/wx/propgrid/property.h File Reference

- #define wxPG_PROP_MAX wxPG_PROP_AUTO_UNSPECIFIED

  Topmost flag.
- #define wxPG_PROP_PARENTAL_FLAGS

  Property with children must have one of these set, otherwise iterators will not work correctly.
- enum wxPGPropertyFlags {
  wxPG_PROP_MODIFIED = 0x0001,
  wxPG_PROP_DISABLED = 0x0002,
  wxPG_PROP_HIDDEN = 0x0004,
  wxPG_PROP_CUSTOMIMAGE = 0x0008,
  wxPG_PROP_NOEDITOR = 0x0010,
  wxPG_PROP_COLLAPSED = 0x0020,
  wxPG_PROP_INVALID_VALUE = 0x0040,
  wxPG_PROP_WAS_MODIFIED = 0x0200,
  wxPG_PROP_AGGREGATE = 0x0400,
  wxPG_PROP_CHILDREN_ARE_COPIES = 0x0800,
  wxPG_PROP_PROPERTY = 0x1000,
  wxPG_PROP_CATEGORY = 0x2000,
  wxPG_PROP_MISC_PARENT = 0x4000,
  wxPG_PROP_READONLY = 0x8000,
  wxPG_PROP_COMPOSED_VALUE = 0x00010000,
  wxPG_PROP_USES_COMMON_VALUE = 0x00020000,
  wxPG_PROP_AUTO_UNSPECIFIED = 0x00040000,
  wxPG_PROP_CLASS_SPECIFIC_1 = 0x00080000,
  wxPG_PROP_CLASS_SPECIFIC_2 = 0x00100000,
  wxPG_PROP_BEING_DELETED = 0x00200000 }

Classes

- class wxPGProperty

  wxPGProperty is base class for all wxPropertyGrid properties.
- class wxPGCell

  Base class for wxPropertyGrid cell information.
- class wxPGChoices

  Helper class for managing choices of wxPropertyGrid properties.

Macros

- #define wxNullProperty ((wxPGProperty*)NULL)

- #define wxPG_ATTR_DEFAULT_VALUE wxS("DefaultValue")

  Set default value for property.
- #define wxPG_ATTR_MIN wxS("Min")

  Universal, int or double.
- #define wxPG_ATTR_MAX wxS("Max")

  Universal, int or double.
- #define wxPG_ATTR_UNITS wxS("Units")

  Universal, string.
- #define wxPG_ATTR_HINT wxS("Hint")

  When set, will be shown as 'greyed' text in property's value cell when the actual displayed value is blank.
- #define wxPG_ATTR_INLINE_HELP wxS("InlineHelp")
- #define wxPG_ATTR_AUTOCOMPLETE wxS("AutoComplete")

  Universal, wxArrayString.
- #define wxPG_BOOL_USE_CHECKBOX wxS("UseCheckbox")

  wxBoolProperty and wxFlagsProperty specific.
- #define wxPG_BOOL_USE_DOUBLE_CLICK_CYCLING wxS("UseDClickCycling")

  wxBoolProperty and wxFlagsProperty specific.
- #define wxPG_FLOAT_PRECISION wxS("Precision")

  wxFloatProperty (and similar) specific, int, default -1.
- #define wxPG_STRING_PASSWORD wxS("Password")

  The text will be echoed as asterisks (wxTE_PASSWORD will be passed to textctrl etc).
- #define wxPG_UINT_BASE wxS("Base")

  Define base used by a wxUIntProperty.
- #define wxPG_UINT_PREFIX wxS("Prefix")

  Define prefix rendered to wxUIntProperty.
- #define wxPG_FILE_WILDCARD wxS("Wildcard")

  wxFileProperty/wxImageFileProperty specific, wxChar, default is detected/varies.*
- #define wxPG_FILE_SHOW_FULL_PATH wxS("ShowFullPath")

  wxFileProperty/wxImageFileProperty specific, int, default 1.
- #define wxPG_FILE_SHOW_RELATIVE_PATH wxS("ShowRelativePath")

  Specific to wxFileProperty and derived properties, wxString, default empty.
- #define wxPG_FILE_INITIAL_PATH wxS("InitialPath")

  Specific to wxFileProperty and derived properties, wxString, default is empty.
- #define wxPG_FILE_DIALOG_TITLE wxS("DialogTitle")

  Specific to wxFileProperty and derivatives, wxString, default is empty.
- #define wxPG_FILE_DIALOG_STYLE wxS("DialogStyle")

  Specific to wxFileProperty and derivatives, long, default is 0.
- #define wxPG_DIR_DIALOG_MESSAGE wxS("DialogMessage")

Specific to wxDirProperty, wxString, default is empty.

- #define wxPG_ARRAY_DELIMITER wxS("Delimiter")

  wxArrayStringProperty's string delimiter character.
- #define wxPG_DATE_FORMAT wxS("DateFormat")

  Sets displayed date format for wxDateProperty.
- #define wxPG_DATE_PICKER_STYLE wxS("PickerStyle")

  Sets wxDatePickerCtrl window style used with wxDateProperty.
- #define wxPG_ATTR_SPINCTRL_STEP wxS("Step")

  SpinCtrl editor, int or double.
- #define wxPG_ATTR_SPINCTRL_WRAP wxS("Wrap")

  SpinCtrl editor, bool.
- #define wxPG_ATTR_SPINCTRL_MOTIONSPIN wxS("MotionSpin")

  SpinCtrl editor, bool.
- #define wxPG_ATTR_MULTICHOICE_USERSTRINGMODE wxS("UserStringMode")

  wxMultiChoiceProperty, int.
- #define wxPG_COLOUR_ALLOW_CUSTOM wxS("AllowCustom")

  wxColourProperty and its kind, int, default 1.
- #define wxPG_COLOUR_HAS_ALPHA wxS("HasAlpha")

  wxColourProperty and its kind: Set to True in order to support editing alpha colour component.

22.391.1 Macro Definition Documentation

#define wxNullProperty ((wxPGProperty*)NULL)

#define wxPG_ARRAY_DELIMITER wxS("Delimiter")

wxArrayStringProperty's string delimiter character.

If this is a quotation mark or hyphen, then strings will be quoted instead (with given character).

Default delimiter is quotation mark.

#define wxPG_ATTR_AUTOCOMPLETE wxS("AutoComplete")

Universal, wxArrayString.

Set to enable auto-completion in any wxTextCtrl-based property editor.

#define wxPG_ATTR_DEFAULT_VALUE wxS("DefaultValue")

Set default value for property.

22.391.2 wxPropertyGrid Property Attribute Identifiers

wxPGProperty::SetAttribute() and wxPropertyGridInterface::SetPropertyAttribute() accept one of these as attribute name argument.

You can use strings instead of constants. However, some of these constants are redefined to use cached strings which may reduce your binary size by some amount.

#define wxPG_ATTR_HINT wxS("Hint")

When set, will be shown as 'greyed' text in property's value cell when the actual displayed value is blank.

#define wxPG_ATTR_INLINE_HELP wxS("InlineHelp")

Deprecated Use "Hint" (wxPG_ATTR_HINT) instead.

#define wxPG_ATTR_MAX wxS("Max")

Universal, int or double.

Maximum value for numeric properties.

#define wxPG_ATTR_MIN wxS("Min")

Universal, int or double.

Minimum value for numeric properties.

#define wxPG_ATTR_MULTICHOICE_USERSTRINGMODE wxS("UserStringMode")

wxMultiChoiceProperty, int.

If 0, no user strings allowed. If 1, user strings appear before list strings. If 2, user strings appear after list string.

#define wxPG_ATTR_SPINCTRL_MOTIONSPIN wxS("MotionSpin")

SpinCtrl editor, bool.

If true, value can also by changed by moving mouse when left mouse button is being pressed.

#define wxPG_ATTR_SPINCTRL_STEP wxS("Step")

SpinCtrl editor, int or double.

How much number changes when button is pressed (or up/down on keyboard).

#define wxPG_ATTR_SPINCTRL_WRAP wxS("Wrap")

SpinCtrl editor, bool.

If true, value wraps at Min/Max.

#define wxPG_ATTR_UNITS wxS("Units")

Universal, string.

When set, will be shown as text after the displayed text value. Alternatively, if third column is enabled, text will be shown there (for any type of property).

#define wxPG_BOOL_USE_CHECKBOX wxS("UseCheckbox")

wxBoolProperty and wxFlagsProperty specific.

Value type is bool. Default value is False.

When set to True, bool property will use check box instead of a combo box as its editor control. If you set this attribute for a wxFlagsProperty, it is automatically applied to child bool properties.

#define wxPG_BOOL_USE_DOUBLE_CLICK_CYCLING wxS("UseDClickCycling")

wxBoolProperty and wxFlagsProperty specific.

Value type is bool. Default value is False.

Set to True for the bool property to cycle value on double click (instead of showing the popup listbox). If you set this attribute for a wxFlagsProperty, it is automatically applied to child bool properties.

#define wxPG_COLOUR_ALLOW_CUSTOM wxS("AllowCustom")

wxColourProperty and its kind, int, default 1.

Setting this attribute to 0 hides custom colour from property's list of choices.

#define wxPG_COLOUR_HAS_ALPHA wxS("HasAlpha")

wxColourProperty and its kind: Set to True in order to support editing alpha colour component.

#define wxPG_DATE_FORMAT wxS("DateFormat")

Sets displayed date format for wxDateProperty.

#define wxPG_DATE_PICKER_STYLE wxS("PickerStyle")

Sets wxDatePickerCtrl window style used with wxDateProperty.

Default is wxDP_DEFAULT | wxDP_SHOWCENTURY. Using wxDP_ALLOWNONE will enable better unspecified value support in the editor.

#define wxPG_DIR_DIALOG_MESSAGE wxS("DialogMessage")

Specific to wxDirProperty, wxString, default is empty.

Sets a specific message for the dir dialog.

#define wxPG_FILE_DIALOG_STYLE wxS("DialogStyle")

Specific to wxFileProperty and derivatives, long, default is 0.

Sets a specific wxFileDialog style for the file dialog, e.g. wxFD_SAVE.

Since

2.9.4

#define wxPG_FILE_DIALOG_TITLE wxS("DialogTitle")

Specific to wxFileProperty and derivatives, wxString, default is empty.

Sets a specific title for the dir dialog.

#define wxPG_FILE_INITIAL_PATH wxS("InitialPath")

Specific to wxFileProperty and derived properties, wxString, default is empty.

Sets the initial path of where to look for files.

#define wxPG_FILE_SHOW_FULL_PATH wxS("ShowFullPath")

wxFileProperty/wxImageFileProperty specific, int, default 1.

When 0, only the file name is shown (i.e. drive and directory are hidden).

#define wxPG_FILE_SHOW_RELATIVE_PATH wxS("ShowRelativePath")

Specific to wxFileProperty and derived properties, wxString, default empty.

If set, then the filename is shown relative to the given path string.

#define wxPG_FILE_WILDCARD wxS("Wildcard")

wxFileProperty/wxImageFileProperty specific, wxChar*, default is detected/varies.

Sets the wildcard used in the triggered wxFileDialog. Format is the same.

#define wxPG_FLOAT_PRECISION wxS("Precision")

wxFloatProperty (and similar) specific, int, default -1.

Sets the (max) precision used when floating point value is rendered as text. The default -1 means infinite precision.

#define wxPG_PROP_MAX wxPG_PROP_AUTO_UNSPECIFIED

Topmost flag.

#define wxPG_PROP_PARENTAL_FLAGS

Value:

```
((wxPGPropertyFlags)(wxPG_PROP_AGGREGATE | \
                     wxPG_PROP_CATEGORY | \
                     wxPG_PROP_MISC_PARENT))
```

Property with children must have one of these set, otherwise iterators will not work correctly.

Code should automatically take care of this, however.

#define wxPG_STRING_PASSWORD wxS("Password")

The text will be echoed as asterisks (wxTE_PASSWORD will be passed to textctrl etc).

#define wxPG_UINT_BASE wxS("Base")

Define base used by a wxUIntProperty.

Valid constants are wxPG_BASE_OCT, wxPG_BASE_DEC, wxPG_BASE_HEX and wxPG_BASE_HEXL (lowercase characters).

#define wxPG_UINT_PREFIX wxS("Prefix")

Define prefix rendered to wxUIntProperty.

Accepted constants wxPG_PREFIX_NONE, wxPG_PREFIX_0x, and wxPG_PREFIX_DOLLAR_SIGN. **Note:** Only wxPG_PREFIX_NONE works with Decimal and Octal numbers.

22.391.3 Enumeration Type Documentation

enum wxPGPropertyFlags

22.391.4 wxPGProperty Flags

Enumerator

wxPG_PROP_MODIFIED Indicates bold font.

wxPG_PROP_DISABLED Disables ('greyed' text and editor does not activate) property.

wxPG_PROP_HIDDEN Hider button will hide this property.

wxPG_PROP_CUSTOMIMAGE This property has custom paint image just in front of its value. If property only draws custom images into a popup list, then this flag should not be set.

wxPG_PROP_NOEDITOR Do not create text based editor for this property (but button-triggered dialog and choice are ok).

wxPG_PROP_COLLAPSED Property is collapsed, ie. it's children are hidden.

wxPG_PROP_INVALID_VALUE If property is selected, then indicates that validation failed for pending value. If property is not selected, then indicates that the actual property value has failed validation (NB: this behaviour is not currently supported, but may be used in the future).

wxPG_PROP_WAS_MODIFIED Switched via SetWasModified(). Temporary flag - only used when setting/changing property value.

wxPG_PROP_AGGREGATE If set, then child properties (if any) are private, and should be "invisible" to the application.

wxPG_PROP_CHILDREN_ARE_COPIES If set, then child properties (if any) are copies and should not be deleted in dtor.

wxPG_PROP_PROPERTY Classifies this item as a non-category. Used for faster item type identification.

wxPG_PROP_CATEGORY Classifies this item as a category. Used for faster item type identification.

wxPG_PROP_MISC_PARENT Classifies this item as a property that has children, but is not aggregate (ie children are not private).

wxPG_PROP_READONLY Property is read-only. Editor is still created for wxTextCtrl-based property editors. For others, editor is not usually created because they do implement wxTE_READONLY style or equivalent.

wxPG_PROP_COMPOSED_VALUE Property's value is composed from values of child properties.

Remarks

This flag cannot be used with property iterators.

wxPG_PROP_USES_COMMON_VALUE Common value of property is selectable in editor.

Remarks

This flag cannot be used with property iterators.

wxPG_PROP_AUTO_UNSPECIFIED Property can be set to unspecified value via editor. Currently, this applies to following properties:

- wxIntProperty, wxUIntProperty, wxFloatProperty, wxEditEnumProperty: Clear the text field

Remarks

This flag cannot be used with property iterators.

See also

wxPGProperty::SetAutoUnspecified()

wxPG_PROP_CLASS_SPECIFIC_1 Indicates the bit useable by derived properties.

wxPG_PROP_CLASS_SPECIFIC_2 Indicates the bit useable by derived properties.

wxPG_PROP_BEING_DELETED Indicates that the property is being deleted and should be ignored.

22.392 interface/wx/propgrid/propgridiface.h File Reference

Classes

- class wxPropertyGridInterface

  Most of the shared property manipulation interface shared by wxPropertyGrid, wxPropertyGridPage, and wxPropertyGridManager is defined in this class.

22.393 interface/wx/propgrid/propgridpagestate.h File Reference

Classes

- struct wxPropertyGridHitTestResult
- class wxPropertyGridIterator
- class wxPGVIterator

Macros

- #define wxPG_IT_CHILDREN(A) (A<<16)

Enumerations

- enum wxPG_ITERATOR_FLAGS {
  wxPG_ITERATE_PROPERTIES,
  wxPG_ITERATE_HIDDEN = (wxPG_PROP_HIDDEN|wxPG_IT_CHILDREN(wxPG_PROP_COLLAPSED)),
  wxPG_ITERATE_FIXED_CHILDREN = (wxPG_IT_CHILDREN(wxPG_PROP_AGGREGATE)|wxPG_ITERATE_PROPERTIES),
  wxPG_ITERATE_CATEGORIES = (wxPG_PROP_CATEGORY|wxPG_IT_CHILDREN(wxPG_PROP_CATEGORY)|wxPG_PROP_COLLAPSED),
  wxPG_ITERATE_ALL_PARENTS = (wxPG_PROP_MISC_PARENT|wxPG_PROP_AGGREGATE|wxPG_PROP_CATEGORY),
  wxPG_ITERATE_ALL_PARENTS_RECURSIVELY = (wxPG_ITERATE_ALL_PARENTS|wxPG_IT_CHILDREN(wxPG_ITERATE_ALL_PARENTS)),
  wxPG_ITERATOR_FLAGS_ALL,
  wxPG_ITERATOR_MASK_OP_ITEM = wxPG_ITERATOR_FLAGS_ALL,
  wxPG_ITERATOR_MASK_OP_PARENT = wxPG_ITERATOR_FLAGS_ALL,
  wxPG_ITERATE_VISIBLE = (wxPG_ITERATE_PROPERTIES|wxPG_PROP_CATEGORY|wxPG_IT_CHILDREN(wxPG_PROP_AGGREGATE)),
  wxPG_ITERATE_ALL = (wxPG_ITERATE_VISIBLE|wxPG_ITERATE_HIDDEN),
  wxPG_ITERATE_NORMAL = (wxPG_ITERATE_PROPERTIES|wxPG_ITERATE_HIDDEN),
  wxPG_ITERATE_DEFAULT = wxPG_ITERATE_NORMAL }

22.393.1 Macro Definition Documentation

#define wxPG_IT_CHILDREN(*A*) (A<<16)

22.393.2 Enumeration Type Documentation

enum wxPG_ITERATOR_FLAGS

22.393.3 wxPropertyGridIterator Flags

NOTES: At lower 16-bits, there are flags to check if item will be included. At higher 16-bits, there are same flags, but to instead check if children will be included.

Enumerator

wxPG_ITERATE_PROPERTIES Iterate through 'normal' property items (does not include children of aggregate or hidden items by default).

wxPG_ITERATE_HIDDEN Iterate children of collapsed parents, and individual items that are hidden.

wxPG_ITERATE_FIXED_CHILDREN Iterate children of parent that is an aggregate property (ie. has fixed children).

wxPG_ITERATE_CATEGORIES Iterate categories. Note that even without this flag, children of categories are still iterated through.

wxPG_ITERATE_ALL_PARENTS

wxPG_ITERATE_ALL_PARENTS_RECURSIVELY

wxPG_ITERATOR_FLAGS_ALL

wxPG_ITERATOR_MASK_OP_ITEM

wxPG_ITERATOR_MASK_OP_PARENT

wxPG_ITERATE_VISIBLE Combines all flags needed to iterate through visible properties (ie. hidden properties and children of collapsed parents are skipped).

wxPG_ITERATE_ALL Iterate all items.

wxPG_ITERATE_NORMAL Iterate through individual properties (ie. categories and children of aggregate properties are skipped).

wxPG_ITERATE_DEFAULT Default iterator flags.

22.394 interface/wx/protocol/ftp.h File Reference

Classes

- class wxFTP

  wxFTP can be used to establish a connection to an FTP server and perform all the usual operations.

22.395 interface/wx/protocol/http.h File Reference

Classes

- class wxHTTP

  wxHTTP can be used to establish a connection to an HTTP server.

22.396 interface/wx/protocol/protocol.h File Reference

Classes

- class wxProtocol

  Represents a protocol for use with wxURL.

Enumerations

- enum wxProtocolError {
  wxPROTO_NOERR = 0,
  wxPROTO_NETERR,
  wxPROTO_PROTERR,
  wxPROTO_CONNERR,
  wxPROTO_INVVAL,
  wxPROTO_NOHNDLR,
  wxPROTO_NOFILE,
  wxPROTO_ABRT,
  wxPROTO_RCNCT,
  wxPROTO_STREAMING }

  Error values returned by wxProtocol.

22.396.1 Enumeration Type Documentation

enum wxProtocolError

Error values returned by wxProtocol.

Enumerator

wxPROTO_NOERR No error.

wxPROTO_NETERR A generic network error occurred.

wxPROTO_PROTERR An error occurred during negotiation.

wxPROTO_CONNERR The client failed to connect the server.

wxPROTO_INVVAL Invalid value.

wxPROTO_NOHNDLR Not currently used.

wxPROTO_NOFILE The remote file doesn't exist.

wxPROTO_ABRT Last action aborted.

wxPROTO_RCNCT An error occurred during reconnection.

wxPROTO_STREAMING Someone tried to send a command during a transfer.

22.397 interface/wx/quantize.h File Reference

Classes

- class wxQuantize

  Performs quantization, or colour reduction, on a wxImage.

22.398 interface/wx/radiobox.h File Reference

Classes

- class wxRadioBox

  A radio box item is used to select one of number of mutually exclusive choices.

22.399 interface/wx/radiobut.h File Reference

Classes

- class wxRadioButton

  A radio button item is a button which usually denotes one of several mutually exclusive options.

22.400 interface/wx/rawbmp.h File Reference

Classes

- class wxPixelData< Image, PixelFormat >

  A class template with ready to use implementations for getting direct and efficient access to wxBitmap's internal data and wxImage's internal data through a standard interface.
- class wxPixelData< Image, PixelFormat >::Iterator

  The iterator of class wxPixelData.

22.401 interface/wx/rearrangectrl.h File Reference

Classes

- class wxRearrangeList

  A listbox-like control allowing the user to rearrange the items and to enable or disable them.
- class wxRearrangeCtrl

  A composite control containing a wxRearrangeList and the buttons allowing to move the items in it.
- class wxRearrangeDialog

  A dialog allowing the user to rearrange the specified items.

22.402 interface/wx/recguard.h File Reference

Classes

- class wxRecursionGuardFlag

  This is a completely opaque class which exists only to be used with wxRecursionGuard, please see the example in that class' documentation.
- class wxRecursionGuard

  wxRecursionGuard is a very simple class which can be used to prevent reentrancy problems in a function.

22.403 interface/wx/regex.h File Reference

Classes

- class wxRegEx

  wxRegEx represents a regular expression.

Enumerations

- enum {
  wxRE_EXTENDED = 0,
  wxRE_ADVANCED = 1,
  wxRE_BASIC = 2,
  wxRE_ICASE = 4,
  wxRE_NOSUB = 8,
  wxRE_NEWLINE = 16,
  wxRE_DEFAULT = wxRE_EXTENDED }
- enum {
  wxRE_NOTBOL = 32,
  wxRE_NOTEOL = 64 }

22.403.1 Enumeration Type Documentation

anonymous enum

Flags for regex compilation to be used with wxRegEx::Compile().

Enumerator

wxRE_EXTENDED Use extended regex syntax.

wxRE_ADVANCED Use advanced RE syntax (built-in regex only).

wxRE_BASIC Use basic RE syntax.

wxRE_ICASE Ignore case in match.

wxRE_NOSUB Only check match, don't set back references.

wxRE_NEWLINE If not set, treat '
' as an ordinary character, otherwise it is special: it is not matched by '. ' and '^' and '$' always match after/before it regardless of the setting of wxRE_NOT[BE]OL.

wxRE_DEFAULT Default flags.

anonymous enum

Flags for regex matching to be used with wxRegEx::Matches(). These flags are mainly useful when doing several matches in a long string to prevent erroneous matches for '^' and '$':

Enumerator

wxRE_NOTBOL '^' doesn't match at the start of line.

wxRE_NOTEOL '$' doesn't match at the end of line.

22.404 interface/wx/region.h File Reference

Classes

- class wxRegionIterator

This class is used to iterate through the rectangles in a region, typically when examining the damaged regions of a window within an OnPaint call.

- class wxRegion

  A wxRegion represents a simple or complex region on a device context or window.

Enumerations

- enum wxRegionContain {
  wxOutRegion = 0,
  wxPartRegion = 1,
  wxInRegion = 2 }

  Types of results returned from a call to wxRegion::Contains().

Variables

- wxRegion wxNullRegion

  An empty region.

22.404.1 Enumeration Type Documentation

enum wxRegionContain

Types of results returned from a call to wxRegion::Contains().

Enumerator

wxOutRegion The specified value is not contained within this region.

wxPartRegion The specified value is partially contained within this region. On Windows, this result is not supported. wxInRegion will be returned instead.

wxInRegion The specified value is fully contained within this region. On Windows, this result will be returned even if only part of the specified value is contained in this region.

22.404.2 Variable Documentation

wxRegion wxNullRegion

An empty region.

22.405 interface/wx/renderer.h File Reference

Classes

- struct wxSplitterRenderParams

  This is just a simple `struct` used as a return value of wxRendererNative::GetSplitterParams().

- struct wxHeaderButtonParams

  This `struct` can optionally be used with wxRendererNative::DrawHeaderButton() to specify custom values used to draw the text or bitmap label.

- class wxDelegateRendererNative

  wxDelegateRendererNative allows reuse of renderers code by forwarding all the wxRendererNative methods to the given object and thus allowing you to only modify some of its methods – without having to reimplement all of them.

- class wxRendererNative

First, a brief introduction to wxRendererNative and why it is needed.

- struct wxRendererVersion

  This simple struct represents the wxRendererNative interface version and is only used as the return value of wxRendererNative::GetVersion().

Enumerations

- enum {
  wxCONTROL_NONE = 0x00000000,
  wxCONTROL_DISABLED = 0x00000001,
  wxCONTROL_FOCUSED = 0x00000002,
  wxCONTROL_PRESSED = 0x00000004,
  wxCONTROL_SPECIAL = 0x00000008,
  wxCONTROL_ISDEFAULT = wxCONTROL_SPECIAL,
  wxCONTROL_ISSUBMENU = wxCONTROL_SPECIAL,
  wxCONTROL_EXPANDED = wxCONTROL_SPECIAL,
  wxCONTROL_SIZEGRIP = wxCONTROL_SPECIAL,
  wxCONTROL_FLAT = wxCONTROL_SPECIAL,
  wxCONTROL_CURRENT = 0x00000010,
  wxCONTROL_SELECTED = 0x00000020,
  wxCONTROL_CHECKED = 0x00000040,
  wxCONTROL_CHECKABLE = 0x00000080,
  wxCONTROL_UNDETERMINED = wxCONTROL_CHECKABLE }
- enum wxTitleBarButton {
  wxTITLEBAR_BUTTON_CLOSE = 0x01000000,
  wxTITLEBAR_BUTTON_MAXIMIZE = 0x02000000,
  wxTITLEBAR_BUTTON_ICONIZE = 0x04000000,
  wxTITLEBAR_BUTTON_RESTORE = 0x08000000,
  wxTITLEBAR_BUTTON_HELP = 0x10000000 }

  Title bar buttons supported by wxRendererNative::DrawTitleBarBitmap().
- enum wxHeaderSortIconType {
  wxHDR_SORT_ICON_NONE,
  wxHDR_SORT_ICON_UP,
  wxHDR_SORT_ICON_DOWN }

  Used to specify the type of sort arrow used with wxRendererNative::DrawHeaderButton().

22.405.1 Enumeration Type Documentation

anonymous enum

The following rendering flags are defined for wxRendererNative:

Enumerator

wxCONTROL_NONE Default state, no special flags.

Since

3.1.0

wxCONTROL_DISABLED Control is disabled.

wxCONTROL_FOCUSED Currently has keyboard focus.

wxCONTROL_PRESSED (Button) is pressed.

wxCONTROL_SPECIAL Control-specific bit.

wxCONTROL_ISDEFAULT Only for the buttons.

wxCONTROL_ISSUBMENU Only for the menu items.

wxCONTROL_EXPANDED Only for the tree items.

wxCONTROL_SIZEGRIP Only for the status bar panes.

wxCONTROL_FLAT Checkboxes only: flat border.

wxCONTROL_CURRENT Mouse is currently over the control.

wxCONTROL_SELECTED Selected item in e.g. listbox.

wxCONTROL_CHECKED (Check/radio button) is checked.

wxCONTROL_CHECKABLE (Menu) item can be checked.

wxCONTROL_UNDETERMINED (Check) undetermined state.

enum wxHeaderSortIconType

Used to specify the type of sort arrow used with wxRendererNative::DrawHeaderButton().

Enumerator

wxHDR_SORT_ICON_NONE Don't draw a sort arrow.

wxHDR_SORT_ICON_UP Draw a sort arrow icon pointing up.

wxHDR_SORT_ICON_DOWN Draw a sort arrow icon pointing down.

enum wxTitleBarButton

Title bar buttons supported by wxRendererNative::DrawTitleBarBitmap().

Enumerator

wxTITLEBAR_BUTTON_CLOSE

wxTITLEBAR_BUTTON_MAXIMIZE

wxTITLEBAR_BUTTON_ICONIZE

wxTITLEBAR_BUTTON_RESTORE

wxTITLEBAR_BUTTON_HELP

22.406 interface/wx/ribbon/art.h File Reference

Classes

- class wxRibbonArtProvider

  wxRibbonArtProvider is responsible for drawing all the components of the ribbon interface.

Enumerations

- enum wxRibbonArtSetting {
  wxRIBBON_ART_TAB_SEPARATION_SIZE,
  wxRIBBON_ART_PAGE_BORDER_LEFT_SIZE,
  wxRIBBON_ART_PAGE_BORDER_TOP_SIZE,
  wxRIBBON_ART_PAGE_BORDER_RIGHT_SIZE,
  wxRIBBON_ART_PAGE_BORDER_BOTTOM_SIZE,
  wxRIBBON_ART_PANEL_X_SEPARATION_SIZE,
  wxRIBBON_ART_PANEL_Y_SEPARATION_SIZE,
  wxRIBBON_ART_TOOL_GROUP_SEPARATION_SIZE,
  wxRIBBON_ART_GALLERY_BITMAP_PADDING_LEFT_SIZE,
  wxRIBBON_ART_GALLERY_BITMAP_PADDING_RIGHT_SIZE,
  wxRIBBON_ART_GALLERY_BITMAP_PADDING_TOP_SIZE,
  wxRIBBON_ART_GALLERY_BITMAP_PADDING_BOTTOM_SIZE,
  wxRIBBON_ART_PANEL_LABEL_FONT,
  wxRIBBON_ART_BUTTON_BAR_LABEL_FONT,
  wxRIBBON_ART_TAB_LABEL_FONT,
  wxRIBBON_ART_BUTTON_BAR_LABEL_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_LABEL_DISABLED_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_HOVER_BORDER_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_TOP_GRADIENT_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_ACTIVE_BORDER_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_TOP_GRADIENT_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_COLOUR,
  wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_GALLERY_BORDER_COLOUR,
  wxRIBBON_ART_GALLERY_HOVER_BACKGROUND_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_BACKGROUND_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_FACE_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_HOVER_BACKGROUND_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_HOVER_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_HOVER_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_HOVER_FACE_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_BACKGROUND_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_FACE_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_DISABLED_BACKGROUND_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_DISABLED_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_DISABLED_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_GALLERY_BUTTON_DISABLED_FACE_COLOUR,
  wxRIBBON_ART_GALLERY_ITEM_BORDER_COLOUR,
  wxRIBBON_ART_TAB_LABEL_COLOUR,
  wxRIBBON_ART_TAB_SEPARATOR_COLOUR,
  wxRIBBON_ART_TAB_SEPARATOR_GRADIENT_COLOUR,
  wxRIBBON_ART_TAB_CTRL_BACKGROUND_COLOUR,
  wxRIBBON_ART_TAB_CTRL_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_TAB_HOVER_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_TAB_HOVER_BACKGROUND_TOP_GRADIENT_COLOUR,
  wxRIBBON_ART_TAB_HOVER_BACKGROUND_COLOUR,
  wxRIBBON_ART_TAB_HOVER_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_TOP_COLOUR,
  wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_TOP_GRADIENT_COLOUR,
  wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_COLOUR,
  wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_GRADIENT_COLOUR,
  wxRIBBON_ART_TAB_BORDER_COLOUR,
  wxRIBBON_ART_PANEL_BORDER_COLOUR,
  wxRIBBON_ART_PANEL_BORDER_GRADIENT_COLOUR,

Identifiers for common settings on ribbon art providers which can be used to tweak the appearance of the art provider.

- enum wxRibbonScrollButtonStyle {
  wxRIBBON_SCROLL_BTN_LEFT = 0,
  wxRIBBON_SCROLL_BTN_RIGHT = 1,
  wxRIBBON_SCROLL_BTN_UP = 2,
  wxRIBBON_SCROLL_BTN_DOWN = 3,
  wxRIBBON_SCROLL_BTN_DIRECTION_MASK = 3,
  wxRIBBON_SCROLL_BTN_NORMAL = 0,
  wxRIBBON_SCROLL_BTN_HOVERED = 4,
  wxRIBBON_SCROLL_BTN_ACTIVE = 8,
  wxRIBBON_SCROLL_BTN_STATE_MASK = 12,
  wxRIBBON_SCROLL_BTN_FOR_OTHER = 0,
  wxRIBBON_SCROLL_BTN_FOR_TABS = 16,
  wxRIBBON_SCROLL_BTN_FOR_PAGE = 32,
  wxRIBBON_SCROLL_BTN_FOR_MASK = 48 }

  Flags used to describe the direction, state, and/or purpose of a ribbon-style scroll button.
- enum wxRibbonButtonKind {
  wxRIBBON_BUTTON_NORMAL = 1 << 0,
  wxRIBBON_BUTTON_DROPDOWN = 1 << 1,
  wxRIBBON_BUTTON_HYBRID = wxRIBBON_BUTTON_NORMAL | wxRIBBON_BUTTON_DROPDOWN,
  wxRIBBON_BUTTON_TOGGLE = 1 << 2 }

  Buttons on a ribbon button bar and tools on a ribbon tool bar can each be one of three different kinds.

22.406.1 Enumeration Type Documentation

enum wxRibbonArtSetting

Identifiers for common settings on ribbon art providers which can be used to tweak the appearance of the art provider.

See also

wxRibbonArtProvider::GetColour()
wxRibbonArtProvider::GetFont()
wxRibbonArtProvider::GetMetric()
wxRibbonArtProvider::SetColour()
wxRibbonArtProvider::SetFont()
wxRibbonArtProvider::SetMetric()

Enumerator

wxRIBBON_ART_TAB_SEPARATION_SIZE

wxRIBBON_ART_PAGE_BORDER_LEFT_SIZE

wxRIBBON_ART_PAGE_BORDER_TOP_SIZE

wxRIBBON_ART_PAGE_BORDER_RIGHT_SIZE

wxRIBBON_ART_PAGE_BORDER_BOTTOM_SIZE

wxRIBBON_ART_PANEL_X_SEPARATION_SIZE

wxRIBBON_ART_PANEL_Y_SEPARATION_SIZE

wxRIBBON_ART_TOOL_GROUP_SEPARATION_SIZE

wxRIBBON_ART_GALLERY_BITMAP_PADDING_LEFT_SIZE

wxRIBBON_ART_GALLERY_BITMAP_PADDING_RIGHT_SIZE

wxRIBBON_ART_GALLERY_BITMAP_PADDING_TOP_SIZE

wxRIBBON_ART_GALLERY_BITMAP_PADDING_BOTTOM_SIZE

wxRIBBON_ART_PANEL_LABEL_FONT

wxRIBBON_ART_BUTTON_BAR_LABEL_FONT
wxRIBBON_ART_TAB_LABEL_FONT
wxRIBBON_ART_BUTTON_BAR_LABEL_COLOUR

	Since
wxRIBBON_ART_BUTTON_BAR_LABEL_DISABLED_COLOUR	2.9.5

wxRIBBON_ART_BUTTON_BAR_HOVER_BORDER_COLOUR
wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_TOP_GRADIENT_COLOUR
wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_COLOUR
wxRIBBON_ART_BUTTON_BAR_HOVER_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_BUTTON_BAR_ACTIVE_BORDER_COLOUR
wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_TOP_GRADIENT_COLOUR
wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_COLOUR
wxRIBBON_ART_BUTTON_BAR_ACTIVE_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_GALLERY_BORDER_COLOUR
wxRIBBON_ART_GALLERY_HOVER_BACKGROUND_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_BACKGROUND_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_FACE_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_HOVER_BACKGROUND_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_HOVER_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_HOVER_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_HOVER_FACE_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_BACKGROUND_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_ACTIVE_FACE_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_DISABLED_BACKGROUND_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_DISABLED_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_DISABLED_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_GALLERY_BUTTON_DISABLED_FACE_COLOUR
wxRIBBON_ART_GALLERY_ITEM_BORDER_COLOUR
wxRIBBON_ART_TAB_LABEL_COLOUR
wxRIBBON_ART_TAB_SEPARATOR_COLOUR
wxRIBBON_ART_TAB_SEPARATOR_GRADIENT_COLOUR
wxRIBBON_ART_TAB_CTRL_BACKGROUND_COLOUR
wxRIBBON_ART_TAB_CTRL_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_TAB_HOVER_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_TAB_HOVER_BACKGROUND_TOP_GRADIENT_COLOUR
wxRIBBON_ART_TAB_HOVER_BACKGROUND_COLOUR
wxRIBBON_ART_TAB_HOVER_BACKGROUND_GRADIENT_COLOUR
wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_TOP_COLOUR
wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_COLOUR

wxRIBBON_ART_TAB_ACTIVE_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_TAB_BORDER_COLOUR

wxRIBBON_ART_PANEL_BORDER_COLOUR

wxRIBBON_ART_PANEL_BORDER_GRADIENT_COLOUR

wxRIBBON_ART_PANEL_MINIMISED_BORDER_COLOUR

wxRIBBON_ART_PANEL_MINIMISED_BORDER_GRADIENT_COLOUR

wxRIBBON_ART_PANEL_LABEL_BACKGROUND_COLOUR

wxRIBBON_ART_PANEL_LABEL_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_PANEL_LABEL_COLOUR

wxRIBBON_ART_PANEL_HOVER_LABEL_BACKGROUND_COLOUR

wxRIBBON_ART_PANEL_HOVER_LABEL_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_PANEL_HOVER_LABEL_COLOUR

wxRIBBON_ART_PANEL_MINIMISED_LABEL_COLOUR

wxRIBBON_ART_PANEL_ACTIVE_BACKGROUND_TOP_COLOUR

wxRIBBON_ART_PANEL_ACTIVE_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_PANEL_ACTIVE_BACKGROUND_COLOUR

wxRIBBON_ART_PANEL_ACTIVE_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_PAGE_BORDER_COLOUR

wxRIBBON_ART_PAGE_BACKGROUND_TOP_COLOUR

wxRIBBON_ART_PAGE_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_PAGE_BACKGROUND_COLOUR

wxRIBBON_ART_PAGE_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_PAGE_HOVER_BACKGROUND_TOP_COLOUR

wxRIBBON_ART_PAGE_HOVER_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_PAGE_HOVER_BACKGROUND_COLOUR

wxRIBBON_ART_PAGE_HOVER_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_TOOLBAR_BORDER_COLOUR

wxRIBBON_ART_TOOLBAR_HOVER_BORDER_COLOUR

wxRIBBON_ART_TOOLBAR_FACE_COLOUR

wxRIBBON_ART_TOOL_BACKGROUND_TOP_COLOUR

wxRIBBON_ART_TOOL_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_TOOL_BACKGROUND_COLOUR

wxRIBBON_ART_TOOL_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_TOOL_HOVER_BACKGROUND_TOP_COLOUR

wxRIBBON_ART_TOOL_HOVER_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_TOOL_HOVER_BACKGROUND_COLOUR

wxRIBBON_ART_TOOL_HOVER_BACKGROUND_GRADIENT_COLOUR

wxRIBBON_ART_TOOL_ACTIVE_BACKGROUND_TOP_COLOUR

wxRIBBON_ART_TOOL_ACTIVE_BACKGROUND_TOP_GRADIENT_COLOUR

wxRIBBON_ART_TOOL_ACTIVE_BACKGROUND_COLOUR

wxRIBBON_ART_TOOL_ACTIVE_BACKGROUND_GRADIENT_COLOUR

enum wxRibbonButtonKind

Buttons on a ribbon button bar and tools on a ribbon tool bar can each be one of three different kinds.

Enumerator

wxRIBBON_BUTTON_NORMAL Normal button or tool with a clickable area which causes some generic action.

wxRIBBON_BUTTON_DROPDOWN Dropdown button or tool with a clickable area which typically causes a dropdown menu.

wxRIBBON_BUTTON_HYBRID Button or tool with two clickable areas - one which causes a dropdown menu, and one which causes a generic action.

wxRIBBON_BUTTON_TOGGLE Normal button or tool with a clickable area which toggles the button between a pressed and unpressed state.

enum wxRibbonScrollButtonStyle

Flags used to describe the direction, state, and/or purpose of a ribbon-style scroll button.

See also

wxRibbonArtProvider::DrawScrollButton()
wxRibbonArtProvider::GetScrollButtonMinimumSize()

Enumerator

wxRIBBON_SCROLL_BTN_LEFT Button will scroll to the left.

wxRIBBON_SCROLL_BTN_RIGHT Button will scroll to the right.

wxRIBBON_SCROLL_BTN_UP Button will scroll upward.

wxRIBBON_SCROLL_BTN_DOWN Button will scroll downward.

wxRIBBON_SCROLL_BTN_DIRECTION_MASK A mask to extract direction from a combination of flags.

wxRIBBON_SCROLL_BTN_NORMAL Button is not active or hovered.

wxRIBBON_SCROLL_BTN_HOVERED Button has a cursor hovering over it.

wxRIBBON_SCROLL_BTN_ACTIVE Button is being pressed.

wxRIBBON_SCROLL_BTN_STATE_MASK A mask to extract state from a combination of flags.

wxRIBBON_SCROLL_BTN_FOR_OTHER Button is not for scrolling tabs nor pages.

wxRIBBON_SCROLL_BTN_FOR_TABS Button is for scrolling tabs.

wxRIBBON_SCROLL_BTN_FOR_PAGE Button is for scrolling pages.

wxRIBBON_SCROLL_BTN_FOR_MASK A mask to extract purpose from a combination of flags.

22.407 interface/wx/ribbon/bar.h File Reference

Classes

- class wxRibbonBarEvent

  Event used to indicate various actions relating to a wxRibbonBar.
- class wxRibbonBar

  Top-level control in a ribbon user interface.

Enumerations

- enum wxRibbonDisplayMode {
  wxRIBBON_BAR_PINNED,
  wxRIBBON_BAR_MINIMIZED,
  wxRIBBON_BAR_EXPANDED }

  The possible display modes of the panel area of a wxRibbonBar widget.

22.407.1 Enumeration Type Documentation

enum wxRibbonDisplayMode

The possible display modes of the panel area of a wxRibbonBar widget.

See also

wxRibbonBar::ShowPanels()
wxRibbonBar::GetDisplayMode()

Since

2.9.5

Enumerator

wxRIBBON_BAR_PINNED The panel area is visible and pinned: it remains visible when the ribbon bar loses the focus.

wxRIBBON_BAR_MINIMIZED The panel area is hidden: only the pages tabs remain visible.

wxRIBBON_BAR_EXPANDED The panel area is visible, but not pinned: it minimizes as soon as the focus is lost.

22.408 interface/wx/ribbon/buttonbar.h File Reference

Classes

- class wxRibbonButtonBar

  A ribbon button bar is similar to a traditional toolbar.
- class wxRibbonButtonBarEvent

  Event used to indicate various actions relating to a button on a wxRibbonButtonBar.

Enumerations

- enum wxRibbonButtonBarButtonState {
  wxRIBBON_BUTTONBAR_BUTTON_SMALL = 0 << 0,
  wxRIBBON_BUTTONBAR_BUTTON_MEDIUM = 1 << 0,
  wxRIBBON_BUTTONBAR_BUTTON_LARGE = 2 << 0,
  wxRIBBON_BUTTONBAR_BUTTON_SIZE_MASK = 3 << 0,
  wxRIBBON_BUTTONBAR_BUTTON_NORMAL_HOVERED = 1 << 3,
  wxRIBBON_BUTTONBAR_BUTTON_DROPDOWN_HOVERED = 1 << 4,
  wxRIBBON_BUTTONBAR_BUTTON_HOVER_MASK = wxRIBBON_BUTTONBAR_BUTTON_NORMAL_↩
  HOVERED | wxRIBBON_BUTTONBAR_BUTTON_DROPDOWN_HOVERED,
  wxRIBBON_BUTTONBAR_BUTTON_NORMAL_ACTIVE = 1 << 5,
  wxRIBBON_BUTTONBAR_BUTTON_DROPDOWN_ACTIVE = 1 << 6,
  wxRIBBON_BUTTONBAR_BUTTON_DISABLED = 1 << 7,
  wxRIBBON_BUTTONBAR_BUTTON_TOGGLED = 1 << 8,
  wxRIBBON_BUTTONBAR_BUTTON_STATE_MASK = 0x1F8 }

Flags for button bar button size and state.

22.408.1 Enumeration Type Documentation

enum wxRibbonButtonBarButtonState

Flags for button bar button size and state.

Buttons on a ribbon button bar can each come in three sizes: small, medium, and large. In some places this is called the size class, and the term size used for the pixel width and height associated with a particular size class.

A button can also be in zero or more hovered or active states, or in the disabled state.

Enumerator

wxRIBBON_BUTTONBAR_BUTTON_SMALL Button is small (the interpretation of small is dependent upon the art provider, but it will be smaller than medium).

wxRIBBON_BUTTONBAR_BUTTON_MEDIUM Button is medium sized (the interpretation of medium is dependent upon the art provider, but it will be between small and large).

wxRIBBON_BUTTONBAR_BUTTON_LARGE Button is large (the interpretation of large is dependent upon the art provider, but it will be larger than medium).

wxRIBBON_BUTTONBAR_BUTTON_SIZE_MASK A mask to extract button size from a combination of flags.

wxRIBBON_BUTTONBAR_BUTTON_NORMAL_HOVERED The normal (non-dropdown) region of the button is being hovered over by the mouse cursor. Only applicable to normal and hybrid buttons.

wxRIBBON_BUTTONBAR_BUTTON_DROPDOWN_HOVERED The dropdown region of the button is being hovered over by the mouse cursor. Only applicable to dropdown and hybrid buttons.

wxRIBBON_BUTTONBAR_BUTTON_HOVER_MASK A mask to extract button hover state from a combination of flags.

wxRIBBON_BUTTONBAR_BUTTON_NORMAL_ACTIVE The normal (non-dropdown) region of the button is being pressed. Only applicable to normal and hybrid buttons.

wxRIBBON_BUTTONBAR_BUTTON_DROPDOWN_ACTIVE The dropdown region of the button is being pressed. Only applicable to dropdown and hybrid buttons.

wxRIBBON_BUTTONBAR_BUTTON_DISABLED The button is disabled. Hover flags may still be set when a button is disabled, but should be ignored during drawing if the button is disabled.

wxRIBBON_BUTTONBAR_BUTTON_TOGGLED The button is a toggle button which is currently in the toggled state.

wxRIBBON_BUTTONBAR_BUTTON_STATE_MASK A mask to extract button state from a combination of flags.

22.409 interface/wx/ribbon/gallery.h File Reference

Classes

- class wxRibbonGallery

  A ribbon gallery is like a wxListBox, but for bitmaps rather than strings.
- class wxRibbonGalleryEvent

Enumerations

- enum wxRibbonGalleryButtonState {
  wxRIBBON_GALLERY_BUTTON_NORMAL,
  wxRIBBON_GALLERY_BUTTON_HOVERED,
  wxRIBBON_GALLERY_BUTTON_ACTIVE,
  wxRIBBON_GALLERY_BUTTON_DISABLED }

22.409.1 Enumeration Type Documentation

enum wxRibbonGalleryButtonState

Enumerator

wxRIBBON_GALLERY_BUTTON_NORMAL

wxRIBBON_GALLERY_BUTTON_HOVERED

wxRIBBON_GALLERY_BUTTON_ACTIVE

wxRIBBON_GALLERY_BUTTON_DISABLED

22.410 interface/wx/ribbon/page.h File Reference

Classes

- class wxRibbonPage

  Container for related ribbon panels, and a tab within a ribbon bar.

22.411 interface/wx/richmsgdlg.h File Reference

Classes

- class wxRichMessageDialog

  Extension of wxMessageDialog with additional functionality.

22.412 interface/wx/richtext/richtextbuffer.h File Reference

Classes

- class wxTextAttrDimension

  A class representing a rich text dimension, including units and position.
- class wxTextAttrDimensions

  A class for left, right, top and bottom dimensions.
- class wxTextAttrSize

  A class for representing width and height.
- class wxTextAttrDimensionConverter

  A class to make it easier to convert dimensions.
- class wxTextAttrBorder

  A class representing a rich text object border.
- class wxTextAttrBorders

  A class representing a rich text object's borders.
- class wxTextAttrShadow

  A class representing a shadow.
- class wxTextBoxAttr

  A class representing the box attributes of a rich text object.
- class wxRichTextAttr

  A class representing enhanced attributes for rich text objects.
- class wxRichTextProperties

  A simple property class using wxVariants.

- class wxRichTextFontTable

  Manages quick access to a pool of fonts for rendering rich text.
- class wxRichTextRange

  This stores beginning and end positions for a range of data.
- class wxRichTextSelection

  Stores selection information.
- class wxRichTextDrawingContext

  A class for passing information to drawing and measuring functions.
- class wxRichTextObject

  This is the base for drawable rich text objects.
- class wxRichTextCompositeObject

  Objects of this class can contain other objects.
- class wxRichTextParagraphLayoutBox

  This class knows how to lay out paragraphs.
- class wxRichTextBox

  This class implements a floating or inline text box, containing paragraphs.
- class wxRichTextField

  This class implements the general concept of a field, an object that represents additional functionality such as a footnote, a bookmark, a page number, a table of contents, and so on.
- class wxRichTextFieldType

  The base class for custom field types.
- class wxRichTextFieldTypeStandard

  A field type that can handle fields with text or bitmap labels, with a small range of styles for implementing rectangular fields and fields that can be used for start and end tags.
- class wxRichTextLine

  This object represents a line in a paragraph, and stores offsets from the start of the paragraph representing the start and end positions of the line.
- class wxRichTextParagraph

  This object represents a single paragraph containing various objects such as text content, images, and further paragraph layout objects.
- class wxRichTextPlainText

  This object represents a single piece of text.
- class wxRichTextImageBlock

  This class stores information about an image, in binary in-memory form.
- class wxRichTextImage

  This class implements a graphic object.
- class wxRichTextBuffer

  This is a kind of paragraph layout box, used to represent the whole buffer.
- class wxRichTextCell

  wxRichTextCell is the cell in a table, in which the user can type.
- class wxRichTextTable

  wxRichTextTable represents a table with arbitrary columns and rows.
- class wxRichTextTableBlock

  Stores the coordinates for a block of cells.
- class wxRichTextObjectAddress

  A class for specifying an object anywhere in an object hierarchy, without using a pointer, necessary since wxRTC commands may delete and recreate sub-objects so physical object addresses change.
- class wxRichTextCommand

  Implements a command on the undo/redo stack.
- class wxRichTextAction

  Implements a part of a command.
- class wxRichTextFileHandler

The base class for file handlers.

- class wxRichTextPlainTextHandler

  Implements saving a buffer to plain text.
- class wxRichTextDrawingHandler

  The base class for custom drawing handlers.
- class wxRichTextBufferDataObject

  Implements a rich text data object for clipboard transfer.
- class wxRichTextRenderer

  This class isolates some common drawing functionality.
- class wxRichTextStdRenderer

  The standard renderer for drawing bullets.

Macros

- #define wxRICHTEXT_FIXED_WIDTH 0x01

  Flags determining the available space, passed to Layout.
- #define wxRICHTEXT_FIXED_HEIGHT 0x02
- #define wxRICHTEXT_VARIABLE_WIDTH 0x04
- #define wxRICHTEXT_VARIABLE_HEIGHT 0x08
- #define wxRICHTEXT_LAYOUT_SPECIFIED_RECT 0x10
- #define wxRICHTEXT_DRAW_IGNORE_CACHE 0x01

  Flags to pass to Draw.
- #define wxRICHTEXT_DRAW_SELECTED 0x02
- #define wxRICHTEXT_DRAW_PRINT 0x04
- #define wxRICHTEXT_DRAW_GUIDELINES 0x08
- #define wxRICHTEXT_FORMATTED 0x01

  Flags for GetRangeSize.
- #define wxRICHTEXT_UNFORMATTED 0x02
- #define wxRICHTEXT_CACHE_SIZE 0x04
- #define wxRICHTEXT_HEIGHT_ONLY 0x08
- #define wxRICHTEXT_SETSTYLE_NONE 0x00

  Flags for SetStyle/SetListStyle.
- #define wxRICHTEXT_SETSTYLE_WITH_UNDO 0x01
- #define wxRICHTEXT_SETSTYLE_OPTIMIZE 0x02
- #define wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY 0x04
- #define wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY 0x08
- #define wxRICHTEXT_SETSTYLE_RENUMBER 0x10
- #define wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL 0x20
- #define wxRICHTEXT_SETSTYLE_RESET 0x40
- #define wxRICHTEXT_SETSTYLE_REMOVE 0x80
- #define wxRICHTEXT_SETPROPERTIES_NONE 0x00

  Flags for SetProperties.
- #define wxRICHTEXT_SETPROPERTIES_WITH_UNDO 0x01
- #define wxRICHTEXT_SETPROPERTIES_PARAGRAPHS_ONLY 0x02
- #define wxRICHTEXT_SETPROPERTIES_CHARACTERS_ONLY 0x04
- #define wxRICHTEXT_SETPROPERTIES_RESET 0x08
- #define wxRICHTEXT_SETPROPERTIES_REMOVE 0x10
- #define wxRICHTEXT_INSERT_NONE 0x00

  Flags for object insertion.
- #define wxRICHTEXT_INSERT_WITH_PREVIOUS_PARAGRAPH_STYLE 0x01
- #define wxRICHTEXT_INSERT_INTERACTIVE 0x02
- #define wxTEXT_ATTR_KEEP_FIRST_PARA_STYLE 0x10000000

- #define wxSCRIPT_MUL_FACTOR 1.5

  Default superscript/subscript font multiplication factor.
- #define wxRICHTEXT_ALL wxRichTextRange(-2, -2)
- #define wxRICHTEXT_NONE wxRichTextRange(-1, -1)
- #define wxRICHTEXT_NO_SELECTION wxRichTextRange(-2, -2)
- #define wxRICHTEXT_HANDLER_INCLUDE_STYLESHEET 0x0001
- #define wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_MEMORY 0x0010
- #define wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_FILES 0x0020
- #define wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_BASE64 0x0040
- #define wxRICHTEXT_HANDLER_NO_HEADER_FOOTER 0x0080
- #define wxRICHTEXT_HANDLER_CONVERT_FACENAMES 0x0100

Typedefs

- typedef unsigned short wxTextAttrDimensionFlags

  The type for wxTextAttrDimension flags.

Enumerations

- enum wxRichTextFileType {
  wxRICHTEXT_TYPE_ANY = 0,
  wxRICHTEXT_TYPE_TEXT,
  wxRICHTEXT_TYPE_XML,
  wxRICHTEXT_TYPE_HTML,
  wxRICHTEXT_TYPE_RTF,
  wxRICHTEXT_TYPE_PDF }

  File types in wxRichText context.
- enum wxRichTextHitTestFlags {
  wxRICHTEXT_HITTEST_NONE = 0x01,
  wxRICHTEXT_HITTEST_BEFORE = 0x02,
  wxRICHTEXT_HITTEST_AFTER = 0x04,
  wxRICHTEXT_HITTEST_ON = 0x08,
  wxRICHTEXT_HITTEST_OUTSIDE = 0x10,
  wxRICHTEXT_HITTEST_NO_NESTED_OBJECTS = 0x20,
  wxRICHTEXT_HITTEST_NO_FLOATING_OBJECTS = 0x40,
  wxRICHTEXT_HITTEST_HONOUR_ATOMIC = 0x80 }

  Flags returned from hit-testing, or passed to hit-test function.
- enum wxTextBoxAttrFlags {
  wxTEXT_BOX_ATTR_FLOAT = 0x00000001,
  wxTEXT_BOX_ATTR_CLEAR = 0x00000002,
  wxTEXT_BOX_ATTR_COLLAPSE_BORDERS = 0x00000004,
  wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT = 0x00000008,
  wxTEXT_BOX_ATTR_BOX_STYLE_NAME = 0x00000010,
  wxTEXT_BOX_ATTR_WHITESPACE = 0x00000020,
  wxTEXT_BOX_ATTR_CORNER_RADIUS = 0x00000040 }

  Miscellaneous text box flags.
- enum wxTextAttrValueFlags {
  wxTEXT_ATTR_VALUE_VALID = 0x1000,
  wxTEXT_ATTR_VALUE_VALID_MASK = 0x1000 }

  Whether a value is present, used in dimension flags.

- enum wxTextAttrUnits {
  wxTEXT_ATTR_UNITS_TENTHS_MM = 0x0001,
  wxTEXT_ATTR_UNITS_PIXELS = 0x0002,
  wxTEXT_ATTR_UNITS_PERCENTAGE = 0x0004,
  wxTEXT_ATTR_UNITS_POINTS = 0x0008,
  wxTEXT_ATTR_UNITS_HUNDREDTHS_POINT = 0x0100,
  wxTEXT_ATTR_UNITS_MASK = 0x010F }

  Units, included in the dimension value.
- enum wxTextBoxAttrPosition {
  wxTEXT_BOX_ATTR_POSITION_STATIC = 0x0000,
  wxTEXT_BOX_ATTR_POSITION_RELATIVE = 0x0010,
  wxTEXT_BOX_ATTR_POSITION_ABSOLUTE = 0x0020,
  wxTEXT_BOX_ATTR_POSITION_FIXED = 0x0040,
  wxTEXT_BOX_ATTR_POSITION_MASK = 0x00F0 }

  Position alternatives, included in the dimension flags.
- enum wxTextAttrBorderStyle {
  wxTEXT_BOX_ATTR_BORDER_NONE = 0,
  wxTEXT_BOX_ATTR_BORDER_SOLID = 1,
  wxTEXT_BOX_ATTR_BORDER_DOTTED = 2,
  wxTEXT_BOX_ATTR_BORDER_DASHED = 3,
  wxTEXT_BOX_ATTR_BORDER_DOUBLE = 4,
  wxTEXT_BOX_ATTR_BORDER_GROOVE = 5,
  wxTEXT_BOX_ATTR_BORDER_RIDGE = 6,
  wxTEXT_BOX_ATTR_BORDER_INSET = 7,
  wxTEXT_BOX_ATTR_BORDER_OUTSET = 8 }

  Border styles, used with wxTextAttrBorder.
- enum wxTextAttrBorderFlags {
  wxTEXT_BOX_ATTR_BORDER_STYLE = 0x0001,
  wxTEXT_BOX_ATTR_BORDER_COLOUR = 0x0002 }

  Border style presence flags, used with wxTextAttrBorder.
- enum wxTextAttrBorderWidth {
  wxTEXT_BOX_ATTR_BORDER_THIN = -1,
  wxTEXT_BOX_ATTR_BORDER_MEDIUM = -2,
  wxTEXT_BOX_ATTR_BORDER_THICK = -3 }

  Border width symbols for qualitative widths, used with wxTextAttrBorder.
- enum wxTextBoxAttrFloatStyle {
  wxTEXT_BOX_ATTR_FLOAT_NONE = 0,
  wxTEXT_BOX_ATTR_FLOAT_LEFT = 1,
  wxTEXT_BOX_ATTR_FLOAT_RIGHT = 2 }

  Float styles.
- enum wxTextBoxAttrClearStyle {
  wxTEXT_BOX_ATTR_CLEAR_NONE = 0,
  wxTEXT_BOX_ATTR_CLEAR_LEFT = 1,
  wxTEXT_BOX_ATTR_CLEAR_RIGHT = 2,
  wxTEXT_BOX_ATTR_CLEAR_BOTH = 3 }

  Clear styles.
- enum wxTextBoxAttrCollapseMode {
  wxTEXT_BOX_ATTR_COLLAPSE_NONE = 0,
  wxTEXT_BOX_ATTR_COLLAPSE_FULL = 1 }

  Collapse mode styles.
- enum wxTextBoxAttrVerticalAlignment {
  wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_NONE = 0,
  wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_TOP = 1,
  wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_CENTRE = 2,
  wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_BOTTOM = 3 }

  Vertical alignment values.

- enum wxTextBoxAttrWhitespaceMode {
  wxTEXT_BOX_ATTR_WHITESPACE_NONE = 0,
  wxTEXT_BOX_ATTR_WHITESPACE_NORMAL = 1,
  wxTEXT_BOX_ATTR_WHITESPACE_NO_WRAP = 2,
  wxTEXT_BOX_ATTR_WHITESPACE_PREFORMATTED = 3,
  wxTEXT_BOX_ATTR_WHITESPACE_PREFORMATTED_LINE = 4,
  wxTEXT_BOX_ATTR_WHITESPACE_PREFORMATTED_WRAP = 5 }

  Whitespace values mirroring the CSS white-space attribute.
- enum wxRichTextCommandId {
  wxRICHTEXT_INSERT,
  wxRICHTEXT_DELETE,
  wxRICHTEXT_CHANGE_ATTRIBUTES,
  wxRICHTEXT_CHANGE_STYLE,
  wxRICHTEXT_CHANGE_OBJECT }

  The command identifiers for Do/Undo.

Functions

- bool wxRichTextHasStyle (int flags, int style)
- bool wxTextAttrEq (const wxRichTextAttr &attr1, const wxRichTextAttr &attr2)

  Compare two attribute objects.
- bool wxRichTextApplyStyle (wxRichTextAttr &destStyle, const wxRichTextAttr &style, wxRichTextAttr *compareWith=NULL)

  Apply one style to another.
- bool wxRichTextRemoveStyle (wxRichTextAttr &destStyle, const wxRichTextAttr &style)
- bool wxRichTextCombineBitlists (int &valueA, int valueB, int &flagsA, int flagsB)

  Combine two bitlists.
- bool wxRichTextBitlistsEqPartial (int valueA, int valueB, int flags)

  Compare two bitlists.
- bool wxRichTextSplitParaCharStyles (const wxRichTextAttr &style, wxRichTextAttr &parStyle, wxRichTextAttr &charStyle)

  Split into paragraph and character styles.
- bool wxRichTextTabsEq (const wxArrayInt &tabs1, const wxArrayInt &tabs2)

  Compare tabs.
- wxString wxRichTextDecimalToRoman (long n)

  Convert a decimal to Roman numerals.
- void wxTextAttrCollectCommonAttributes (wxTextAttr &currentStyle, const wxTextAttr &attr, wxTextAttr &clashingAttr, wxTextAttr &absentAttr)
- void wxRichTextModuleInit ()

Variables

- const wxChar wxRichTextLineBreakChar

  The line break character that can be embedded in content.

22.412.1 Macro Definition Documentation

#define wxRICHTEXT_ALL wxRichTextRange(-2, -2)

#define wxRICHTEXT_CACHE_SIZE 0x04

#define wxRICHTEXT_DRAW_GUIDELINES 0x08

#define wxRICHTEXT_DRAW_IGNORE_CACHE 0x01

Flags to pass to Draw.

#define wxRICHTEXT_DRAW_PRINT 0x04

#define wxRICHTEXT_DRAW_SELECTED 0x02

#define wxRICHTEXT_FIXED_HEIGHT 0x02

#define wxRICHTEXT_FIXED_WIDTH 0x01

Flags determining the available space, passed to Layout.

#define wxRICHTEXT_FORMATTED 0x01

Flags for GetRangeSize.

#define wxRICHTEXT_HANDLER_CONVERT_FACENAMES 0x0100

#define wxRICHTEXT_HANDLER_INCLUDE_STYLESHEET 0x0001

Handler flags

#define wxRICHTEXT_HANDLER_NO_HEADER_FOOTER 0x0080

#define wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_BASE64 0x0040

#define wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_FILES 0x0020

#define wxRICHTEXT_HANDLER_SAVE_IMAGES_TO_MEMORY 0x0010

#define wxRICHTEXT_HEIGHT_ONLY 0x08

#define wxRICHTEXT_INSERT_INTERACTIVE 0x02

#define wxRICHTEXT_INSERT_NONE 0x00

Flags for object insertion.

#define wxRICHTEXT_INSERT_WITH_PREVIOUS_PARAGRAPH_STYLE 0x01

#define wxRICHTEXT_LAYOUT_SPECIFIED_RECT 0x10

#define wxRICHTEXT_NO_SELECTION **wxRichTextRange**(-2, -2)

#define wxRICHTEXT_NONE **wxRichTextRange**(-1, -1)

#define wxRICHTEXT_SETPROPERTIES_CHARACTERS_ONLY 0x04

#define wxRICHTEXT_SETPROPERTIES_NONE 0x00

Flags for SetProperties.

#define wxRICHTEXT_SETPROPERTIES_PARAGRAPHS_ONLY 0x02

#define wxRICHTEXT_SETPROPERTIES_REMOVE 0x10

#define wxRICHTEXT_SETPROPERTIES_RESET 0x08

#define wxRICHTEXT_SETPROPERTIES_WITH_UNDO 0x01

#define wxRICHTEXT_SETSTYLE_CHARACTERS_ONLY 0x08

#define wxRICHTEXT_SETSTYLE_NONE 0x00

Flags for SetStyle/SetListStyle.

#define wxRICHTEXT_SETSTYLE_OPTIMIZE 0x02

#define wxRICHTEXT_SETSTYLE_PARAGRAPHS_ONLY 0x04

#define wxRICHTEXT_SETSTYLE_REMOVE 0x80

#define wxRICHTEXT_SETSTYLE_RENUMBER 0x10

#define wxRICHTEXT_SETSTYLE_RESET 0x40

#define wxRICHTEXT_SETSTYLE_SPECIFY_LEVEL 0x20

#define wxRICHTEXT_SETSTYLE_WITH_UNDO 0x01

#define wxRICHTEXT_UNFORMATTED 0x02

#define wxRICHTEXT_VARIABLE_HEIGHT 0x08

#define wxRICHTEXT_VARIABLE_WIDTH 0x04

#define wxSCRIPT_MUL_FACTOR 1.5

Default superscript/subscript font multiplication factor.

#define wxTEXT_ATTR_KEEP_FIRST_PARA_STYLE 0x10000000

22.412.2 Typedef Documentation

typedef unsigned short **wxTextAttrDimensionFlags**

The type for wxTextAttrDimension flags.

22.412.3 Enumeration Type Documentation

enum **wxRichTextCommandId**

The command identifiers for Do/Undo.

Enumerator

wxRICHTEXT_INSERT

wxRICHTEXT_DELETE

wxRICHTEXT_CHANGE_ATTRIBUTES

wxRICHTEXT_CHANGE_STYLE

wxRICHTEXT_CHANGE_OBJECT

enum wxRichTextFileType

File types in wxRichText context.

Enumerator

wxRICHTEXT_TYPE_ANY

wxRICHTEXT_TYPE_TEXT

wxRICHTEXT_TYPE_XML

wxRICHTEXT_TYPE_HTML

wxRICHTEXT_TYPE_RTF

wxRICHTEXT_TYPE_PDF

enum wxRichTextHitTestFlags

Flags returned from hit-testing, or passed to hit-test function.

Enumerator

wxRICHTEXT_HITTEST_NONE

wxRICHTEXT_HITTEST_BEFORE

wxRICHTEXT_HITTEST_AFTER

wxRICHTEXT_HITTEST_ON

wxRICHTEXT_HITTEST_OUTSIDE

wxRICHTEXT_HITTEST_NO_NESTED_OBJECTS

wxRICHTEXT_HITTEST_NO_FLOATING_OBJECTS

wxRICHTEXT_HITTEST_HONOUR_ATOMIC

enum wxTextAttrBorderFlags

Border style presence flags, used with wxTextAttrBorder.

Enumerator

wxTEXT_BOX_ATTR_BORDER_STYLE

wxTEXT_BOX_ATTR_BORDER_COLOUR

enum **wxTextAttrBorderStyle**

Border styles, used with wxTextAttrBorder.

Enumerator

wxTEXT_BOX_ATTR_BORDER_NONE

wxTEXT_BOX_ATTR_BORDER_SOLID

wxTEXT_BOX_ATTR_BORDER_DOTTED

wxTEXT_BOX_ATTR_BORDER_DASHED

wxTEXT_BOX_ATTR_BORDER_DOUBLE

wxTEXT_BOX_ATTR_BORDER_GROOVE

wxTEXT_BOX_ATTR_BORDER_RIDGE

wxTEXT_BOX_ATTR_BORDER_INSET

wxTEXT_BOX_ATTR_BORDER_OUTSET

enum **wxTextAttrBorderWidth**

Border width symbols for qualitative widths, used with wxTextAttrBorder.

Enumerator

wxTEXT_BOX_ATTR_BORDER_THIN

wxTEXT_BOX_ATTR_BORDER_MEDIUM

wxTEXT_BOX_ATTR_BORDER_THICK

enum **wxTextAttrUnits**

Units, included in the dimension value.

Enumerator

wxTEXT_ATTR_UNITS_TENTHS_MM

wxTEXT_ATTR_UNITS_PIXELS

wxTEXT_ATTR_UNITS_PERCENTAGE

wxTEXT_ATTR_UNITS_POINTS

wxTEXT_ATTR_UNITS_HUNDREDTHS_POINT

wxTEXT_ATTR_UNITS_MASK

enum **wxTextAttrValueFlags**

Whether a value is present, used in dimension flags.

Enumerator

wxTEXT_ATTR_VALUE_VALID

wxTEXT_ATTR_VALUE_VALID_MASK

enum wxTextBoxAttrClearStyle

Clear styles.

Enumerator

wxTEXT_BOX_ATTR_CLEAR_NONE
wxTEXT_BOX_ATTR_CLEAR_LEFT
wxTEXT_BOX_ATTR_CLEAR_RIGHT
wxTEXT_BOX_ATTR_CLEAR_BOTH

enum wxTextBoxAttrCollapseMode

Collapse mode styles.

Enumerator

wxTEXT_BOX_ATTR_COLLAPSE_NONE
wxTEXT_BOX_ATTR_COLLAPSE_FULL

enum wxTextBoxAttrFlags

Miscellaneous text box flags.

Enumerator

wxTEXT_BOX_ATTR_FLOAT
wxTEXT_BOX_ATTR_CLEAR
wxTEXT_BOX_ATTR_COLLAPSE_BORDERS
wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT
wxTEXT_BOX_ATTR_BOX_STYLE_NAME
wxTEXT_BOX_ATTR_WHITESPACE
wxTEXT_BOX_ATTR_CORNER_RADIUS

enum wxTextBoxAttrFloatStyle

Float styles.

Enumerator

wxTEXT_BOX_ATTR_FLOAT_NONE
wxTEXT_BOX_ATTR_FLOAT_LEFT
wxTEXT_BOX_ATTR_FLOAT_RIGHT

enum wxTextBoxAttrPosition

Position alternatives, included in the dimension flags.

Enumerator

wxTEXT_BOX_ATTR_POSITION_STATIC
wxTEXT_BOX_ATTR_POSITION_RELATIVE
wxTEXT_BOX_ATTR_POSITION_ABSOLUTE
wxTEXT_BOX_ATTR_POSITION_FIXED
wxTEXT_BOX_ATTR_POSITION_MASK

enum wxTextBoxAttrVerticalAlignment

Vertical alignment values.

Enumerator

wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_NONE

wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_TOP

wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_CENTRE

wxTEXT_BOX_ATTR_VERTICAL_ALIGNMENT_BOTTOM

enum wxTextBoxAttrWhitespaceMode

Whitespace values mirroring the CSS white-space attribute.

Only wxTEXT_BOX_ATTR_WHITESPACE_NO_WRAP is currently implemented, in table cells.

Enumerator

wxTEXT_BOX_ATTR_WHITESPACE_NONE

wxTEXT_BOX_ATTR_WHITESPACE_NORMAL

wxTEXT_BOX_ATTR_WHITESPACE_NO_WRAP

wxTEXT_BOX_ATTR_WHITESPACE_PREFORMATTED

wxTEXT_BOX_ATTR_WHITESPACE_PREFORMATTED_LINE

wxTEXT_BOX_ATTR_WHITESPACE_PREFORMATTED_WRAP

22.412.4 Function Documentation

bool wxRichTextApplyStyle (wxRichTextAttr & *destStyle,* const wxRichTextAttr & *style,* wxRichTextAttr * *compareWith* = NULL)

Apply one style to another.

bool wxRichTextBitlistsEqPartial (int *valueA,* int *valueB,* int *flags*)

Compare two bitlists.

bool wxRichTextCombineBitlists (int & *valueA,* int *valueB,* int & *flagsA,* int *flagsB*)

Combine two bitlists.

wxString wxRichTextDecimalToRoman (long *n*)

Convert a decimal to Roman numerals.

bool wxRichTextHasStyle (int *flags,* int *style*) [inline]

Utilities

void wxRichTextModuleInit ( )

bool wxRichTextRemoveStyle (wxRichTextAttr & *destStyle,* const wxRichTextAttr & *style*)

bool wxRichTextSplitParaCharStyles (const wxRichTextAttr & *style,* wxRichTextAttr & *parStyle,* wxRichTextAttr & *charStyle*)

Split into paragraph and character styles.

bool wxRichTextTabsEq (const wxArrayInt & *tabs1,* const wxArrayInt & *tabs2*)

Compare tabs.

void wxTextAttrCollectCommonAttributes (wxTextAttr & *currentStyle,* const wxTextAttr & *attr,* wxTextAttr & *clashingAttr,* wxTextAttr & *absentAttr*)

bool wxTextAttrEq (const wxRichTextAttr & *attr1,* const wxRichTextAttr & *attr2*)

Compare two attribute objects.

22.412.5 Variable Documentation

const wxChar wxRichTextLineBreakChar

The line break character that can be embedded in content.

22.413 interface/wx/richtext/richtextformatdlg.h File Reference

Classes

- class wxRichTextFormattingDialogFactory

  This class provides pages for wxRichTextFormattingDialog, and allows other customization of the dialog.
- class wxRichTextFormattingDialog

  This dialog allows the user to edit a character and/or paragraph style.

Macros

- #define wxRICHTEXT_FORMAT_STYLE_EDITOR 0x0001
- #define wxRICHTEXT_FORMAT_FONT 0x0002
- #define wxRICHTEXT_FORMAT_TABS 0x0004
- #define wxRICHTEXT_FORMAT_BULLETS 0x0008
- #define wxRICHTEXT_FORMAT_INDENTS_SPACING 0x0010

22.413.1 Macro Definition Documentation

#define wxRICHTEXT_FORMAT_BULLETS 0x0008

#define wxRICHTEXT_FORMAT_FONT 0x0002

#define wxRICHTEXT_FORMAT_INDENTS_SPACING 0x0010

#define wxRICHTEXT_FORMAT_STYLE_EDITOR 0x0001

#define wxRICHTEXT_FORMAT_TABS 0x0004

22.414 interface/wx/richtext/richtexthtml.h File Reference

Classes

- class wxRichTextHTMLHandler

  Handles HTML output (only) for wxRichTextCtrl content.

22.415 interface/wx/richtext/richtextprint.h File Reference

Classes

- class wxRichTextHeaderFooterData

  This class represents header and footer data to be passed to the wxRichTextPrinting and wxRichTextPrintout classes.
- class wxRichTextPrintout

  This class implements print layout for wxRichTextBuffer.
- class wxRichTextPrinting

  This class provides a simple interface for performing wxRichTextBuffer printing and previewing.

Enumerations

- enum wxRichTextOddEvenPage {
  wxRICHTEXT_PAGE_ODD,
  wxRICHTEXT_PAGE_EVEN,
  wxRICHTEXT_PAGE_ALL }

  These are the header and footer page identifiers, passed to functions such as wxRichTextHeaderFooterData::Set↩ FooterText to specify the odd or even page for the text.
- enum wxRichTextPageLocation {
  wxRICHTEXT_PAGE_LEFT,
  wxRICHTEXT_PAGE_CENTRE,
  wxRICHTEXT_PAGE_RIGHT }

  These are the location identifiers for passing to functions such as wxRichTextHeaderFooterData::SetFooterText(), to specify whether the text is on the left, centre or right of the page.

22.415.1 Enumeration Type Documentation

enum wxRichTextOddEvenPage

These are the header and footer page identifiers, passed to functions such as wxRichTextHeaderFooterData::Set↩ FooterText to specify the odd or even page for the text.

Enumerator

wxRICHTEXT_PAGE_ODD

wxRICHTEXT_PAGE_EVEN

wxRICHTEXT_PAGE_ALL

enum wxRichTextPageLocation

These are the location identifiers for passing to functions such as wxRichTextHeaderFooterData::SetFooterText(), to specify whether the text is on the left, centre or right of the page.

Enumerator

wxRICHTEXT_PAGE_LEFT

wxRICHTEXT_PAGE_CENTRE

wxRICHTEXT_PAGE_RIGHT

22.416 interface/wx/richtext/richtextstyledlg.h File Reference

Classes

- class wxRichTextStyleOrganiserDialog

  This class shows a style sheet and allows the user to edit, add and remove styles.

Macros

- #define wxRICHTEXT_ORGANISER_DELETE_STYLES 0x0001
- #define wxRICHTEXT_ORGANISER_CREATE_STYLES 0x0002
- #define wxRICHTEXT_ORGANISER_APPLY_STYLES 0x0004
- #define wxRICHTEXT_ORGANISER_EDIT_STYLES 0x0008
- #define wxRICHTEXT_ORGANISER_RENAME_STYLES 0x0010
- #define wxRICHTEXT_ORGANISER_OK_CANCEL 0x0020
- #define wxRICHTEXT_ORGANISER_RENUMBER 0x0040
- #define wxRICHTEXT_ORGANISER_SHOW_CHARACTER 0x0100
- #define wxRICHTEXT_ORGANISER_SHOW_PARAGRAPH 0x0200
- #define wxRICHTEXT_ORGANISER_SHOW_LIST 0x0400
- #define wxRICHTEXT_ORGANISER_SHOW_BOX 0x0800
- #define wxRICHTEXT_ORGANISER_SHOW_ALL 0x1000
- #define wxRICHTEXT_ORGANISER_ORGANISE (wxRICHTEXT_ORGANISER_SHOW_ALL|wxRICHTE↩XT_ORGANISER_DELETE_STYLES|wxRICHTEXT_ORGANISER_CREATE_STYLES|wxRICHTEXT_O↩RGANISER_APPLY_STYLES|wxRICHTEXT_ORGANISER_EDIT_STYLES|wxRICHTEXT_ORGANISER↩_RENAME_STYLES)
- #define wxRICHTEXT_ORGANISER_BROWSE (wxRICHTEXT_ORGANISER_SHOW_ALL|wxRICHTEX↩T_ORGANISER_OK_CANCEL)
- #define wxRICHTEXT_ORGANISER_BROWSE_NUMBERING (wxRICHTEXT_ORGANISER_SHOW_LI↩ST|wxRICHTEXT_ORGANISER_OK_CANCEL|wxRICHTEXT_ORGANISER_RENUMBER)

22.416.1 Macro Definition Documentation

#define wxRICHTEXT_ORGANISER_APPLY_STYLES 0x0004

#define wxRICHTEXT_ORGANISER_BROWSE (wxRICHTEXT_ORGANISER_SHOW_ALL|wxRICHTEXT_ORGANIS↩ER_OK_CANCEL)

#define wxRICHTEXT_ORGANISER_BROWSE_NUMBERING (wxRICHTEXT_ORGANISER_SHOW_LIST|wxRICHTEX↩T_ORGANISER_OK_CANCEL|wxRICHTEXT_ORGANISER_RENUMBER)

#define wxRICHTEXT_ORGANISER_CREATE_STYLES 0x0002

#define wxRICHTEXT_ORGANISER_DELETE_STYLES 0x0001

Flags for specifying permitted operations

#define wxRICHTEXT_ORGANISER_EDIT_STYLES 0x0008

#define wxRICHTEXT_ORGANISER_OK_CANCEL 0x0020

#define wxRICHTEXT_ORGANISER_ORGANISE (wxRICHTEXT_ORGANISER_SHOW_ALL|wxRICHTEXT_ORGANISER_DELETE_STYLES|wxRICHTEXT_ORGANISER_CREATE_STYLES|wxRICHTEXT_ORGANISER_APPLY_STYLES|wxRICHTEXT_ORGANISER_EDIT_STYLES|wxRICHTEXT_ORGANISER_RENAME_STYLES)

#define wxRICHTEXT_ORGANISER_RENAME_STYLES 0x0010

#define wxRICHTEXT_ORGANISER_RENUMBER 0x0040

#define wxRICHTEXT_ORGANISER_SHOW_ALL 0x1000

#define wxRICHTEXT_ORGANISER_SHOW_BOX 0x0800

#define wxRICHTEXT_ORGANISER_SHOW_CHARACTER 0x0100

#define wxRICHTEXT_ORGANISER_SHOW_LIST 0x0400

#define wxRICHTEXT_ORGANISER_SHOW_PARAGRAPH 0x0200

22.417 interface/wx/richtext/richtextstyles.h File Reference

Classes

- class wxRichTextStyleListCtrl

  This class incorporates a wxRichTextStyleListBox and a choice control that allows the user to select the category of style to view.

- class wxRichTextStyleDefinition

  This is a base class for paragraph and character styles.

- class wxRichTextParagraphStyleDefinition

  This class represents a paragraph style definition, usually added to a wxRichTextStyleSheet.

- class wxRichTextStyleListBox

  This is a listbox that can display the styles in a wxRichTextStyleSheet, and apply the selection to an associated wxRichTextCtrl.

- class wxRichTextStyleComboCtrl

  This is a combo control that can display the styles in a wxRichTextStyleSheet, and apply the selection to an associated wxRichTextCtrl.

- class wxRichTextCharacterStyleDefinition

  This class represents a character style definition, usually added to a wxRichTextStyleSheet.

- class wxRichTextListStyleDefinition

  This class represents a list style definition, usually added to a wxRichTextStyleSheet.

- class wxRichTextStyleSheet

  A style sheet contains named paragraph and character styles that make it easy for a user to apply combinations of attributes to a wxRichTextCtrl.

22.418 interface/wx/richtext/richtextsymboldlg.h File Reference

Classes

- class wxSymbolPickerDialog

  wxSymbolPickerDialog presents the user with a choice of fonts and a grid of available characters.

22.419 interface/wx/richtext/richtextxml.h File Reference

Classes

- class wxRichTextXMLHandler

  A handler for loading and saving content in an XML format specific to wxRichTextBuffer.

22.420 interface/wx/richtooltip.h File Reference

Classes

- class wxRichToolTip

  Allows to show a tool tip with more customizations than wxToolTip.

Enumerations

- enum wxTipKind {
  wxTipKind_None,
  wxTipKind_TopLeft,
  wxTipKind_Top,
  wxTipKind_TopRight,
  wxTipKind_BottomLeft,
  wxTipKind_Bottom,
  wxTipKind_BottomRight,
  wxTipKind_Auto }

  Support tip kinds for wxRichToolTip.

22.420.1 Enumeration Type Documentation

enum wxTipKind

Support tip kinds for wxRichToolTip.

This enum describes the kind of the tip shown which combines both the tip position and appearance because the two are related (when the tip is positioned asymmetrically, a right handed triangle is used but an equilateral one when it's in the middle of a side).

Automatic selects the tip appearance best suited for the current platform and the position best suited for the window the tooltip is shown for, i.e. chosen in such a way that the tooltip is always fully on screen.

Other values describe the position of the tooltip itself, not the window it relates to. E.g. wxTipKind_Top places the tip on the top of the tooltip and so the tooltip itself is located beneath its associated window.

Enumerator

wxTipKind_None Don't show any tip, the tooltip will be (roughly) rectangular.

wxTipKind_TopLeft Show a right triangle tip in the top left corner of the tooltip.

wxTipKind_Top Show an equilateral triangle tip in the middle of the tooltip top side.

wxTipKind_TopRight Show a right triangle tip in the top right corner of the tooltip.

wxTipKind_BottomLeft Show a right triangle tip in the bottom left corner of the tooltip.

wxTipKind_Bottom Show an equilateral triangle tip in the middle of the tooltip bottom side.

wxTipKind_BottomRight Show a right triangle tip in the bottom right corner of the tooltip.

wxTipKind_Auto Choose the appropriate tip shape and position automatically. This is the default and shouldn't normally need to be changed.

Notice that currently wxTipKind_Top or wxTipKind_Bottom are used under Mac while one of the other four values is selected for the other platforms.

22.421 interface/wx/sashwin.h File Reference

Classes

- class wxSashWindow

  wxSashWindow allows any of its edges to have a sash which can be dragged to resize the window.
- class wxSashEvent

  A sash event is sent when the sash of a wxSashWindow has been dragged by the user.

Macros

- #define wxSW_NOBORDER 0x0000

  wxSashWindow flags
- #define wxSW_BORDER 0x0020
- #define wxSW_3DSASH 0x0040
- #define wxSW_3DBORDER 0x0080
- #define wxSW_3D (wxSW_3DSASH | wxSW_3DBORDER)

Enumerations

- enum wxSashEdgePosition {
  wxSASH_TOP = 0,
  wxSASH_RIGHT,
  wxSASH_BOTTOM,
  wxSASH_LEFT,
  wxSASH_NONE = 100 }

  See wxSashWindow.
- enum wxSashDragStatus {
  wxSASH_STATUS_OK,
  wxSASH_STATUS_OUT_OF_RANGE }

  See wxSashEvent.

Variables

- wxEventType wxEVT_SASH_DRAGGED

22.421.1 Macro Definition Documentation

#define wxSW_3D (wxSW_3DSASH | wxSW_3DBORDER)

#define wxSW_3DBORDER 0x0080

#define wxSW_3DSASH 0x0040

#define wxSW_BORDER 0x0020

#define wxSW_NOBORDER 0x0000

wxSashWindow flags

22.421.2 Enumeration Type Documentation

enum wxSashDragStatus

See wxSashEvent.

Enumerator

wxSASH_STATUS_OK

wxSASH_STATUS_OUT_OF_RANGE

enum wxSashEdgePosition

See wxSashWindow.

Enumerator

wxSASH_TOP

wxSASH_RIGHT

wxSASH_BOTTOM

wxSASH_LEFT

wxSASH_NONE

22.421.3 Variable Documentation

wxEventType wxEVT_SASH_DRAGGED

22.422 interface/wx/sckipc.h File Reference

Classes

- class wxTCPServer

  A wxTCPServer object represents the server part of a client-server conversation.
- class wxTCPClient

  A wxTCPClient object represents the client part of a client-server conversation.
- class wxTCPConnection

  A wxTCPClient object represents the connection between a client and a server.

Enumerations

- enum wxIPCFormat {
  wxIPC_INVALID = 0,
  wxIPC_TEXT = 1,
  wxIPC_BITMAP = 2,
  wxIPC_METAFILE = 3,
  wxIPC_SYLK = 4,
  wxIPC_DIF = 5,
  wxIPC_TIFF = 6,
  wxIPC_OEMTEXT = 7,
  wxIPC_DIB = 8,
  wxIPC_PALETTE = 9,
  wxIPC_PENDATA = 10,
  wxIPC_RIFF = 11,
  wxIPC_WAVE = 12,
  wxIPC_UTF16TEXT = 13,
  wxIPC_ENHMETAFILE = 14,
  wxIPC_FILENAME = 15,
  wxIPC_LOCALE = 16,
  wxIPC_UTF8TEXT = 17,
  wxIPC_UTF32TEXT = 18,
  wxIPC_UNICODETEXT = wxIPC_UTF16TEXT,
  wxIPC_PRIVATE = 20,
  wxIPC_INVALID = 0,
  wxIPC_TEXT = 1,
  wxIPC_BITMAP = 2,
  wxIPC_METAFILE = 3,
  wxIPC_SYLK = 4,
  wxIPC_DIF = 5,
  wxIPC_TIFF = 6,
  wxIPC_OEMTEXT = 7,
  wxIPC_DIB = 8,
  wxIPC_PALETTE = 9,
  wxIPC_PENDATA = 10,
  wxIPC_RIFF = 11,
  wxIPC_WAVE = 12,
  wxIPC_UTF16TEXT = 13,
  wxIPC_ENHMETAFILE = 14,
  wxIPC_FILENAME = 15,
  wxIPC_LOCALE = 16,
  wxIPC_UTF8TEXT = 17,
  wxIPC_UTF32TEXT = 18,
  wxIPC_UNICODETEXT,
  wxIPC_PRIVATE = 20 }

  See wxTCPConnection.

22.422.1 Enumeration Type Documentation

enum wxIPCFormat

See wxTCPConnection.

Enumerator

wxIPC_INVALID

wxIPC_TEXT CF_TEXT.

wxIPC_BITMAP CF_BITMAP.
wxIPC_METAFILE CF_METAFILEPICT.
wxIPC_SYLK
wxIPC_DIF
wxIPC_TIFF
wxIPC_OEMTEXT CF_OEMTEXT.
wxIPC_DIB CF_DIB.
wxIPC_PALETTE
wxIPC_PENDATA
wxIPC_RIFF
wxIPC_WAVE
wxIPC_UTF16TEXT CF_UNICODE.
wxIPC_ENHMETAFILE
wxIPC_FILENAME CF_HDROP.
wxIPC_LOCALE
wxIPC_UTF8TEXT
wxIPC_UTF32TEXT
wxIPC_UNICODETEXT
wxIPC_PRIVATE
wxIPC_INVALID
wxIPC_TEXT
wxIPC_BITMAP
wxIPC_METAFILE
wxIPC_SYLK
wxIPC_DIF
wxIPC_TIFF
wxIPC_OEMTEXT
wxIPC_DIB
wxIPC_PALETTE
wxIPC_PENDATA
wxIPC_RIFF
wxIPC_WAVE
wxIPC_UTF16TEXT
wxIPC_ENHMETAFILE
wxIPC_FILENAME
wxIPC_LOCALE
wxIPC_UTF8TEXT
wxIPC_UTF32TEXT
wxIPC_UNICODETEXT
wxIPC_PRIVATE

22.423 interface/wx/sckstrm.h File Reference

Classes

- class wxSocketOutputStream

  This class implements an output stream which writes data from a connected socket.
- class wxSocketInputStream

  This class implements an input stream which reads data from a connected socket.

22.424 interface/wx/scopedarray.h File Reference

Classes

- class wxScopedArray< T >

  A scoped array template class.
- class wxScopedArray< T >

  A scoped array template class.

22.425 interface/wx/scopedptr.h File Reference

Classes

- class wxScopedPtr

  This is a simple scoped smart pointer implementation that is similar to the Boost smart pointers (see `http://www.boost.org`*) but rewritten to use macros instead.*
- class wxScopedTiedPtr

  This is a variation on the topic of wxScopedPtr.
- class wxScopedPtr< T >

  A scoped pointer template class.

22.426 interface/wx/scopeguard.h File Reference

Classes

- class wxScopeGuard

  Scope guard is an object which allows executing an action on scope exit.

Macros

- #define wxON_BLOCK_EXIT(function,...)

  Ensure that the global function with a few (up to some implementation-defined limit) is executed on scope exit, whether due to a normal function return or because an exception has been thrown.
- #define wxON_BLOCK_EXIT0(function)
- #define wxON_BLOCK_EXIT1(function, p1)
- #define wxON_BLOCK_EXIT2(function, p1, p2)
- #define wxON_BLOCK_EXIT3(function, p1, p2, p3)
- #define wxON_BLOCK_EXIT_OBJ(object, method,...)

  This family of macros is similar to wxON_BLOCK_EXIT(), but calls a method of the given object instead of a free function.
- #define wxON_BLOCK_EXIT_OBJ0(object, method)
- #define wxON_BLOCK_EXIT_OBJ1(object, method, p1)
- #define wxON_BLOCK_EXIT_OBJ2(object, method, p1, p2)
- #define wxON_BLOCK_EXIT_OBJ3(object, method, p1, p2, p3)
- #define wxON_BLOCK_EXIT_THIS(method,...)

  This family of macros is similar to wxON_BLOCK_OBJ(), but calls a method of `this` *object instead of a method of the specified object.*
- #define wxON_BLOCK_EXIT_THIS0(method)
- #define wxON_BLOCK_EXIT_THIS1(method, p1)
- #define wxON_BLOCK_EXIT_THIS2(method, p1, p2)

- #define wxON_BLOCK_EXIT_THIS3(method, p1, p2, p3)
- #define wxON_BLOCK_EXIT_SET(var, value)

  This macro sets a variable to the specified value on scope exit.
- #define wxON_BLOCK_EXIT_NULL(ptr)

  This macro sets the pointer passed to it as argument to NULL on scope exit.

Functions

- template<typename F , typename P1 , ... , typename PN >
  wxScopeGuard wxMakeGuard (F func, P1 p1,..., PN pN)

  Returns a scope guard object which will call the specified function with the given parameters on scope exit.

22.427 interface/wx/scrolbar.h File Reference

Classes

- class wxScrollBar

  A wxScrollBar is a control that represents a horizontal or vertical scrollbar.

22.428 interface/wx/scrolwin.h File Reference

Classes

- class wxScrolled< T >

  The wxScrolled class manages scrolling for its client area, transforming the coordinates according to the scrollbar positions, and setting the scroll positions, thumb sizes and ranges according to the area in view.

Typedefs

- typedef wxScrolled< wxPanel > wxScrolledWindow

  Scrolled window derived from wxPanel.
- typedef wxScrolled< wxWindow > wxScrolledCanvas

  Alias for wxScrolled<wxWindow>.

Enumerations

- enum wxScrollbarVisibility {
  wxSHOW_SB_NEVER = -1,
  wxSHOW_SB_DEFAULT,
  wxSHOW_SB_ALWAYS }

  Possible values for the second argument of wxScrolled::ShowScrollbars().

22.428.1 Enumeration Type Documentation

enum wxScrollbarVisibility

Possible values for the second argument of wxScrolled::ShowScrollbars().

Enumerator

wxSHOW_SB_NEVER Never show the scrollbar at all.

wxSHOW_SB_DEFAULT Show scrollbar only if it is needed.

wxSHOW_SB_ALWAYS Always show scrollbar, even if not needed.

22.429 interface/wx/settings.h File Reference

Classes

- class wxSystemSettings

  wxSystemSettings allows the application to ask for details about the system.

Enumerations

- enum wxSystemFont {
  wxSYS_OEM_FIXED_FONT = 10,
  wxSYS_ANSI_FIXED_FONT,
  wxSYS_ANSI_VAR_FONT,
  wxSYS_SYSTEM_FONT,
  wxSYS_DEVICE_DEFAULT_FONT,
  wxSYS_DEFAULT_GUI_FONT }

  Possible values for wxSystemSettings::GetFont() parameter.

- enum wxSystemColour {
  wxSYS_COLOUR_SCROLLBAR,
  wxSYS_COLOUR_DESKTOP,
  wxSYS_COLOUR_ACTIVECAPTION,
  wxSYS_COLOUR_INACTIVECAPTION,
  wxSYS_COLOUR_MENU,
  wxSYS_COLOUR_WINDOW,
  wxSYS_COLOUR_WINDOWFRAME,
  wxSYS_COLOUR_MENUTEXT,
  wxSYS_COLOUR_WINDOWTEXT,
  wxSYS_COLOUR_CAPTIONTEXT,
  wxSYS_COLOUR_ACTIVEBORDER,
  wxSYS_COLOUR_INACTIVEBORDER,
  wxSYS_COLOUR_APPWORKSPACE,
  wxSYS_COLOUR_HIGHLIGHT,
  wxSYS_COLOUR_HIGHLIGHTTEXT,
  wxSYS_COLOUR_BTNFACE,
  wxSYS_COLOUR_BTNSHADOW,
  wxSYS_COLOUR_GRAYTEXT,
  wxSYS_COLOUR_BTNTEXT,
  wxSYS_COLOUR_INACTIVECAPTIONTEXT,
  wxSYS_COLOUR_BTNHIGHLIGHT,
  wxSYS_COLOUR_3DDKSHADOW,
  wxSYS_COLOUR_3DLIGHT,
  wxSYS_COLOUR_INFOTEXT,
  wxSYS_COLOUR_INFOBK,
  wxSYS_COLOUR_LISTBOX,
  wxSYS_COLOUR_HOTLIGHT,
  wxSYS_COLOUR_GRADIENTACTIVECAPTION,
  wxSYS_COLOUR_GRADIENTINACTIVECAPTION,
  wxSYS_COLOUR_MENUHILIGHT,
  wxSYS_COLOUR_MENUBAR,
  wxSYS_COLOUR_LISTBOXTEXT,
  wxSYS_COLOUR_LISTBOXHIGHLIGHTTEXT,
  wxSYS_COLOUR_BACKGROUND = wxSYS_COLOUR_DESKTOP,
  wxSYS_COLOUR_3DFACE = wxSYS_COLOUR_BTNFACE,
  wxSYS_COLOUR_3DSHADOW = wxSYS_COLOUR_BTNSHADOW,
  wxSYS_COLOUR_BTNHILIGHT = wxSYS_COLOUR_BTNHIGHLIGHT,
  wxSYS_COLOUR_3DHIGHLIGHT = wxSYS_COLOUR_BTNHIGHLIGHT,
  wxSYS_COLOUR_3DHILIGHT = wxSYS_COLOUR_BTNHIGHLIGHT,
  wxSYS_COLOUR_FRAMEBK = wxSYS_COLOUR_BTNFACE }

Possible values for wxSystemSettings::GetColour() parameter.

- enum wxSystemMetric {
  wxSYS_MOUSE_BUTTONS,
  wxSYS_BORDER_X,
  wxSYS_BORDER_Y,
  wxSYS_CURSOR_X,
  wxSYS_CURSOR_Y,
  wxSYS_DCLICK_X,
  wxSYS_DCLICK_Y,
  wxSYS_DRAG_X,
  wxSYS_DRAG_Y,
  wxSYS_EDGE_X,
  wxSYS_EDGE_Y,
  wxSYS_HSCROLL_ARROW_X,
  wxSYS_HSCROLL_ARROW_Y,
  wxSYS_HTHUMB_X,
  wxSYS_ICON_X,
  wxSYS_ICON_Y,
  wxSYS_ICONSPACING_X,
  wxSYS_ICONSPACING_Y,
  wxSYS_WINDOWMIN_X,
  wxSYS_WINDOWMIN_Y,
  wxSYS_SCREEN_X,
  wxSYS_SCREEN_Y,
  wxSYS_FRAMESIZE_X,
  wxSYS_FRAMESIZE_Y,
  wxSYS_SMALLICON_X,
  wxSYS_SMALLICON_Y,
  wxSYS_HSCROLL_Y,
  wxSYS_VSCROLL_X,
  wxSYS_VSCROLL_ARROW_X,
  wxSYS_VSCROLL_ARROW_Y,
  wxSYS_VTHUMB_Y,
  wxSYS_CAPTION_Y,
  wxSYS_MENU_Y,
  wxSYS_NETWORK_PRESENT,
  wxSYS_PENWINDOWS_PRESENT,
  wxSYS_SHOW_SOUNDS,
  wxSYS_SWAP_BUTTONS,
  wxSYS_DCLICK_MSEC }

  Possible values for wxSystemSettings::GetMetric() index parameter.

- enum wxSystemFeature {
  wxSYS_CAN_DRAW_FRAME_DECORATIONS = 1,
  wxSYS_CAN_ICONIZE_FRAME,
  wxSYS_TABLET_PRESENT }

  Possible values for wxSystemSettings::HasFeature() parameter.

- enum wxSystemScreenType {
  wxSYS_SCREEN_NONE = 0,
  wxSYS_SCREEN_TINY,
  wxSYS_SCREEN_PDA,
  wxSYS_SCREEN_SMALL,
  wxSYS_SCREEN_DESKTOP }

  Values for different screen designs.

22.429.1 Enumeration Type Documentation

enum wxSystemColour

Possible values for wxSystemSettings::GetColour() parameter.

These values map 1:1 the native values supported by the Windows' `GetSysColor` function. Note that other ports (other than wxMSW) will try to provide meaningful colours but they usually map the same colour to various `wxSYS_COLOUR_*` values.

Enumerator

wxSYS_COLOUR_SCROLLBAR The scrollbar grey area.

wxSYS_COLOUR_DESKTOP The desktop colour.

wxSYS_COLOUR_ACTIVECAPTION Active window caption colour.

wxSYS_COLOUR_INACTIVECAPTION Inactive window caption colour.

wxSYS_COLOUR_MENU Menu background colour.

wxSYS_COLOUR_WINDOW Window background colour.

wxSYS_COLOUR_WINDOWFRAME Window frame colour.

wxSYS_COLOUR_MENUTEXT Colour of the text used in the menus.

wxSYS_COLOUR_WINDOWTEXT Colour of the text used in generic windows.

wxSYS_COLOUR_CAPTIONTEXT Colour of the text used in captions, size boxes and scrollbar arrow boxes.

wxSYS_COLOUR_ACTIVEBORDER Active window border colour.

wxSYS_COLOUR_INACTIVEBORDER Inactive window border colour.

wxSYS_COLOUR_APPWORKSPACE Background colour for MDI applications.

wxSYS_COLOUR_HIGHLIGHT Colour of item(s) selected in a control.

wxSYS_COLOUR_HIGHLIGHTTEXT Colour of the text of item(s) selected in a control.

wxSYS_COLOUR_BTNFACE Face shading colour on push buttons.

wxSYS_COLOUR_BTNSHADOW Edge shading colour on push buttons.

wxSYS_COLOUR_GRAYTEXT Colour of greyed (disabled) text.

wxSYS_COLOUR_BTNTEXT Colour of the text on push buttons.

wxSYS_COLOUR_INACTIVECAPTIONTEXT Colour of the text in active captions.

wxSYS_COLOUR_BTNHIGHLIGHT Highlight colour for buttons.

wxSYS_COLOUR_3DDKSHADOW Dark shadow colour for three-dimensional display elements.

wxSYS_COLOUR_3DLIGHT Light colour for three-dimensional display elements.

wxSYS_COLOUR_INFOTEXT Text colour for tooltip controls.

wxSYS_COLOUR_INFOBK Background colour for tooltip controls.

wxSYS_COLOUR_LISTBOX Background colour for list-like controls.

wxSYS_COLOUR_HOTLIGHT Colour for a hyperlink or hot-tracked item.

wxSYS_COLOUR_GRADIENTACTIVECAPTION Right side colour in the color gradient of an active window's title bar. `wxSYS_COLOUR_ACTIVECAPTION` specifies the left side color.

wxSYS_COLOUR_GRADIENTINACTIVECAPTION Right side colour in the color gradient of an inactive window's title bar. `wxSYS_COLOUR_INACTIVECAPTION` specifies the left side color.

wxSYS_COLOUR_MENUHILIGHT The colour used to highlight menu items when the menu appears as a flat menu. The highlighted menu item is outlined with `wxSYS_COLOUR_HIGHLIGHT`.

wxSYS_COLOUR_MENUBAR The background colour for the menu bar when menus appear as flat menus. However, `wxSYS_COLOUR_MENU` continues to specify the background color of the menu popup.

wxSYS_COLOUR_LISTBOXTEXT Text colour for list-like controls.

Since

2.9.0

wxSYS_COLOUR_LISTBOXHIGHLIGHTTEXT Text colour for the unfocused selection of list-like controls.

Since

2.9.1

wxSYS_COLOUR_BACKGROUND Synonym for `wxSYS_COLOUR_DESKTOP`.

wxSYS_COLOUR_3DFACE Synonym for `wxSYS_COLOUR_BTNFACE`.

wxSYS_COLOUR_3DSHADOW Synonym for `wxSYS_COLOUR_BTNSHADOW`.

wxSYS_COLOUR_BTNHILIGHT Synonym for `wxSYS_COLOUR_BTNHIGHLIGHT`.

wxSYS_COLOUR_3DHIGHLIGHT Synonym for `wxSYS_COLOUR_BTNHIGHLIGHT`.

wxSYS_COLOUR_3DHILIGHT Synonym for `wxSYS_COLOUR_BTNHIGHLIGHT`.

wxSYS_COLOUR_FRAMEBK Synonym for `wxSYS_COLOUR_BTNFACE`. On wxMSW this colour should be used as the background colour of wxFrames which are used as containers of controls; this is in fact the same colour used for the borders of controls like e.g. wxNotebook or for the background of e.g. wxPanel.

Since

2.9.0

enum wxSystemFeature

Possible values for wxSystemSettings::HasFeature() parameter.

Enumerator

wxSYS_CAN_DRAW_FRAME_DECORATIONS

wxSYS_CAN_ICONIZE_FRAME

wxSYS_TABLET_PRESENT

enum wxSystemFont

Possible values for wxSystemSettings::GetFont() parameter.

These values map 1:1 the native values supported by the Windows' `GetStockObject` function. Note that other ports (other than wxMSW) will try to provide meaningful fonts but they usually map the same font to various `wxSYS_*_FONT` values.

Enumerator

wxSYS_OEM_FIXED_FONT Original equipment manufacturer dependent fixed-pitch font.

wxSYS_ANSI_FIXED_FONT Windows fixed-pitch (monospaced) font.

wxSYS_ANSI_VAR_FONT Windows variable-pitch (proportional) font.

wxSYS_SYSTEM_FONT System font. By default, the system uses the system font to draw menus, dialog box controls, and text.

wxSYS_DEVICE_DEFAULT_FONT Device-dependent font (Windows NT and later only).

wxSYS_DEFAULT_GUI_FONT Default font for user interface objects such as menus and dialog boxes. Note that with modern GUIs nothing guarantees that the same font is used for all GUI elements, so some controls might use a different font by default.

enum wxSystemMetric

Possible values for wxSystemSettings::GetMetric() index parameter.

Enumerator

wxSYS_MOUSE_BUTTONS Number of buttons on mouse, or zero if no mouse was installed.

wxSYS_BORDER_X Width of single border.

wxSYS_BORDER_Y Height of single border.

wxSYS_CURSOR_X Width of cursor.

wxSYS_CURSOR_Y Height of cursor.

wxSYS_DCLICK_X Width in pixels of rectangle within which two successive mouse clicks must fall to generate a double-click.

wxSYS_DCLICK_Y Height in pixels of rectangle within which two successive mouse clicks must fall to generate a double-click.

wxSYS_DRAG_X Width in pixels of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.

wxSYS_DRAG_Y Height in pixels of a rectangle centered on a drag point to allow for limited movement of the mouse pointer before a drag operation begins.

wxSYS_EDGE_X Width of a 3D border, in pixels.

wxSYS_EDGE_Y Height of a 3D border, in pixels.

wxSYS_HSCROLL_ARROW_X Width of arrow bitmap on horizontal scrollbar.

wxSYS_HSCROLL_ARROW_Y Height of arrow bitmap on horizontal scrollbar.

wxSYS_HTHUMB_X Width of horizontal scrollbar thumb.

wxSYS_ICON_X The default width of an icon.

wxSYS_ICON_Y The default height of an icon.

wxSYS_ICONSPACING_X Width of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged.

wxSYS_ICONSPACING_Y Height of a grid cell for items in large icon view, in pixels. Each item fits into a rectangle of this size when arranged.

wxSYS_WINDOWMIN_X Minimum width of a window.

wxSYS_WINDOWMIN_Y Minimum height of a window.

wxSYS_SCREEN_X Width of the screen in pixels.

wxSYS_SCREEN_Y Height of the screen in pixels.

wxSYS_FRAMESIZE_X Width of the window frame for a wxTHICK_FRAME window.

wxSYS_FRAMESIZE_Y Height of the window frame for a wxTHICK_FRAME window.

wxSYS_SMALLICON_X Recommended width of a small icon (in window captions, and small icon view).

wxSYS_SMALLICON_Y Recommended height of a small icon (in window captions, and small icon view).

wxSYS_HSCROLL_Y Height of horizontal scrollbar in pixels.

wxSYS_VSCROLL_X Width of vertical scrollbar in pixels.

wxSYS_VSCROLL_ARROW_X Width of arrow bitmap on a vertical scrollbar.

wxSYS_VSCROLL_ARROW_Y Height of arrow bitmap on a vertical scrollbar.

wxSYS_VTHUMB_Y Height of vertical scrollbar thumb.

wxSYS_CAPTION_Y Height of normal caption area.

wxSYS_MENU_Y Height of single-line menu bar.

wxSYS_NETWORK_PRESENT 1 if there is a network present, 0 otherwise.

wxSYS_PENWINDOWS_PRESENT 1 if PenWindows is installed, 0 otherwise.

wxSYS_SHOW_SOUNDS Non-zero if the user requires an application to present information visually in situations where it would otherwise present the information only in audible form; zero otherwise.

wxSYS_SWAP_BUTTONS Non-zero if the meanings of the left and right mouse buttons are swapped; zero otherwise.

wxSYS_DCLICK_MSEC Maximal time, in milliseconds, which may pass between subsequent clicks for a double click to be generated.

enum wxSystemScreenType

Values for different screen designs.

See wxSystemSettings::GetScreenType().

Enumerator

wxSYS_SCREEN_NONE Undefined screen type.

wxSYS_SCREEN_TINY Tiny screen, less than 320x240.

wxSYS_SCREEN_PDA PDA screen, 320x240 or more but less than 640x480.

wxSYS_SCREEN_SMALL Small screen, 640x480 or more but less than 800x600.

wxSYS_SCREEN_DESKTOP Desktop screen, 800x600 or more.

22.430 interface/wx/sharedptr.h File Reference

Classes

- class wxSharedPtr< T >

  A smart pointer with non-intrusive reference counting.

22.431 interface/wx/simplebook.h File Reference

Classes

- class wxSimplebook

  wxSimplebook is a control showing exactly one of its several pages.

22.432 interface/wx/slider.h File Reference

Classes

- class wxSlider

  A slider is a control with a handle which can be pulled back and forth to change the value.

Macros

- #define wxSL_HORIZONTAL wxHORIZONTAL /* 0x0004 */
- #define wxSL_VERTICAL wxVERTICAL /* 0x0008 */
- #define wxSL_TICKS 0x0010
- #define wxSL_AUTOTICKS wxSL_TICKS
- #define wxSL_LEFT 0x0040

- #define wxSL_TOP 0x0080
- #define wxSL_RIGHT 0x0100
- #define wxSL_BOTTOM 0x0200
- #define wxSL_BOTH 0x0400
- #define wxSL_SELRANGE 0x0800
- #define wxSL_INVERSE 0x1000
- #define wxSL_MIN_MAX_LABELS 0x2000
- #define wxSL_VALUE_LABEL 0x4000
- #define wxSL_LABELS (wxSL_MIN_MAX_LABELS|wxSL_VALUE_LABEL)

22.432.1 Macro Definition Documentation

#define wxSL_AUTOTICKS wxSL_TICKS

#define wxSL_BOTH 0x0400

#define wxSL_BOTTOM 0x0200

#define wxSL_HORIZONTAL wxHORIZONTAL /∗ 0x0004 ∗/

#define wxSL_INVERSE 0x1000

#define wxSL_LABELS (wxSL_MIN_MAX_LABELS|wxSL_VALUE_LABEL)

#define wxSL_LEFT 0x0040

#define wxSL_MIN_MAX_LABELS 0x2000

#define wxSL_RIGHT 0x0100

#define wxSL_SELRANGE 0x0800

#define wxSL_TICKS 0x0010

#define wxSL_TOP 0x0080

#define wxSL_VALUE_LABEL 0x4000

#define wxSL_VERTICAL wxVERTICAL /∗ 0x0008 ∗/

22.433 interface/wx/snglinst.h File Reference

Classes

- class wxSingleInstanceChecker

  wxSingleInstanceChecker class allows to check that only a single instance of a program is running.

22.434 interface/wx/socket.h File Reference

Classes

- class wxIPaddress

  wxIPaddress is an abstract base class for all internet protocol address objects.

- class wxIPV4address

  A class for working with IPv4 network addresses.
- class wxSocketServer
- class wxSocketClient
- class wxSockAddress

  You are unlikely to need to use this class: only wxSocketBase uses it.
- class wxSocketEvent

  This event class contains information about socket events.
- class wxSocketBase

  wxSocketBase is the base class for all socket-related objects, and it defines all basic IO functionality.
- class wxDatagramSocket

Typedefs

- typedef int wxSOCKET_T

  The type of the native socket.

Enumerations

- enum wxSocketError {
  wxSOCKET_NOERROR,
  wxSOCKET_INVOP,
  wxSOCKET_IOERR,
  wxSOCKET_INVADDR,
  wxSOCKET_INVSOCK,
  wxSOCKET_NOHOST,
  wxSOCKET_INVPORT,
  wxSOCKET_WOULDBLOCK,
  wxSOCKET_TIMEDOUT,
  wxSOCKET_MEMERR }

  wxSocket error return values.
- enum wxSocketEventFlags {
  wxSOCKET_INPUT,
  wxSOCKET_OUTPUT,
  wxSOCKET_CONNECTION,
  wxSOCKET_LOST }
- enum {
  wxSOCKET_NONE = 0,
  wxSOCKET_NOWAIT = 1,
  wxSOCKET_WAITALL = 2,
  wxSOCKET_BLOCK = 4,
  wxSOCKET_REUSEADDR = 8,
  wxSOCKET_BROADCAST = 16,
  wxSOCKET_NOBIND = 32,
  wxSOCKET_NOWAIT_READ = 64,
  wxSOCKET_WAITALL_READ = 128,
  wxSOCKET_NOWAIT_WRITE = 256,
  wxSOCKET_WAITALL_WRITE = 512 }

22.434.1 Typedef Documentation

typedef int wxSOCKET_T

The type of the native socket.

Notice that the definition below is simplified and this type is not always int, e.g. it is a 64 bit integer type under Win64.

Since

2.9.5

22.434.2 Enumeration Type Documentation

anonymous enum

wxSocket Flags.

A brief overview on how to use these flags follows.

If no flag is specified (this is the same as **wxSOCKET_NONE**), IO calls will return after some data has been read or written, even when the transfer might not be complete. This is the same as issuing exactly one blocking low-level call to **recv()** or **send()**. Note that *blocking* here refers to when the function returns, not to whether the GUI blocks during this time.

If **wxSOCKET_NOWAIT** is specified, IO calls will return immediately. Read operations will retrieve only available data. Write operations will write as much data as possible, depending on how much space is available in the output buffer. This is the same as issuing exactly one nonblocking low-level call to **recv()** or **send()**. Note that *nonblocking* here refers to when the function returns, not to whether the GUI blocks during this time. Also note that this flag impacts both Read and Write operations. If it is desired to control Read independently of Write, for example you want no wait on Read(), but you do want to wait on Write(), then use wxSOCKET_NOWAIT_READ and wxSOCKET_NOWAIT_WRITE.

If **wxSOCKET_NOWAIT_READ** (this flag is new since wxWidgets 2.9.5) is specified, Read operations will return immediately. Read operations will retrieve only available data. This is the same as issuing exactly one nonblocking low-level call to **recv()**. Note that *nonblocking* here refers to when the function returns, not to whether the GUI blocks during this time. This flag should not be enabled if ReadMsg() is going to be used (it will be ignored), if you do then thread-safety may be at risk. Note that wxSOCKET_NOWAIT_READ impacts only Read operations and does not impact Write operations, allowing Read and Write operations to be set differently.

If **wxSOCKET_NOWAIT_WRITE** (this flag is new since wxWidgets 2.9.5) is specified, Write operations will return immediately. Write operations will write as much data as possible, depending on how much space is available in the output buffer. This is the same as issuing exactly one nonblocking low-level call to **send()**. Note that *nonblocking* here refers to when the function returns, not to whether the GUI blocks during this time. This flag should not be enabled if WriteMsg() is going to be used (it will be ignored), if you use it then thread safety may be at risk. Note that wxSOCKET_NOWAIT_WRITE impacts only Write operations and does not impact Read operations, allowing Read and Write operations to be set differently.

If **wxSOCKET_WAITALL** is specified, IO calls won't return until ALL the data has been read or written (or until an error occurs), blocking if necessary, and issuing several low level calls if necessary. This is the same as having a loop which makes as many blocking low-level calls to **recv()** or **send()** as needed so as to transfer all the data. Note that *blocking* here refers to when the function returns, not to whether the GUI blocks during this time. Note that wxSOCKET_WAITALL impacts both Read and Write operations. If you desire to wait for all on just Read operations, but not on Write operations, (or vice versa), use wxSOCKET_WAITALL_READ or wxSOCKET_WAITALL_WRITE.

If **wxSOCKET_WAITALL_READ** (this flag is new since wxWidgets 2.9.5) is specified, Read operations won't return until ALL the data has been read (or until an error occurs), blocking if necessary, and issuing several low level calls if necessary. This is the same as having a loop which makes as many blocking low-level calls to **recv()** as needed so as to transfer all the data. Note that *blocking* here refers to when the function returns, not to whether the GUI blocks during this time. Note that wxSOCKET_WAITALL_READ only has an impact on Read operations, and has no impact on Write operations, allowing Read and Write operations to have different settings.

If **wxSOCKET_WAITALL_WRITE** (this flag is new since wxWidgets 2.9.5) is specified, Write() and WriteMsg() calls won't return until ALL the data has been written (or until an error occurs), blocking if necessary, and issuing several low level calls if necessary. This is the same as having a loop which makes as many blocking low-level calls to **send()** as needed so as to transfer all the data. Note that *blocking* here refers to when the function returns, not to whether the GUI blocks during this time. Note that wxSOCKET_WAITALL_WRITE only has an impact on Write

operations, and has no impact on Read operations, allowing Read and Write operations to have different settings.

The **wxSOCKET_BLOCK** flag controls whether the GUI blocks during IO operations. If this flag is specified, the socket will not yield during IO calls, so the GUI will remain blocked until the operation completes. If it is not used, then the application must take extra care to avoid unwanted reentrance.

The **wxSOCKET_REUSEADDR** flag controls the use of the **SO_REUSEADDR** standard **setsockopt()** flag. This flag allows the socket to bind to a port that is already in use. This is mostly used on UNIX-based systems to allow rapid starting and stopping of a server, otherwise you may have to wait several minutes for the port to become available.

wxSOCKET_REUSEADDR can also be used with socket clients to (re)bind to a particular local port for an outgoing connection. This option can have surprising platform dependent behaviour, so check the documentation for your platform's implementation of setsockopt().

Note that on BSD-based systems(e.g. Mac OS X), use of **wxSOCKET_REUSEADDR** implies **SO_REUSEPORT** in addition to **SO_REUSEADDR** to be consistent with Windows.

The **wxSOCKET_BROADCAST** flag controls the use of the **SO_BROADCAST** standard **setsockopt()** flag. This flag allows the socket to use the broadcast address, and is generally used in conjunction with **wxSOCKET_NOBIND** and wxIPaddress::BroadcastAddress().

So:

- **wxSOCKET_NONE** will try to read at least SOME data, no matter how much.
- **wxSOCKET_NOWAIT** will always return immediately, even if it cannot read or write ANY data.
- **wxSOCKET_WAITALL** will only return when it has read or written ALL the data.
- **wxSOCKET_BLOCK** has nothing to do with the previous flags and it controls whether the GUI blocks.
- **wxSOCKET_REUSEADDR** controls special platform-specific behaviour for reusing local addresses/ports.

Enumerator

wxSOCKET_NONE Normal functionality.

wxSOCKET_NOWAIT Read/write as much data as possible and return immediately.

wxSOCKET_WAITALL Wait for all required data to be read/written unless an error occurs.

wxSOCKET_BLOCK Block the GUI (do not yield) while reading/writing data.

wxSOCKET_REUSEADDR Allows the use of an in-use port.

wxSOCKET_BROADCAST Switches the socket to broadcast mode.

wxSOCKET_NOBIND Stops the socket from being bound to a specific adapter (normally used in conjunction with **wxSOCKET_BROADCAST**)

wxSOCKET_NOWAIT_READ Read as much data as possible and return immediately.

wxSOCKET_WAITALL_READ Wait for all required data to be read unless an error occurs.

wxSOCKET_NOWAIT_WRITE Write as much data as possible and return immediately.

wxSOCKET_WAITALL_WRITE Wait for all required data to be written unless an error occurs.

enum wxSocketError

wxSocket error return values.

Enumerator

wxSOCKET_NOERROR No error happened.

wxSOCKET_INVOP Invalid operation.

wxSOCKET_IOERR Input/Output error.

wxSOCKET_INVADDR Invalid address passed to wxSocket.

wxSOCKET_INVSOCK Invalid socket (uninitialized).

wxSOCKET_NOHOST No corresponding host.

wxSOCKET_INVPORT Invalid port.

wxSOCKET_WOULDBLOCK The socket is non-blocking and the operation would block.

wxSOCKET_TIMEDOUT The timeout for this operation expired.

wxSOCKET_MEMERR Memory exhausted.

enum wxSocketEventFlags

wxSocket Event Flags.

A brief note on how to use these events:

The **wxSOCKET_INPUT** event will be issued whenever there is data available for reading. This will be the case if the input queue was empty and new data arrives, or if the application has read some data yet there is still more data available. This means that the application does not need to read all available data in response to a **wxSOCKET_INPUT** event, as more events will be produced as necessary.

The **wxSOCKET_OUTPUT** event is issued when a socket is first connected with Connect() or accepted with Accept(). After that, new events will be generated only after an output operation fails with **wxSOCKET_WOULDBLOCK** and buffer space becomes available again. This means that the application should assume that it can write data to the socket until an **wxSOCKET_WOULDBLOCK** error occurs; after this, whenever the socket becomes writable again the application will be notified with another **wxSOCKET_OUTPUT** event.

The **wxSOCKET_CONNECTION** event is issued when a delayed connection request completes successfully (client) or when a new connection arrives at the incoming queue (server).

The **wxSOCKET_LOST** event is issued when a close indication is received for the socket. This means that the connection broke down or that it was closed by the peer. Also, this event will be issued if a connection request fails.

Enumerator

wxSOCKET_INPUT There is data available for reading.

wxSOCKET_OUTPUT The socket is ready to be written to.

wxSOCKET_CONNECTION Incoming connection request (server), or successful connection establishment (client).

wxSOCKET_LOST The connection has been closed.

22.435 interface/wx/sound.h File Reference

Classes

- class wxSound

  This class represents a short sound (loaded from Windows WAV file), that can be stored in memory and played.

Macros

- #define wxSOUND_SYNC 0
- #define wxSOUND_ASYNC 1
- #define wxSOUND_LOOP 2

22.435.1 Macro Definition Documentation

#define wxSOUND_ASYNC 1

#define wxSOUND_LOOP 2

#define wxSOUND_SYNC 0

22.436 interface/wx/spinbutt.h File Reference

Classes

- class wxSpinEvent

  This event class is used for the events generated by wxSpinButton and wxSpinCtrl.
- class wxSpinButton

  A wxSpinButton has two small up and down (or left and right) arrow buttons.

22.437 interface/wx/spinctrl.h File Reference

Classes

- class wxSpinCtrl

  wxSpinCtrl combines wxTextCtrl and wxSpinButton in one control.
- class wxSpinCtrlDouble

  wxSpinCtrlDouble combines wxTextCtrl and wxSpinButton in one control and displays a real number.
- class wxSpinDoubleEvent

  This event class is used for the events generated by wxSpinCtrlDouble.

Variables

- wxEventType wxEVT_SPINCTRL
- wxEventType wxEVT_SPINCTRLDOUBLE

22.437.1 Variable Documentation

wxEventType wxEVT_SPINCTRL

wxEventType wxEVT_SPINCTRLDOUBLE

22.438 interface/wx/splash.h File Reference

Classes

- class wxSplashScreen

  wxSplashScreen shows a window with a thin border, displaying a bitmap describing your application.

Macros

- #define wxSPLASH_CENTRE_ON_PARENT 0x01
- #define wxSPLASH_CENTRE_ON_SCREEN 0x02
- #define wxSPLASH_NO_CENTRE 0x00
- #define wxSPLASH_TIMEOUT 0x04
- #define wxSPLASH_NO_TIMEOUT 0x00

22.438.1 Macro Definition Documentation

#define wxSPLASH_CENTRE_ON_PARENT 0x01

#define wxSPLASH_CENTRE_ON_SCREEN 0x02

#define wxSPLASH_NO_CENTRE 0x00

#define wxSPLASH_NO_TIMEOUT 0x00

#define wxSPLASH_TIMEOUT 0x04

22.439 interface/wx/splitter.h File Reference

Classes

- class wxSplitterWindow

  This class manages up to two subwindows.
- class wxSplitterEvent

  This class represents the events generated by a splitter control.

Macros

- #define wxSP_NOBORDER 0x0000
- #define wxSP_THIN_SASH 0x0000
- #define wxSP_NOSASH 0x0010
- #define wxSP_PERMIT_UNSPLIT 0x0040
- #define wxSP_LIVE_UPDATE 0x0080
- #define wxSP_3DSASH 0x0100
- #define wxSP_3DBORDER 0x0200
- #define wxSP_NO_XP_THEME 0x0400
- #define wxSP_BORDER wxSP_3DBORDER
- #define wxSP_3D (wxSP_3DBORDER | wxSP_3DSASH)

Enumerations

- enum wxSplitMode {
  wxSPLIT_HORIZONTAL = 1,
  wxSPLIT_VERTICAL }
- enum {
  wxSPLIT_DRAG_NONE,
  wxSPLIT_DRAG_DRAGGING,
  wxSPLIT_DRAG_LEFT_DOWN }

Variables

- wxEventType wxEVT_SPLITTER_SASH_POS_CHANGED
- wxEventType wxEVT_SPLITTER_SASH_POS_CHANGING
- wxEventType wxEVT_SPLITTER_DOUBLECLICKED
- wxEventType wxEVT_SPLITTER_UNSPLIT

22.439.1 Macro Definition Documentation

#define wxSP_3D (wxSP_3DBORDER | wxSP_3DSASH)

#define wxSP_3DBORDER 0x0200

#define wxSP_3DSASH 0x0100

#define wxSP_BORDER wxSP_3DBORDER

#define wxSP_LIVE_UPDATE 0x0080

#define wxSP_NO_XP_THEME 0x0400

#define wxSP_NOBORDER 0x0000

#define wxSP_NOSASH 0x0010

#define wxSP_PERMIT_UNSPLIT 0x0040

#define wxSP_THIN_SASH 0x0000

22.439.2 Enumeration Type Documentation

anonymous enum

Enumerator

wxSPLIT_DRAG_NONE

wxSPLIT_DRAG_DRAGGING

wxSPLIT_DRAG_LEFT_DOWN

enum wxSplitMode

Enumerator

wxSPLIT_HORIZONTAL

wxSPLIT_VERTICAL

22.439.3 Variable Documentation

wxEventType wxEVT_SPLITTER_DOUBLECLICKED

wxEventType wxEVT_SPLITTER_SASH_POS_CHANGED

wxEventType wxEVT_SPLITTER_SASH_POS_CHANGING

wxEventType wxEVT_SPLITTER_UNSPLIT

22.440 interface/wx/srchctrl.h File Reference

Classes

- class wxSearchCtrl

  A search control is a composite control with a search button, a text control, and a cancel button.

Variables

- wxEventType wxEVT_SEARCHCTRL_CANCEL_BTN
- wxEventType wxEVT_SEARCHCTRL_SEARCH_BTN

22.440.1 Variable Documentation

wxEventType wxEVT_SEARCHCTRL_CANCEL_BTN

wxEventType wxEVT_SEARCHCTRL_SEARCH_BTN

22.441 interface/wx/sstream.h File Reference

Classes

- class wxStringInputStream

  This class implements an input stream which reads data from a string.
- class wxStringOutputStream

  This class implements an output stream which writes data either to a user-provided or internally allocated string.

22.442 interface/wx/stack.h File Reference

Classes

- class wxStack< T >

  wxStack<T> is similar to `std::stack` *and can be used exactly like it.*

22.443 interface/wx/stackwalk.h File Reference

Classes

- class wxStackWalker

  wxStackWalker allows an application to enumerate, or walk, the stack frames (the function callstack).
- class wxStackFrame

  wxStackFrame represents a single stack frame, or a single function in the call stack, and is used exclusively together with wxStackWalker, see there for a more detailed discussion.

Macros

- #define wxSTACKWALKER_MAX_DEPTH (200)

  This is the default value of the wxStackWalker::Walk function.

22.443.1 Macro Definition Documentation

#define wxSTACKWALKER_MAX_DEPTH (200)

This is the default value of the wxStackWalker::Walk function.

22.444 interface/wx/statbmp.h File Reference

Classes

- class wxStaticBitmap

  A static bitmap control displays a bitmap.

22.445 interface/wx/statbox.h File Reference

Classes

- class wxStaticBox

  A static box is a rectangle drawn around other windows to denote a logical grouping of items.

22.446 interface/wx/statline.h File Reference

Classes

- class wxStaticLine

  A static line is just a line which may be used in a dialog to separate the groups of controls.

22.447 interface/wx/stattext.h File Reference

Classes

- class wxStaticText

  A static text control displays one or more lines of read-only text.

Macros

- #define wxST_NO_AUTORESIZE 0x0001
- #define wxST_ELLIPSIZE_START 0x0004
- #define wxST_ELLIPSIZE_MIDDLE 0x0008
- #define wxST_ELLIPSIZE_END 0x0010

22.447.1 Macro Definition Documentation

#define wxST_ELLIPSIZE_END 0x0010

#define wxST_ELLIPSIZE_MIDDLE 0x0008

#define wxST_ELLIPSIZE_START 0x0004

#define wxST_NO_AUTORESIZE 0x0001

22.448 interface/wx/statusbr.h File Reference

Classes

- class wxStatusBarPane

  A status bar pane data container used by wxStatusBar.
- class wxStatusBar

  A status bar is a narrow window that can be placed along the bottom of a frame to give small amounts of status information.

Macros

- #define wxSTB_SIZEGRIP 0x0010
- #define wxSTB_SHOW_TIPS 0x0020
- #define wxSTB_ELLIPSIZE_START 0x0040
- #define wxSTB_ELLIPSIZE_MIDDLE 0x0080
- #define wxSTB_ELLIPSIZE_END 0x0100
- #define wxSTB_DEFAULT_STYLE (wxSTB_SIZEGRIP|wxSTB_ELLIPSIZE_END|wxSTB_SHOW_TIP↩S|wxFULL_REPAINT_ON_RESIZE)
- #define wxSB_NORMAL 0x0000
- #define wxSB_FLAT 0x0001
- #define wxSB_RAISED 0x0002
- #define wxSB_SUNKEN 0x0003

22.448.1 Macro Definition Documentation

#define wxSB_FLAT 0x0001

#define wxSB_NORMAL 0x0000

#define wxSB_RAISED 0x0002

#define wxSB_SUNKEN 0x0003

#define wxSTB_DEFAULT_STYLE (wxSTB_SIZEGRIP|wxSTB_ELLIPSIZE_END|wxSTB_SHOW_TIPS|wxFULL_↩REPAINT_ON_RESIZE)

#define wxSTB_ELLIPSIZE_END 0x0100

#define wxSTB_ELLIPSIZE_MIDDLE 0x0080

#define wxSTB_ELLIPSIZE_START 0x0040

#define wxSTB_SHOW_TIPS 0x0020

#define wxSTB_SIZEGRIP 0x0010

22.449 interface/wx/stc/stc.h File Reference

Classes

- class wxStyledTextCtrl

  A wxWidgets implementation of the Scintilla source code editing component.
- class wxStyledTextEvent

  The type of events sent from wxStyledTextCtrl.

Macros

- #define wxSTC_INVALID_POSITION -1
- #define wxSTC_START 2000

  Define start of Scintilla messages to be greater than all Windows edit (EM_∗) messages as many EM_ messages can be used although that use is deprecated.
- #define wxSTC_OPTIONAL_START 3000
- #define wxSTC_LEXER_START 4000
- #define wxSTC_WS_INVISIBLE 0
- #define wxSTC_WS_VISIBLEALWAYS 1
- #define wxSTC_WS_VISIBLEAFTERINDENT 2
- #define wxSTC_EOL_CRLF 0
- #define wxSTC_EOL_CR 1
- #define wxSTC_EOL_LF 2
- #define wxSTC_CP_UTF8 65001

  The SC_CP_UTF8 value can be used to enter Unicode mode.
- #define wxSTC_MARKER_MAX 31
- #define wxSTC_MARK_CIRCLE 0
- #define wxSTC_MARK_ROUNDRECT 1
- #define wxSTC_MARK_ARROW 2
- #define wxSTC_MARK_SMALLRECT 3
- #define wxSTC_MARK_SHORTARROW 4
- #define wxSTC_MARK_EMPTY 5
- #define wxSTC_MARK_ARROWDOWN 6
- #define wxSTC_MARK_MINUS 7
- #define wxSTC_MARK_PLUS 8
- #define wxSTC_MARK_VLINE 9

  Shapes used for outlining column.
- #define wxSTC_MARK_LCORNER 10
- #define wxSTC_MARK_TCORNER 11
- #define wxSTC_MARK_BOXPLUS 12
- #define wxSTC_MARK_BOXPLUSCONNECTED 13
- #define wxSTC_MARK_BOXMINUS 14
- #define wxSTC_MARK_BOXMINUSCONNECTED 15
- #define wxSTC_MARK_LCORNERCURVE 16
- #define wxSTC_MARK_TCORNERCURVE 17
- #define wxSTC_MARK_CIRCLEPLUS 18
- #define wxSTC_MARK_CIRCLEPLUSCONNECTED 19
- #define wxSTC_MARK_CIRCLEMINUS 20
- #define wxSTC_MARK_CIRCLEMINUSCONNECTED 21
- #define wxSTC_MARK_BACKGROUND 22

  Invisible mark that only sets the line background colour.
- #define wxSTC_MARK_DOTDOTDOT 23
- #define wxSTC_MARK_ARROWS 24
- #define wxSTC_MARK_PIXMAP 25
- #define wxSTC_MARK_FULLRECT 26

- #define wxSTC_MARK_LEFTRECT 27
- #define wxSTC_MARK_AVAILABLE 28
- #define wxSTC_MARK_UNDERLINE 29
- #define wxSTC_MARK_RGBAIMAGE 30
- #define wxSTC_MARK_BOOKMARK 31
- #define wxSTC_MARK_CHARACTER 10000
- #define wxSTC_MARKNUM_FOLDEREND 25

  Markers used for outlining column.
- #define wxSTC_MARKNUM_FOLDEROPENMID 26
- #define wxSTC_MARKNUM_FOLDERMIDTAIL 27
- #define wxSTC_MARKNUM_FOLDERTAIL 28
- #define wxSTC_MARKNUM_FOLDERSUB 29
- #define wxSTC_MARKNUM_FOLDER 30
- #define wxSTC_MARKNUM_FOLDEROPEN 31
- #define wxSTC_MASK_FOLDERS 0xFE000000
- #define wxSTC_MAX_MARGIN 4
- #define wxSTC_MARGIN_SYMBOL 0
- #define wxSTC_MARGIN_NUMBER 1
- #define wxSTC_MARGIN_BACK 2
- #define wxSTC_MARGIN_FORE 3
- #define wxSTC_MARGIN_TEXT 4
- #define wxSTC_MARGIN_RTEXT 5
- #define wxSTC_STYLE_DEFAULT 32

  Styles in range 32..38 are predefined for parts of the UI and are not used as normal styles.
- #define wxSTC_STYLE_LINENUMBER 33
- #define wxSTC_STYLE_BRACELIGHT 34
- #define wxSTC_STYLE_BRACEBAD 35
- #define wxSTC_STYLE_CONTROLCHAR 36
- #define wxSTC_STYLE_INDENTGUIDE 37
- #define wxSTC_STYLE_CALLTIP 38
- #define wxSTC_STYLE_LASTPREDEFINED 39
- #define wxSTC_STYLE_MAX 255
- #define wxSTC_CHARSET_ANSI 0

  Character set identifiers are used in StyleSetCharacterSet.
- #define wxSTC_CHARSET_DEFAULT 1
- #define wxSTC_CHARSET_BALTIC 186
- #define wxSTC_CHARSET_CHINESEBIG5 136
- #define wxSTC_CHARSET_EASTEUROPE 238
- #define wxSTC_CHARSET_GB2312 134
- #define wxSTC_CHARSET_GREEK 161
- #define wxSTC_CHARSET_HANGUL 129
- #define wxSTC_CHARSET_MAC 77
- #define wxSTC_CHARSET_OEM 255
- #define wxSTC_CHARSET_RUSSIAN 204
- #define wxSTC_CHARSET_CYRILLIC 1251
- #define wxSTC_CHARSET_SHIFTJIS 128
- #define wxSTC_CHARSET_SYMBOL 2
- #define wxSTC_CHARSET_TURKISH 162
- #define wxSTC_CHARSET_JOHAB 130
- #define wxSTC_CHARSET_HEBREW 177
- #define wxSTC_CHARSET_ARABIC 178
- #define wxSTC_CHARSET_VIETNAMESE 163
- #define wxSTC_CHARSET_THAI 222
- #define wxSTC_CHARSET_8859_15 1000

- #define wxSTC_CASE_MIXED 0
- #define wxSTC_CASE_UPPER 1
- #define wxSTC_CASE_LOWER 2
- #define wxSTC_FONT_SIZE_MULTIPLIER 100
- #define wxSTC_WEIGHT_NORMAL 400
- #define wxSTC_WEIGHT_SEMIBOLD 600
- #define wxSTC_WEIGHT_BOLD 700
- #define wxSTC_INDIC_PLAIN 0

  Indicator style enumeration and some constants.
- #define wxSTC_INDIC_SQUIGGLE 1
- #define wxSTC_INDIC_TT 2
- #define wxSTC_INDIC_DIAGONAL 3
- #define wxSTC_INDIC_STRIKE 4
- #define wxSTC_INDIC_HIDDEN 5
- #define wxSTC_INDIC_BOX 6
- #define wxSTC_INDIC_ROUNDBOX 7
- #define wxSTC_INDIC_STRAIGHTBOX 8
- #define wxSTC_INDIC_DASH 9
- #define wxSTC_INDIC_DOTS 10
- #define wxSTC_INDIC_SQUIGGLELOW 11
- #define wxSTC_INDIC_DOTBOX 12
- #define wxSTC_INDIC_SQUIGGLEPIXMAP 13
- #define wxSTC_INDIC_COMPOSITIONTHICK 14
- #define wxSTC_INDIC_MAX 31
- #define wxSTC_INDIC_CONTAINER 8
- #define wxSTC_INDIC0_MASK 0x20
- #define wxSTC_INDIC1_MASK 0x40
- #define wxSTC_INDIC2_MASK 0x80
- #define wxSTC_INDICS_MASK 0xE0
- #define wxSTC_IV_NONE 0
- #define wxSTC_IV_REAL 1
- #define wxSTC_IV_LOOKFORWARD 2
- #define wxSTC_IV_LOOKBOTH 3
- #define wxSTC_PRINT_NORMAL 0

  PrintColourMode - use same colours as screen.
- #define wxSTC_PRINT_INVERTLIGHT 1

  PrintColourMode - invert the light value of each style for printing.
- #define wxSTC_PRINT_BLACKONWHITE 2

  PrintColourMode - force black text on white background for printing.
- #define wxSTC_PRINT_COLOURONWHITE 3

  PrintColourMode - text stays coloured, but all background is forced to be white for printing.
- #define wxSTC_PRINT_COLOURONWHITEDEFAULTBG 4

  PrintColourMode - only the default-background is forced to be white for printing.
- #define wxSTC_FIND_WHOLEWORD 0x2
- #define wxSTC_FIND_MATCHCASE 0x4
- #define wxSTC_FIND_WORDSTART 0x00100000
- #define wxSTC_FIND_REGEXP 0x00200000
- #define wxSTC_FIND_POSIX 0x00400000
- #define wxSTC_FOLDLEVELBASE 0x400
- #define wxSTC_FOLDLEVELWHITEFLAG 0x1000
- #define wxSTC_FOLDLEVELHEADERFLAG 0x2000
- #define wxSTC_FOLDLEVELNUMBERMASK 0x0FFF
- #define wxSTC_FOLDACTION_CONTRACT 0
- #define wxSTC_FOLDACTION_EXPAND 1

- #define wxSTC_FOLDACTION_TOGGLE 2
- #define wxSTC_AUTOMATICFOLD_SHOW 0x0001
- #define wxSTC_AUTOMATICFOLD_CLICK 0x0002
- #define wxSTC_AUTOMATICFOLD_CHANGE 0x0004
- #define wxSTC_FOLDFLAG_LINEBEFORE_EXPANDED 0x0002
- #define wxSTC_FOLDFLAG_LINEBEFORE_CONTRACTED 0x0004
- #define wxSTC_FOLDFLAG_LINEAFTER_EXPANDED 0x0008
- #define wxSTC_FOLDFLAG_LINEAFTER_CONTRACTED 0x0010
- #define wxSTC_FOLDFLAG_LEVELNUMBERS 0x0040
- #define wxSTC_TIME_FOREVER 10000000
- #define wxSTC_WRAP_NONE 0
- #define wxSTC_WRAP_WORD 1
- #define wxSTC_WRAP_CHAR 2
- #define wxSTC_WRAP_WHITESPACE 3
- #define wxSTC_WRAPVISUALFLAG_NONE 0x0000
- #define wxSTC_WRAPVISUALFLAG_END 0x0001
- #define wxSTC_WRAPVISUALFLAG_START 0x0002
- #define wxSTC_WRAPVISUALFLAG_MARGIN 0x0004
- #define wxSTC_WRAPVISUALFLAGLOC_DEFAULT 0x0000
- #define wxSTC_WRAPVISUALFLAGLOC_END_BY_TEXT 0x0001
- #define wxSTC_WRAPVISUALFLAGLOC_START_BY_TEXT 0x0002
- #define wxSTC_WRAPINDENT_FIXED 0
- #define wxSTC_WRAPINDENT_SAME 1
- #define wxSTC_WRAPINDENT_INDENT 2
- #define wxSTC_CACHE_NONE 0
- #define wxSTC_CACHE_CARET 1
- #define wxSTC_CACHE_PAGE 2
- #define wxSTC_CACHE_DOCUMENT 3
- #define wxSTC_EFF_QUALITY_MASK 0xF

  Control font anti-aliasing.
- #define wxSTC_EFF_QUALITY_DEFAULT 0
- #define wxSTC_EFF_QUALITY_NON_ANTIALIASED 1
- #define wxSTC_EFF_QUALITY_ANTIALIASED 2
- #define wxSTC_EFF_QUALITY_LCD_OPTIMIZED 3
- #define wxSTC_MULTIPASTE_ONCE 0
- #define wxSTC_MULTIPASTE_EACH 1
- #define wxSTC_EDGE_NONE 0
- #define wxSTC_EDGE_LINE 1
- #define wxSTC_EDGE_BACKGROUND 2
- #define wxSTC_STATUS_OK 0
- #define wxSTC_STATUS_FAILURE 1
- #define wxSTC_STATUS_BADALLOC 2
- #define wxSTC_CURSORNORMAL -1
- #define wxSTC_CURSORARROW 2
- #define wxSTC_CURSORWAIT 4
- #define wxSTC_CURSORREVERSEARROW 7
- #define wxSTC_VISIBLE_SLOP 0x01

  Constants for use with SetVisiblePolicy, similar to SetCaretPolicy.
- #define wxSTC_VISIBLE_STRICT 0x04
- #define wxSTC_CARET_SLOP 0x01

  Caret policy, used by SetXCaretPolicy and SetYCaretPolicy.
- #define wxSTC_CARET_STRICT 0x04

  If CARET_STRICT is set, the policy is enforced...
- #define wxSTC_CARET_JUMPS 0x10

If CARET_JUMPS is set, the display is moved more energetically so the caret can move in the same direction longer before the policy is applied again.

- #define wxSTC_CARET_EVEN 0x08

  If CARET_EVEN is not set, instead of having symmetrical UZs, the left and bottom UZs are extended up to right and top UZs respectively.
- #define wxSTC_SEL_STREAM 0
- #define wxSTC_SEL_RECTANGLE 1
- #define wxSTC_SEL_LINES 2
- #define wxSTC_SEL_THIN 3
- #define wxSTC_CASEINSENSITIVEBEHAVIOUR_RESPECTCASE 0
- #define wxSTC_CASEINSENSITIVEBEHAVIOUR_IGNORECASE 1
- #define wxSTC_ORDER_PRESORTED 0
- #define wxSTC_ORDER_PERFORMSORT 1
- #define wxSTC_ORDER_CUSTOM 2
- #define wxSTC_CARETSTICKY_OFF 0
- #define wxSTC_CARETSTICKY_ON 1
- #define wxSTC_CARETSTICKY_WHITESPACE 2
- #define wxSTC_ALPHA_TRANSPARENT 0
- #define wxSTC_ALPHA_OPAQUE 255
- #define wxSTC_ALPHA_NOALPHA 256
- #define wxSTC_CARETSTYLE_INVISIBLE 0
- #define wxSTC_CARETSTYLE_LINE 1
- #define wxSTC_CARETSTYLE_BLOCK 2
- #define wxSTC_MARGINOPTION_NONE 0
- #define wxSTC_MARGINOPTION_SUBLINESELECT 1
- #define wxSTC_ANNOTATION_HIDDEN 0
- #define wxSTC_ANNOTATION_STANDARD 1
- #define wxSTC_ANNOTATION_BOXED 2
- #define wxSTC_UNDO_MAY_COALESCE 1
- #define wxSTC_SCVS_NONE 0
- #define wxSTC_SCVS_RECTANGULARSELECTION 1
- #define wxSTC_SCVS_USERACCESSIBLE 2
- #define wxSTC_TECHNOLOGY_DEFAULT 0
- #define wxSTC_TECHNOLOGY_DIRECTWRITE 1
- #define wxSTC_LINE_END_TYPE_DEFAULT 0

  Line end types which may be used in addition to LF, CR, and CRLF SC_LINE_END_TYPE_UNICODE includes U+2028 Line Separator, U+2029 Paragraph Separator, and U+0085 Next Line.
- #define wxSTC_LINE_END_TYPE_UNICODE 1
- #define wxSTC_KEYWORDSET_MAX 8

  Maximum value of keywordSet parameter of SetKeyWords.
- #define wxSTC_TYPE_BOOLEAN 0
- #define wxSTC_TYPE_INTEGER 1
- #define wxSTC_TYPE_STRING 2
- #define wxSTC_MOD_INSERTTEXT 0x1

  Notifications Type of modification and the action which caused the modification.
- #define wxSTC_MOD_DELETETEXT 0x2
- #define wxSTC_MOD_CHANGESTYLE 0x4
- #define wxSTC_MOD_CHANGEFOLD 0x8
- #define wxSTC_PERFORMED_USER 0x10
- #define wxSTC_PERFORMED_UNDO 0x20
- #define wxSTC_PERFORMED_REDO 0x40
- #define wxSTC_MULTISTEPUNDOREDO 0x80
- #define wxSTC_LASTSTEPINUNDOREDO 0x100
- #define wxSTC_MOD_CHANGEMARKER 0x200

- #define wxSTC_MOD_BEFOREINSERT 0x400
- #define wxSTC_MOD_BEFOREDELETE 0x800
- #define wxSTC_MULTILINEUNDOREDO 0x1000
- #define wxSTC_STARTACTION 0x2000
- #define wxSTC_MOD_CHANGEINDICATOR 0x4000
- #define wxSTC_MOD_CHANGELINESTATE 0x8000
- #define wxSTC_MOD_CHANGEMARGIN 0x10000
- #define wxSTC_MOD_CHANGEANNOTATION 0x20000
- #define wxSTC_MOD_CONTAINER 0x40000
- #define wxSTC_MOD_LEXERSTATE 0x80000
- #define wxSTC_MODEVENTMASKALL 0xFFFFF
- #define wxSTC_UPDATE_CONTENT 0x1
- #define wxSTC_UPDATE_SELECTION 0x2
- #define wxSTC_UPDATE_V_SCROLL 0x4
- #define wxSTC_UPDATE_H_SCROLL 0x8
- #define wxSTC_KEY_DOWN 300

  Symbolic key codes and modifier flags.
- #define wxSTC_KEY_UP 301
- #define wxSTC_KEY_LEFT 302
- #define wxSTC_KEY_RIGHT 303
- #define wxSTC_KEY_HOME 304
- #define wxSTC_KEY_END 305
- #define wxSTC_KEY_PRIOR 306
- #define wxSTC_KEY_NEXT 307
- #define wxSTC_KEY_DELETE 308
- #define wxSTC_KEY_INSERT 309
- #define wxSTC_KEY_ESCAPE 7
- #define wxSTC_KEY_BACK 8
- #define wxSTC_KEY_TAB 9
- #define wxSTC_KEY_RETURN 13
- #define wxSTC_KEY_ADD 310
- #define wxSTC_KEY_SUBTRACT 311
- #define wxSTC_KEY_DIVIDE 312
- #define wxSTC_KEY_WIN 313
- #define wxSTC_KEY_RWIN 314
- #define wxSTC_KEY_MENU 315
- #define wxSTC_SCMOD_NORM 0
- #define wxSTC_SCMOD_SHIFT 1
- #define wxSTC_SCMOD_CTRL 2
- #define wxSTC_SCMOD_ALT 4
- #define wxSTC_SCMOD_SUPER 8
- #define wxSTC_SCMOD_META 16
- #define wxSTC_LEX_CONTAINER 0

  For SciLexer.h.
- #define wxSTC_LEX_NULL 1
- #define wxSTC_LEX_PYTHON 2
- #define wxSTC_LEX_CPP 3
- #define wxSTC_LEX_HTML 4
- #define wxSTC_LEX_XML 5
- #define wxSTC_LEX_PERL 6
- #define wxSTC_LEX_SQL 7
- #define wxSTC_LEX_VB 8
- #define wxSTC_LEX_PROPERTIES 9
- #define wxSTC_LEX_ERRORLIST 10

- #define wxSTC_LEX_MAKEFILE 11
- #define wxSTC_LEX_BATCH 12
- #define wxSTC_LEX_XCODE 13
- #define wxSTC_LEX_LATEX 14
- #define wxSTC_LEX_LUA 15
- #define wxSTC_LEX_DIFF 16
- #define wxSTC_LEX_CONF 17
- #define wxSTC_LEX_PASCAL 18
- #define wxSTC_LEX_AVE 19
- #define wxSTC_LEX_ADA 20
- #define wxSTC_LEX_LISP 21
- #define wxSTC_LEX_RUBY 22
- #define wxSTC_LEX_EIFFEL 23
- #define wxSTC_LEX_EIFFELKW 24
- #define wxSTC_LEX_TCL 25
- #define wxSTC_LEX_NNCRONTAB 26
- #define wxSTC_LEX_BULLANT 27
- #define wxSTC_LEX_VBSCRIPT 28
- #define wxSTC_LEX_BAAN 31
- #define wxSTC_LEX_MATLAB 32
- #define wxSTC_LEX_SCRIPTOL 33
- #define wxSTC_LEX_ASM 34
- #define wxSTC_LEX_CPPNOCASE 35
- #define wxSTC_LEX_FORTRAN 36
- #define wxSTC_LEX_F77 37
- #define wxSTC_LEX_CSS 38
- #define wxSTC_LEX_POV 39
- #define wxSTC_LEX_LOUT 40
- #define wxSTC_LEX_ESCRIPT 41
- #define wxSTC_LEX_PS 42
- #define wxSTC_LEX_NSIS 43
- #define wxSTC_LEX_MMIXAL 44
- #define wxSTC_LEX_CLW 45
- #define wxSTC_LEX_CLWNOCASE 46
- #define wxSTC_LEX_LOT 47
- #define wxSTC_LEX_YAML 48
- #define wxSTC_LEX_TEX 49
- #define wxSTC_LEX_METAPOST 50
- #define wxSTC_LEX_POWERBASIC 51
- #define wxSTC_LEX_FORTH 52
- #define wxSTC_LEX_ERLANG 53
- #define wxSTC_LEX_OCTAVE 54
- #define wxSTC_LEX_MSSQL 55
- #define wxSTC_LEX_VERILOG 56
- #define wxSTC_LEX_KIX 57
- #define wxSTC_LEX_GUI4CLI 58
- #define wxSTC_LEX_SPECMAN 59
- #define wxSTC_LEX_AU3 60
- #define wxSTC_LEX_APDL 61
- #define wxSTC_LEX_BASH 62
- #define wxSTC_LEX_ASN1 63
- #define wxSTC_LEX_VHDL 64
- #define wxSTC_LEX_CAML 65
- #define wxSTC_LEX_BLITZBASIC 66
- #define wxSTC_LEX_PUREBASIC 67

- #define wxSTC_LEX_HASKELL 68
- #define wxSTC_LEX_PHPSCRIPT 69
- #define wxSTC_LEX_TADS3 70
- #define wxSTC_LEX_REBOL 71
- #define wxSTC_LEX_SMALLTALK 72
- #define wxSTC_LEX_FLAGSHIP 73
- #define wxSTC_LEX_CSOUND 74
- #define wxSTC_LEX_FREEBASIC 75
- #define wxSTC_LEX_INNOSETUP 76
- #define wxSTC_LEX_OPAL 77
- #define wxSTC_LEX_SPICE 78
- #define wxSTC_LEX_D 79
- #define wxSTC_LEX_CMAKE 80
- #define wxSTC_LEX_GAP 81
- #define wxSTC_LEX_PLM 82
- #define wxSTC_LEX_PROGRESS 83
- #define wxSTC_LEX_ABAQUS 84
- #define wxSTC_LEX_ASYMPTOTE 85
- #define wxSTC_LEX_R 86
- #define wxSTC_LEX_MAGIK 87
- #define wxSTC_LEX_POWERSHELL 88
- #define wxSTC_LEX_MYSQL 89
- #define wxSTC_LEX_PO 90
- #define wxSTC_LEX_TAL 91
- #define wxSTC_LEX_COBOL 92
- #define wxSTC_LEX_TACL 93
- #define wxSTC_LEX_SORCUS 94
- #define wxSTC_LEX_POWERPRO 95
- #define wxSTC_LEX_NIMROD 96
- #define wxSTC_LEX_SML 97
- #define wxSTC_LEX_MARKDOWN 98
- #define wxSTC_LEX_TXT2TAGS 99
- #define wxSTC_LEX_A68K 100
- #define wxSTC_LEX_MODULA 101
- #define wxSTC_LEX_COFFEESCRIPT 102
- #define wxSTC_LEX_TCMD 103
- #define wxSTC_LEX_AVS 104
- #define wxSTC_LEX_ECL 105
- #define wxSTC_LEX_OSCRIPT 106
- #define wxSTC_LEX_VISUALPROLOG 107
- #define wxSTC_LEX_LITERATEHASKELL 108
- #define wxSTC_LEX_STTXT 109
- #define wxSTC_LEX_KVIRC 110
- #define wxSTC_LEX_RUST 111
- #define wxSTC_LEX_DMAP 112
- #define wxSTC_LEX_AS 113
- #define wxSTC_LEX_AUTOMATIC 1000

  When a lexer specifies its language as SCLEX_AUTOMATIC it receives a value assigned in sequence from SCLEX_AUTOMATIC+1.
- #define wxSTC_P_DEFAULT 0

  Lexical states for SCLEX_PYTHON.
- #define wxSTC_P_COMMENTLINE 1
- #define wxSTC_P_NUMBER 2
- #define wxSTC_P_STRING 3

- #define wxSTC_P_CHARACTER 4
- #define wxSTC_P_WORD 5
- #define wxSTC_P_TRIPLE 6
- #define wxSTC_P_TRIPLEDOUBLE 7
- #define wxSTC_P_CLASSNAME 8
- #define wxSTC_P_DEFNAME 9
- #define wxSTC_P_OPERATOR 10
- #define wxSTC_P_IDENTIFIER 11
- #define wxSTC_P_COMMENTBLOCK 12
- #define wxSTC_P_STRINGEOL 13
- #define wxSTC_P_WORD2 14
- #define wxSTC_P_DECORATOR 15
- #define wxSTC_C_DEFAULT 0

Lexical states for SCLEX_CPP.

- #define wxSTC_C_COMMENT 1
- #define wxSTC_C_COMMENTLINE 2
- #define wxSTC_C_COMMENTDOC 3
- #define wxSTC_C_NUMBER 4
- #define wxSTC_C_WORD 5
- #define wxSTC_C_STRING 6
- #define wxSTC_C_CHARACTER 7
- #define wxSTC_C_UUID 8
- #define wxSTC_C_PREPROCESSOR 9
- #define wxSTC_C_OPERATOR 10
- #define wxSTC_C_IDENTIFIER 11
- #define wxSTC_C_STRINGEOL 12
- #define wxSTC_C_VERBATIM 13
- #define wxSTC_C_REGEX 14
- #define wxSTC_C_COMMENTLINEDOC 15
- #define wxSTC_C_WORD2 16
- #define wxSTC_C_COMMENTDOCKEYWORD 17
- #define wxSTC_C_COMMENTDOCKEYWORDERROR 18
- #define wxSTC_C_GLOBALCLASS 19
- #define wxSTC_C_STRINGRAW 20
- #define wxSTC_C_TRIPLEVERBATIM 21
- #define wxSTC_C_HASHQUOTEDSTRING 22
- #define wxSTC_C_PREPROCESSORCOMMENT 23
- #define wxSTC_C_PREPROCESSORCOMMENTDOC 24
- #define wxSTC_C_USERLITERAL 25
- #define wxSTC_D_DEFAULT 0

Lexical states for SCLEX_D.

- #define wxSTC_D_COMMENT 1
- #define wxSTC_D_COMMENTLINE 2
- #define wxSTC_D_COMMENTDOC 3
- #define wxSTC_D_COMMENTNESTED 4
- #define wxSTC_D_NUMBER 5
- #define wxSTC_D_WORD 6
- #define wxSTC_D_WORD2 7
- #define wxSTC_D_WORD3 8
- #define wxSTC_D_TYPEDEF 9
- #define wxSTC_D_STRING 10
- #define wxSTC_D_STRINGEOL 11
- #define wxSTC_D_CHARACTER 12
- #define wxSTC_D_OPERATOR 13

- #define wxSTC_D_IDENTIFIER 14
- #define wxSTC_D_COMMENTLINEDOC 15
- #define wxSTC_D_COMMENTDOCKEYWORD 16
- #define wxSTC_D_COMMENTDOCKEYWORDERROR 17
- #define wxSTC_D_STRINGB 18
- #define wxSTC_D_STRINGR 19
- #define wxSTC_D_WORD5 20
- #define wxSTC_D_WORD6 21
- #define wxSTC_D_WORD7 22
- #define wxSTC_TCL_DEFAULT 0

  Lexical states for SCLEX_TCL.
- #define wxSTC_TCL_COMMENT 1
- #define wxSTC_TCL_COMMENTLINE 2
- #define wxSTC_TCL_NUMBER 3
- #define wxSTC_TCL_WORD_IN_QUOTE 4
- #define wxSTC_TCL_IN_QUOTE 5
- #define wxSTC_TCL_OPERATOR 6
- #define wxSTC_TCL_IDENTIFIER 7
- #define wxSTC_TCL_SUBSTITUTION 8
- #define wxSTC_TCL_SUB_BRACE 9
- #define wxSTC_TCL_MODIFIER 10
- #define wxSTC_TCL_EXPAND 11
- #define wxSTC_TCL_WORD 12
- #define wxSTC_TCL_WORD2 13
- #define wxSTC_TCL_WORD3 14
- #define wxSTC_TCL_WORD4 15
- #define wxSTC_TCL_WORD5 16
- #define wxSTC_TCL_WORD6 17
- #define wxSTC_TCL_WORD7 18
- #define wxSTC_TCL_WORD8 19
- #define wxSTC_TCL_COMMENT_BOX 20
- #define wxSTC_TCL_BLOCK_COMMENT 21
- #define wxSTC_H_DEFAULT 0

  Lexical states for SCLEX_HTML, SCLEX_XML.
- #define wxSTC_H_TAG 1
- #define wxSTC_H_TAGUNKNOWN 2
- #define wxSTC_H_ATTRIBUTE 3
- #define wxSTC_H_ATTRIBUTEUNKNOWN 4
- #define wxSTC_H_NUMBER 5
- #define wxSTC_H_DOUBLESTRING 6
- #define wxSTC_H_SINGLESTRING 7
- #define wxSTC_H_OTHER 8
- #define wxSTC_H_COMMENT 9
- #define wxSTC_H_ENTITY 10
- #define wxSTC_H_TAGEND 11

  XML and ASP.
- #define wxSTC_H_XMLSTART 12
- #define wxSTC_H_XMLEND 13
- #define wxSTC_H_SCRIPT 14
- #define wxSTC_H_ASP 15
- #define wxSTC_H_ASPAT 16
- #define wxSTC_H_CDATA 17
- #define wxSTC_H_QUESTION 18
- #define wxSTC_H_VALUE 19

More HTML.

- #define wxSTC_H_XCCOMMENT 20

  X-Code.

- #define wxSTC_H_SGML_DEFAULT 21

  SGML.

- #define wxSTC_H_SGML_COMMAND 22
- #define wxSTC_H_SGML_1ST_PARAM 23
- #define wxSTC_H_SGML_DOUBLESTRING 24
- #define wxSTC_H_SGML_SIMPLESTRING 25
- #define wxSTC_H_SGML_ERROR 26
- #define wxSTC_H_SGML_SPECIAL 27
- #define wxSTC_H_SGML_ENTITY 28
- #define wxSTC_H_SGML_COMMENT 29
- #define wxSTC_H_SGML_1ST_PARAM_COMMENT 30
- #define wxSTC_H_SGML_BLOCK_DEFAULT 31
- #define wxSTC_HJ_START 40

  Embedded Javascript.

- #define wxSTC_HJ_DEFAULT 41
- #define wxSTC_HJ_COMMENT 42
- #define wxSTC_HJ_COMMENTLINE 43
- #define wxSTC_HJ_COMMENTDOC 44
- #define wxSTC_HJ_NUMBER 45
- #define wxSTC_HJ_WORD 46
- #define wxSTC_HJ_KEYWORD 47
- #define wxSTC_HJ_DOUBLESTRING 48
- #define wxSTC_HJ_SINGLESTRING 49
- #define wxSTC_HJ_SYMBOLS 50
- #define wxSTC_HJ_STRINGEOL 51
- #define wxSTC_HJ_REGEX 52
- #define wxSTC_HJA_START 55

  ASP Javascript.

- #define wxSTC_HJA_DEFAULT 56
- #define wxSTC_HJA_COMMENT 57
- #define wxSTC_HJA_COMMENTLINE 58
- #define wxSTC_HJA_COMMENTDOC 59
- #define wxSTC_HJA_NUMBER 60
- #define wxSTC_HJA_WORD 61
- #define wxSTC_HJA_KEYWORD 62
- #define wxSTC_HJA_DOUBLESTRING 63
- #define wxSTC_HJA_SINGLESTRING 64
- #define wxSTC_HJA_SYMBOLS 65
- #define wxSTC_HJA_STRINGEOL 66
- #define wxSTC_HJA_REGEX 67
- #define wxSTC_HB_START 70

  Embedded VBScript.

- #define wxSTC_HB_DEFAULT 71
- #define wxSTC_HB_COMMENTLINE 72
- #define wxSTC_HB_NUMBER 73
- #define wxSTC_HB_WORD 74
- #define wxSTC_HB_STRING 75
- #define wxSTC_HB_IDENTIFIER 76
- #define wxSTC_HB_STRINGEOL 77
- #define wxSTC_HBA_START 80

  ASP VBScript.

- #define wxSTC_HBA_DEFAULT 81
- #define wxSTC_HBA_COMMENTLINE 82
- #define wxSTC_HBA_NUMBER 83
- #define wxSTC_HBA_WORD 84
- #define wxSTC_HBA_STRING 85
- #define wxSTC_HBA_IDENTIFIER 86
- #define wxSTC_HBA_STRINGEOL 87
- #define wxSTC_HP_START 90

  Embedded Python.
- #define wxSTC_HP_DEFAULT 91
- #define wxSTC_HP_COMMENTLINE 92
- #define wxSTC_HP_NUMBER 93
- #define wxSTC_HP_STRING 94
- #define wxSTC_HP_CHARACTER 95
- #define wxSTC_HP_WORD 96
- #define wxSTC_HP_TRIPLE 97
- #define wxSTC_HP_TRIPLEDOUBLE 98
- #define wxSTC_HP_CLASSNAME 99
- #define wxSTC_HP_DEFNAME 100
- #define wxSTC_HP_OPERATOR 101
- #define wxSTC_HP_IDENTIFIER 102
- #define wxSTC_HPHP_COMPLEX_VARIABLE 104

  PHP.
- #define wxSTC_HPA_START 105

  ASP Python.
- #define wxSTC_HPA_DEFAULT 106
- #define wxSTC_HPA_COMMENTLINE 107
- #define wxSTC_HPA_NUMBER 108
- #define wxSTC_HPA_STRING 109
- #define wxSTC_HPA_CHARACTER 110
- #define wxSTC_HPA_WORD 111
- #define wxSTC_HPA_TRIPLE 112
- #define wxSTC_HPA_TRIPLEDOUBLE 113
- #define wxSTC_HPA_CLASSNAME 114
- #define wxSTC_HPA_DEFNAME 115
- #define wxSTC_HPA_OPERATOR 116
- #define wxSTC_HPA_IDENTIFIER 117
- #define wxSTC_HPHP_DEFAULT 118

  PHP.
- #define wxSTC_HPHP_HSTRING 119
- #define wxSTC_HPHP_SIMPLESTRING 120
- #define wxSTC_HPHP_WORD 121
- #define wxSTC_HPHP_NUMBER 122
- #define wxSTC_HPHP_VARIABLE 123
- #define wxSTC_HPHP_COMMENT 124
- #define wxSTC_HPHP_COMMENTLINE 125
- #define wxSTC_HPHP_HSTRING_VARIABLE 126
- #define wxSTC_HPHP_OPERATOR 127
- #define wxSTC_PL_DEFAULT 0

  Lexical states for SCLEX_PERL.
- #define wxSTC_PL_ERROR 1
- #define wxSTC_PL_COMMENTLINE 2
- #define wxSTC_PL_POD 3
- #define wxSTC_PL_NUMBER 4

- #define wxSTC_PL_WORD 5
- #define wxSTC_PL_STRING 6
- #define wxSTC_PL_CHARACTER 7
- #define wxSTC_PL_PUNCTUATION 8
- #define wxSTC_PL_PREPROCESSOR 9
- #define wxSTC_PL_OPERATOR 10
- #define wxSTC_PL_IDENTIFIER 11
- #define wxSTC_PL_SCALAR 12
- #define wxSTC_PL_ARRAY 13
- #define wxSTC_PL_HASH 14
- #define wxSTC_PL_SYMBOLTABLE 15
- #define wxSTC_PL_VARIABLE_INDEXER 16
- #define wxSTC_PL_REGEX 17
- #define wxSTC_PL_REGSUBST 18
- #define wxSTC_PL_LONGQUOTE 19
- #define wxSTC_PL_BACKTICKS 20
- #define wxSTC_PL_DATASECTION 21
- #define wxSTC_PL_HERE_DELIM 22
- #define wxSTC_PL_HERE_Q 23
- #define wxSTC_PL_HERE_QQ 24
- #define wxSTC_PL_HERE_QX 25
- #define wxSTC_PL_STRING_Q 26
- #define wxSTC_PL_STRING_QQ 27
- #define wxSTC_PL_STRING_QX 28
- #define wxSTC_PL_STRING_QR 29
- #define wxSTC_PL_STRING_QW 30
- #define wxSTC_PL_POD_VERB 31
- #define wxSTC_PL_SUB_PROTOTYPE 40
- #define wxSTC_PL_FORMAT_IDENT 41
- #define wxSTC_PL_FORMAT 42
- #define wxSTC_PL_STRING_VAR 43
- #define wxSTC_PL_XLAT 44
- #define wxSTC_PL_REGEX_VAR 54
- #define wxSTC_PL_REGSUBST_VAR 55
- #define wxSTC_PL_BACKTICKS_VAR 57
- #define wxSTC_PL_HERE_QQ_VAR 61
- #define wxSTC_PL_HERE_QX_VAR 62
- #define wxSTC_PL_STRING_QQ_VAR 64
- #define wxSTC_PL_STRING_QX_VAR 65
- #define wxSTC_PL_STRING_QR_VAR 66
- #define wxSTC_RB_DEFAULT 0

  Lexical states for SCLEX_RUBY.
- #define wxSTC_RB_ERROR 1
- #define wxSTC_RB_COMMENTLINE 2
- #define wxSTC_RB_POD 3
- #define wxSTC_RB_NUMBER 4
- #define wxSTC_RB_WORD 5
- #define wxSTC_RB_STRING 6
- #define wxSTC_RB_CHARACTER 7
- #define wxSTC_RB_CLASSNAME 8
- #define wxSTC_RB_DEFNAME 9
- #define wxSTC_RB_OPERATOR 10
- #define wxSTC_RB_IDENTIFIER 11
- #define wxSTC_RB_REGEX 12

- #define wxSTC_RB_GLOBAL 13
- #define wxSTC_RB_SYMBOL 14
- #define wxSTC_RB_MODULE_NAME 15
- #define wxSTC_RB_INSTANCE_VAR 16
- #define wxSTC_RB_CLASS_VAR 17
- #define wxSTC_RB_BACKTICKS 18
- #define wxSTC_RB_DATASECTION 19
- #define wxSTC_RB_HERE_DELIM 20
- #define wxSTC_RB_HERE_Q 21
- #define wxSTC_RB_HERE_QQ 22
- #define wxSTC_RB_HERE_QX 23
- #define wxSTC_RB_STRING_Q 24
- #define wxSTC_RB_STRING_QQ 25
- #define wxSTC_RB_STRING_QX 26
- #define wxSTC_RB_STRING_QR 27
- #define wxSTC_RB_STRING_QW 28
- #define wxSTC_RB_WORD_DEMOTED 29
- #define wxSTC_RB_STDIN 30
- #define wxSTC_RB_STDOUT 31
- #define wxSTC_RB_STDERR 40
- #define wxSTC_RB_UPPER_BOUND 41
- #define wxSTC_B_DEFAULT 0

  Lexical states for SCLEX_VB, SCLEX_VBSCRIPT, SCLEX_POWERBASIC.
- #define wxSTC_B_COMMENT 1
- #define wxSTC_B_NUMBER 2
- #define wxSTC_B_KEYWORD 3
- #define wxSTC_B_STRING 4
- #define wxSTC_B_PREPROCESSOR 5
- #define wxSTC_B_OPERATOR 6
- #define wxSTC_B_IDENTIFIER 7
- #define wxSTC_B_DATE 8
- #define wxSTC_B_STRINGEOL 9
- #define wxSTC_B_KEYWORD2 10
- #define wxSTC_B_KEYWORD3 11
- #define wxSTC_B_KEYWORD4 12
- #define wxSTC_B_CONSTANT 13
- #define wxSTC_B_ASM 14
- #define wxSTC_B_LABEL 15
- #define wxSTC_B_ERROR 16
- #define wxSTC_B_HEXNUMBER 17
- #define wxSTC_B_BINNUMBER 18
- #define wxSTC_B_COMMENTBLOCK 19
- #define wxSTC_B_DOCLINE 20
- #define wxSTC_B_DOCBLOCK 21
- #define wxSTC_B_DOCKEYWORD 22
- #define wxSTC_PROPS_DEFAULT 0

  Lexical states for SCLEX_PROPERTIES.
- #define wxSTC_PROPS_COMMENT 1
- #define wxSTC_PROPS_SECTION 2
- #define wxSTC_PROPS_ASSIGNMENT 3
- #define wxSTC_PROPS_DEFVAL 4
- #define wxSTC_PROPS_KEY 5
- #define wxSTC_L_DEFAULT 0

  Lexical states for SCLEX_LATEX.

- #define wxSTC_L_COMMAND 1
- #define wxSTC_L_TAG 2
- #define wxSTC_L_MATH 3
- #define wxSTC_L_COMMENT 4
- #define wxSTC_L_TAG2 5
- #define wxSTC_L_MATH2 6
- #define wxSTC_L_COMMENT2 7
- #define wxSTC_L_VERBATIM 8
- #define wxSTC_L_SHORTCMD 9
- #define wxSTC_L_SPECIAL 10
- #define wxSTC_L_CMDOPT 11
- #define wxSTC_L_ERROR 12
- #define wxSTC_LUA_DEFAULT 0

Lexical states for SCLEX_LUA.

- #define wxSTC_LUA_COMMENT 1
- #define wxSTC_LUA_COMMENTLINE 2
- #define wxSTC_LUA_COMMENTDOC 3
- #define wxSTC_LUA_NUMBER 4
- #define wxSTC_LUA_WORD 5
- #define wxSTC_LUA_STRING 6
- #define wxSTC_LUA_CHARACTER 7
- #define wxSTC_LUA_LITERALSTRING 8
- #define wxSTC_LUA_PREPROCESSOR 9
- #define wxSTC_LUA_OPERATOR 10
- #define wxSTC_LUA_IDENTIFIER 11
- #define wxSTC_LUA_STRINGEOL 12
- #define wxSTC_LUA_WORD2 13
- #define wxSTC_LUA_WORD3 14
- #define wxSTC_LUA_WORD4 15
- #define wxSTC_LUA_WORD5 16
- #define wxSTC_LUA_WORD6 17
- #define wxSTC_LUA_WORD7 18
- #define wxSTC_LUA_WORD8 19
- #define wxSTC_LUA_LABEL 20
- #define wxSTC_ERR_DEFAULT 0

Lexical states for SCLEX_ERRORLIST.

- #define wxSTC_ERR_PYTHON 1
- #define wxSTC_ERR_GCC 2
- #define wxSTC_ERR_MS 3
- #define wxSTC_ERR_CMD 4
- #define wxSTC_ERR_BORLAND 5
- #define wxSTC_ERR_PERL 6
- #define wxSTC_ERR_NET 7
- #define wxSTC_ERR_LUA 8
- #define wxSTC_ERR_CTAG 9
- #define wxSTC_ERR_DIFF_CHANGED 10
- #define wxSTC_ERR_DIFF_ADDITION 11
- #define wxSTC_ERR_DIFF_DELETION 12
- #define wxSTC_ERR_DIFF_MESSAGE 13
- #define wxSTC_ERR_PHP 14
- #define wxSTC_ERR_ELF 15
- #define wxSTC_ERR_IFC 16
- #define wxSTC_ERR_IFORT 17
- #define wxSTC_ERR_ABSF 18

- #define wxSTC_ERR_TIDY 19
- #define wxSTC_ERR_JAVA_STACK 20
- #define wxSTC_ERR_VALUE 21
- #define wxSTC_ERR_GCC_INCLUDED_FROM 22
- #define wxSTC_BAT_DEFAULT 0

  Lexical states for SCLEX_BATCH.
- #define wxSTC_BAT_COMMENT 1
- #define wxSTC_BAT_WORD 2
- #define wxSTC_BAT_LABEL 3
- #define wxSTC_BAT_HIDE 4
- #define wxSTC_BAT_COMMAND 5
- #define wxSTC_BAT_IDENTIFIER 6
- #define wxSTC_BAT_OPERATOR 7
- #define wxSTC_TCMD_DEFAULT 0

  Lexical states for SCLEX_TCMD.
- #define wxSTC_TCMD_COMMENT 1
- #define wxSTC_TCMD_WORD 2
- #define wxSTC_TCMD_LABEL 3
- #define wxSTC_TCMD_HIDE 4
- #define wxSTC_TCMD_COMMAND 5
- #define wxSTC_TCMD_IDENTIFIER 6
- #define wxSTC_TCMD_OPERATOR 7
- #define wxSTC_TCMD_ENVIRONMENT 8
- #define wxSTC_TCMD_EXPANSION 9
- #define wxSTC_TCMD_CLABEL 10
- #define wxSTC_MAKE_DEFAULT 0

  Lexical states for SCLEX_MAKEFILE.
- #define wxSTC_MAKE_COMMENT 1
- #define wxSTC_MAKE_PREPROCESSOR 2
- #define wxSTC_MAKE_IDENTIFIER 3
- #define wxSTC_MAKE_OPERATOR 4
- #define wxSTC_MAKE_TARGET 5
- #define wxSTC_MAKE_IDEOL 9
- #define wxSTC_DIFF_DEFAULT 0

  Lexical states for SCLEX_DIFF.
- #define wxSTC_DIFF_COMMENT 1
- #define wxSTC_DIFF_COMMAND 2
- #define wxSTC_DIFF_HEADER 3
- #define wxSTC_DIFF_POSITION 4
- #define wxSTC_DIFF_DELETED 5
- #define wxSTC_DIFF_ADDED 6
- #define wxSTC_DIFF_CHANGED 7
- #define wxSTC_CONF_DEFAULT 0

  Lexical states for SCLEX_CONF (Apache Configuration Files Lexer)
- #define wxSTC_CONF_COMMENT 1
- #define wxSTC_CONF_NUMBER 2
- #define wxSTC_CONF_IDENTIFIER 3
- #define wxSTC_CONF_EXTENSION 4
- #define wxSTC_CONF_PARAMETER 5
- #define wxSTC_CONF_STRING 6
- #define wxSTC_CONF_OPERATOR 7
- #define wxSTC_CONF_IP 8
- #define wxSTC_CONF_DIRECTIVE 9
- #define wxSTC_AVE_DEFAULT 0

Lexical states for SCLEX_AVE, Avenue.

- #define wxSTC_AVE_COMMENT 1
- #define wxSTC_AVE_NUMBER 2
- #define wxSTC_AVE_WORD 3
- #define wxSTC_AVE_STRING 6
- #define wxSTC_AVE_ENUM 7
- #define wxSTC_AVE_STRINGEOL 8
- #define wxSTC_AVE_IDENTIFIER 9
- #define wxSTC_AVE_OPERATOR 10
- #define wxSTC_AVE_WORD1 11
- #define wxSTC_AVE_WORD2 12
- #define wxSTC_AVE_WORD3 13
- #define wxSTC_AVE_WORD4 14
- #define wxSTC_AVE_WORD5 15
- #define wxSTC_AVE_WORD6 16
- #define wxSTC_ADA_DEFAULT 0

Lexical states for SCLEX_ADA.

- #define wxSTC_ADA_WORD 1
- #define wxSTC_ADA_IDENTIFIER 2
- #define wxSTC_ADA_NUMBER 3
- #define wxSTC_ADA_DELIMITER 4
- #define wxSTC_ADA_CHARACTER 5
- #define wxSTC_ADA_CHARACTEREOL 6
- #define wxSTC_ADA_STRING 7
- #define wxSTC_ADA_STRINGEOL 8
- #define wxSTC_ADA_LABEL 9
- #define wxSTC_ADA_COMMENTLINE 10
- #define wxSTC_ADA_ILLEGAL 11
- #define wxSTC_BAAN_DEFAULT 0

Lexical states for SCLEX_BAAN.

- #define wxSTC_BAAN_COMMENT 1
- #define wxSTC_BAAN_COMMENTDOC 2
- #define wxSTC_BAAN_NUMBER 3
- #define wxSTC_BAAN_WORD 4
- #define wxSTC_BAAN_STRING 5
- #define wxSTC_BAAN_PREPROCESSOR 6
- #define wxSTC_BAAN_OPERATOR 7
- #define wxSTC_BAAN_IDENTIFIER 8
- #define wxSTC_BAAN_STRINGEOL 9
- #define wxSTC_BAAN_WORD2 10
- #define wxSTC_LISP_DEFAULT 0

Lexical states for SCLEX_LISP.

- #define wxSTC_LISP_COMMENT 1
- #define wxSTC_LISP_NUMBER 2
- #define wxSTC_LISP_KEYWORD 3
- #define wxSTC_LISP_KEYWORD_KW 4
- #define wxSTC_LISP_SYMBOL 5
- #define wxSTC_LISP_STRING 6
- #define wxSTC_LISP_STRINGEOL 8
- #define wxSTC_LISP_IDENTIFIER 9
- #define wxSTC_LISP_OPERATOR 10
- #define wxSTC_LISP_SPECIAL 11
- #define wxSTC_LISP_MULTI_COMMENT 12
- #define wxSTC_EIFFEL_DEFAULT 0

Lexical states for SCLEX_EIFFEL and SCLEX_EIFFELKW.

- #define wxSTC_EIFFEL_COMMENTLINE 1
- #define wxSTC_EIFFEL_NUMBER 2
- #define wxSTC_EIFFEL_WORD 3
- #define wxSTC_EIFFEL_STRING 4
- #define wxSTC_EIFFEL_CHARACTER 5
- #define wxSTC_EIFFEL_OPERATOR 6
- #define wxSTC_EIFFEL_IDENTIFIER 7
- #define wxSTC_EIFFEL_STRINGEOL 8
- #define wxSTC_NNCRONTAB_DEFAULT 0

Lexical states for SCLEX_NNCRONTAB (nnCron crontab Lexer)

- #define wxSTC_NNCRONTAB_COMMENT 1
- #define wxSTC_NNCRONTAB_TASK 2
- #define wxSTC_NNCRONTAB_SECTION 3
- #define wxSTC_NNCRONTAB_KEYWORD 4
- #define wxSTC_NNCRONTAB_MODIFIER 5
- #define wxSTC_NNCRONTAB_ASTERISK 6
- #define wxSTC_NNCRONTAB_NUMBER 7
- #define wxSTC_NNCRONTAB_STRING 8
- #define wxSTC_NNCRONTAB_ENVIRONMENT 9
- #define wxSTC_NNCRONTAB_IDENTIFIER 10
- #define wxSTC_FORTH_DEFAULT 0

Lexical states for SCLEX_FORTH (Forth Lexer)

- #define wxSTC_FORTH_COMMENT 1
- #define wxSTC_FORTH_COMMENT_ML 2
- #define wxSTC_FORTH_IDENTIFIER 3
- #define wxSTC_FORTH_CONTROL 4
- #define wxSTC_FORTH_KEYWORD 5
- #define wxSTC_FORTH_DEFWORD 6
- #define wxSTC_FORTH_PREWORD1 7
- #define wxSTC_FORTH_PREWORD2 8
- #define wxSTC_FORTH_NUMBER 9
- #define wxSTC_FORTH_STRING 10
- #define wxSTC_FORTH_LOCALE 11
- #define wxSTC_MATLAB_DEFAULT 0

Lexical states for SCLEX_MATLAB.

- #define wxSTC_MATLAB_COMMENT 1
- #define wxSTC_MATLAB_COMMAND 2
- #define wxSTC_MATLAB_NUMBER 3
- #define wxSTC_MATLAB_KEYWORD 4
- #define wxSTC_MATLAB_STRING 5

single quoted string

- #define wxSTC_MATLAB_OPERATOR 6
- #define wxSTC_MATLAB_IDENTIFIER 7
- #define wxSTC_MATLAB_DOUBLEQUOTESTRING 8
- #define wxSTC_SCRIPTOL_DEFAULT 0

Lexical states for SCLEX_SCRIPTOL.

- #define wxSTC_SCRIPTOL_WHITE 1
- #define wxSTC_SCRIPTOL_COMMENTLINE 2
- #define wxSTC_SCRIPTOL_PERSISTENT 3
- #define wxSTC_SCRIPTOL_CSTYLE 4
- #define wxSTC_SCRIPTOL_COMMENTBLOCK 5
- #define wxSTC_SCRIPTOL_NUMBER 6
- #define wxSTC_SCRIPTOL_STRING 7

- #define wxSTC_SCRIPTOL_CHARACTER 8
- #define wxSTC_SCRIPTOL_STRINGEOL 9
- #define wxSTC_SCRIPTOL_KEYWORD 10
- #define wxSTC_SCRIPTOL_OPERATOR 11
- #define wxSTC_SCRIPTOL_IDENTIFIER 12
- #define wxSTC_SCRIPTOL_TRIPLE 13
- #define wxSTC_SCRIPTOL_CLASSNAME 14
- #define wxSTC_SCRIPTOL_PREPROCESSOR 15
- #define wxSTC_ASM_DEFAULT 0

  Lexical states for SCLEX_ASM, SCLEX_AS.
- #define wxSTC_ASM_COMMENT 1
- #define wxSTC_ASM_NUMBER 2
- #define wxSTC_ASM_STRING 3
- #define wxSTC_ASM_OPERATOR 4
- #define wxSTC_ASM_IDENTIFIER 5
- #define wxSTC_ASM_CPUINSTRUCTION 6
- #define wxSTC_ASM_MATHINSTRUCTION 7
- #define wxSTC_ASM_REGISTER 8
- #define wxSTC_ASM_DIRECTIVE 9
- #define wxSTC_ASM_DIRECTIVEOPERAND 10
- #define wxSTC_ASM_COMMENTBLOCK 11
- #define wxSTC_ASM_CHARACTER 12
- #define wxSTC_ASM_STRINGEOL 13
- #define wxSTC_ASM_EXTINSTRUCTION 14
- #define wxSTC_ASM_COMMENTDIRECTIVE 15
- #define wxSTC_F_DEFAULT 0

  Lexical states for SCLEX_FORTRAN.
- #define wxSTC_F_COMMENT 1
- #define wxSTC_F_NUMBER 2
- #define wxSTC_F_STRING1 3
- #define wxSTC_F_STRING2 4
- #define wxSTC_F_STRINGEOL 5
- #define wxSTC_F_OPERATOR 6
- #define wxSTC_F_IDENTIFIER 7
- #define wxSTC_F_WORD 8
- #define wxSTC_F_WORD2 9
- #define wxSTC_F_WORD3 10
- #define wxSTC_F_PREPROCESSOR 11
- #define wxSTC_F_OPERATOR2 12
- #define wxSTC_F_LABEL 13
- #define wxSTC_F_CONTINUATION 14
- #define wxSTC_CSS_DEFAULT 0

  Lexical states for SCLEX_CSS.
- #define wxSTC_CSS_TAG 1
- #define wxSTC_CSS_CLASS 2
- #define wxSTC_CSS_PSEUDOCLASS 3
- #define wxSTC_CSS_UNKNOWN_PSEUDOCLASS 4
- #define wxSTC_CSS_OPERATOR 5
- #define wxSTC_CSS_IDENTIFIER 6
- #define wxSTC_CSS_UNKNOWN_IDENTIFIER 7
- #define wxSTC_CSS_VALUE 8
- #define wxSTC_CSS_COMMENT 9
- #define wxSTC_CSS_ID 10
- #define wxSTC_CSS_IMPORTANT 11

- #define wxSTC_CSS_DIRECTIVE 12
- #define wxSTC_CSS_DOUBLESTRING 13
- #define wxSTC_CSS_SINGLESTRING 14
- #define wxSTC_CSS_IDENTIFIER2 15
- #define wxSTC_CSS_ATTRIBUTE 16
- #define wxSTC_CSS_IDENTIFIER3 17
- #define wxSTC_CSS_PSEUDOELEMENT 18
- #define wxSTC_CSS_EXTENDED_IDENTIFIER 19
- #define wxSTC_CSS_EXTENDED_PSEUDOCLASS 20
- #define wxSTC_CSS_EXTENDED_PSEUDOELEMENT 21
- #define wxSTC_CSS_MEDIA 22
- #define wxSTC_CSS_VARIABLE 23
- #define wxSTC_POV_DEFAULT 0

  Lexical states for SCLEX_POV.
- #define wxSTC_POV_COMMENT 1
- #define wxSTC_POV_COMMENTLINE 2
- #define wxSTC_POV_NUMBER 3
- #define wxSTC_POV_OPERATOR 4
- #define wxSTC_POV_IDENTIFIER 5
- #define wxSTC_POV_STRING 6
- #define wxSTC_POV_STRINGEOL 7
- #define wxSTC_POV_DIRECTIVE 8
- #define wxSTC_POV_BADDIRECTIVE 9
- #define wxSTC_POV_WORD2 10
- #define wxSTC_POV_WORD3 11
- #define wxSTC_POV_WORD4 12
- #define wxSTC_POV_WORD5 13
- #define wxSTC_POV_WORD6 14
- #define wxSTC_POV_WORD7 15
- #define wxSTC_POV_WORD8 16
- #define wxSTC_LOUT_DEFAULT 0

  Lexical states for SCLEX_LOUT.
- #define wxSTC_LOUT_COMMENT 1
- #define wxSTC_LOUT_NUMBER 2
- #define wxSTC_LOUT_WORD 3
- #define wxSTC_LOUT_WORD2 4
- #define wxSTC_LOUT_WORD3 5
- #define wxSTC_LOUT_WORD4 6
- #define wxSTC_LOUT_STRING 7
- #define wxSTC_LOUT_OPERATOR 8
- #define wxSTC_LOUT_IDENTIFIER 9
- #define wxSTC_LOUT_STRINGEOL 10
- #define wxSTC_ESCRIPT_DEFAULT 0

  Lexical states for SCLEX_ESCRIPT.
- #define wxSTC_ESCRIPT_COMMENT 1
- #define wxSTC_ESCRIPT_COMMENTLINE 2
- #define wxSTC_ESCRIPT_COMMENTDOC 3
- #define wxSTC_ESCRIPT_NUMBER 4
- #define wxSTC_ESCRIPT_WORD 5
- #define wxSTC_ESCRIPT_STRING 6
- #define wxSTC_ESCRIPT_OPERATOR 7
- #define wxSTC_ESCRIPT_IDENTIFIER 8
- #define wxSTC_ESCRIPT_BRACE 9
- #define wxSTC_ESCRIPT_WORD2 10

- #define wxSTC_ESCRIPT_WORD3 11
- #define wxSTC_PS_DEFAULT 0

  Lexical states for SCLEX_PS.
- #define wxSTC_PS_COMMENT 1
- #define wxSTC_PS_DSC_COMMENT 2
- #define wxSTC_PS_DSC_VALUE 3
- #define wxSTC_PS_NUMBER 4
- #define wxSTC_PS_NAME 5
- #define wxSTC_PS_KEYWORD 6
- #define wxSTC_PS_LITERAL 7
- #define wxSTC_PS_IMMEVAL 8
- #define wxSTC_PS_PAREN_ARRAY 9
- #define wxSTC_PS_PAREN_DICT 10
- #define wxSTC_PS_PAREN_PROC 11
- #define wxSTC_PS_TEXT 12
- #define wxSTC_PS_HEXSTRING 13
- #define wxSTC_PS_BASE85STRING 14
- #define wxSTC_PS_BADSTRINGCHAR 15
- #define wxSTC_NSIS_DEFAULT 0

  Lexical states for SCLEX_NSIS.
- #define wxSTC_NSIS_COMMENT 1
- #define wxSTC_NSIS_STRINGDQ 2
- #define wxSTC_NSIS_STRINGLQ 3
- #define wxSTC_NSIS_STRINGRQ 4
- #define wxSTC_NSIS_FUNCTION 5
- #define wxSTC_NSIS_VARIABLE 6
- #define wxSTC_NSIS_LABEL 7
- #define wxSTC_NSIS_USERDEFINED 8
- #define wxSTC_NSIS_SECTIONDEF 9
- #define wxSTC_NSIS_SUBSECTIONDEF 10
- #define wxSTC_NSIS_IFDEFINEDEF 11
- #define wxSTC_NSIS_MACRODEF 12
- #define wxSTC_NSIS_STRINGVAR 13
- #define wxSTC_NSIS_NUMBER 14
- #define wxSTC_NSIS_SECTIONGROUP 15
- #define wxSTC_NSIS_PAGEEX 16
- #define wxSTC_NSIS_FUNCTIONDEF 17
- #define wxSTC_NSIS_COMMENTBOX 18
- #define wxSTC_MMIXAL_LEADWS 0

  Lexical states for SCLEX_MMIXAL.
- #define wxSTC_MMIXAL_COMMENT 1
- #define wxSTC_MMIXAL_LABEL 2
- #define wxSTC_MMIXAL_OPCODE 3
- #define wxSTC_MMIXAL_OPCODE_PRE 4
- #define wxSTC_MMIXAL_OPCODE_VALID 5
- #define wxSTC_MMIXAL_OPCODE_UNKNOWN 6
- #define wxSTC_MMIXAL_OPCODE_POST 7
- #define wxSTC_MMIXAL_OPERANDS 8
- #define wxSTC_MMIXAL_NUMBER 9
- #define wxSTC_MMIXAL_REF 10
- #define wxSTC_MMIXAL_CHAR 11
- #define wxSTC_MMIXAL_STRING 12
- #define wxSTC_MMIXAL_REGISTER 13
- #define wxSTC_MMIXAL_HEX 14

- #define wxSTC_MMIXAL_OPERATOR 15
- #define wxSTC_MMIXAL_SYMBOL 16
- #define wxSTC_MMIXAL_INCLUDE 17
- #define wxSTC_CLW_DEFAULT 0

  Lexical states for SCLEX_CLW.
- #define wxSTC_CLW_LABEL 1
- #define wxSTC_CLW_COMMENT 2
- #define wxSTC_CLW_STRING 3
- #define wxSTC_CLW_USER_IDENTIFIER 4
- #define wxSTC_CLW_INTEGER_CONSTANT 5
- #define wxSTC_CLW_REAL_CONSTANT 6
- #define wxSTC_CLW_PICTURE_STRING 7
- #define wxSTC_CLW_KEYWORD 8
- #define wxSTC_CLW_COMPILER_DIRECTIVE 9
- #define wxSTC_CLW_RUNTIME_EXPRESSIONS 10
- #define wxSTC_CLW_BUILTIN_PROCEDURES_FUNCTION 11
- #define wxSTC_CLW_STRUCTURE_DATA_TYPE 12
- #define wxSTC_CLW_ATTRIBUTE 13
- #define wxSTC_CLW_STANDARD_EQUATE 14
- #define wxSTC_CLW_ERROR 15
- #define wxSTC_CLW_DEPRECATED 16
- #define wxSTC_LOT_DEFAULT 0

  Lexical states for SCLEX_LOT.
- #define wxSTC_LOT_HEADER 1
- #define wxSTC_LOT_BREAK 2
- #define wxSTC_LOT_SET 3
- #define wxSTC_LOT_PASS 4
- #define wxSTC_LOT_FAIL 5
- #define wxSTC_LOT_ABORT 6
- #define wxSTC_YAML_DEFAULT 0

  Lexical states for SCLEX_YAML.
- #define wxSTC_YAML_COMMENT 1
- #define wxSTC_YAML_IDENTIFIER 2
- #define wxSTC_YAML_KEYWORD 3
- #define wxSTC_YAML_NUMBER 4
- #define wxSTC_YAML_REFERENCE 5
- #define wxSTC_YAML_DOCUMENT 6
- #define wxSTC_YAML_TEXT 7
- #define wxSTC_YAML_ERROR 8
- #define wxSTC_YAML_OPERATOR 9
- #define wxSTC_TEX_DEFAULT 0

  Lexical states for SCLEX_TEX.
- #define wxSTC_TEX_SPECIAL 1
- #define wxSTC_TEX_GROUP 2
- #define wxSTC_TEX_SYMBOL 3
- #define wxSTC_TEX_COMMAND 4
- #define wxSTC_TEX_TEXT 5
- #define wxSTC_METAPOST_DEFAULT 0
- #define wxSTC_METAPOST_SPECIAL 1
- #define wxSTC_METAPOST_GROUP 2
- #define wxSTC_METAPOST_SYMBOL 3
- #define wxSTC_METAPOST_COMMAND 4
- #define wxSTC_METAPOST_TEXT 5
- #define wxSTC_METAPOST_EXTRA 6

- #define wxSTC_ERLANG_DEFAULT 0

  Lexical states for SCLEX_ERLANG.
- #define wxSTC_ERLANG_COMMENT 1
- #define wxSTC_ERLANG_VARIABLE 2
- #define wxSTC_ERLANG_NUMBER 3
- #define wxSTC_ERLANG_KEYWORD 4
- #define wxSTC_ERLANG_STRING 5
- #define wxSTC_ERLANG_OPERATOR 6
- #define wxSTC_ERLANG_ATOM 7
- #define wxSTC_ERLANG_FUNCTION_NAME 8
- #define wxSTC_ERLANG_CHARACTER 9
- #define wxSTC_ERLANG_MACRO 10
- #define wxSTC_ERLANG_RECORD 11
- #define wxSTC_ERLANG_PREPROC 12
- #define wxSTC_ERLANG_NODE_NAME 13
- #define wxSTC_ERLANG_COMMENT_FUNCTION 14
- #define wxSTC_ERLANG_COMMENT_MODULE 15
- #define wxSTC_ERLANG_COMMENT_DOC 16
- #define wxSTC_ERLANG_COMMENT_DOC_MACRO 17
- #define wxSTC_ERLANG_ATOM_QUOTED 18
- #define wxSTC_ERLANG_MACRO_QUOTED 19
- #define wxSTC_ERLANG_RECORD_QUOTED 20
- #define wxSTC_ERLANG_NODE_NAME_QUOTED 21
- #define wxSTC_ERLANG_BIFS 22
- #define wxSTC_ERLANG_MODULES 23
- #define wxSTC_ERLANG_MODULES_ATT 24
- #define wxSTC_ERLANG_UNKNOWN 31
- #define wxSTC_MSSQL_DEFAULT 0

  Lexical states for SCLEX_OCTAVE are identical to MatLab Lexical states for SCLEX_MSSQL.
- #define wxSTC_MSSQL_COMMENT 1
- #define wxSTC_MSSQL_LINE_COMMENT 2
- #define wxSTC_MSSQL_NUMBER 3
- #define wxSTC_MSSQL_STRING 4
- #define wxSTC_MSSQL_OPERATOR 5
- #define wxSTC_MSSQL_IDENTIFIER 6
- #define wxSTC_MSSQL_VARIABLE 7
- #define wxSTC_MSSQL_COLUMN_NAME 8
- #define wxSTC_MSSQL_STATEMENT 9
- #define wxSTC_MSSQL_DATATYPE 10
- #define wxSTC_MSSQL_SYSTABLE 11
- #define wxSTC_MSSQL_GLOBAL_VARIABLE 12
- #define wxSTC_MSSQL_FUNCTION 13
- #define wxSTC_MSSQL_STORED_PROCEDURE 14
- #define wxSTC_MSSQL_DEFAULT_PREF_DATATYPE 15
- #define wxSTC_MSSQL_COLUMN_NAME_2 16
- #define wxSTC_V_DEFAULT 0

  Lexical states for SCLEX_VERILOG.
- #define wxSTC_V_COMMENT 1
- #define wxSTC_V_COMMENTLINE 2
- #define wxSTC_V_COMMENTLINEBANG 3
- #define wxSTC_V_NUMBER 4
- #define wxSTC_V_WORD 5
- #define wxSTC_V_STRING 6
- #define wxSTC_V_WORD2 7

- #define wxSTC_V_WORD3 8
- #define wxSTC_V_PREPROCESSOR 9
- #define wxSTC_V_OPERATOR 10
- #define wxSTC_V_IDENTIFIER 11
- #define wxSTC_V_STRINGEOL 12
- #define wxSTC_V_USER 19
- #define wxSTC_KIX_DEFAULT 0

  Lexical states for SCLEX_KIX.
- #define wxSTC_KIX_COMMENT 1
- #define wxSTC_KIX_STRING1 2
- #define wxSTC_KIX_STRING2 3
- #define wxSTC_KIX_NUMBER 4
- #define wxSTC_KIX_VAR 5
- #define wxSTC_KIX_MACRO 6
- #define wxSTC_KIX_KEYWORD 7
- #define wxSTC_KIX_FUNCTIONS 8
- #define wxSTC_KIX_OPERATOR 9
- #define wxSTC_KIX_IDENTIFIER 31
- #define wxSTC_GC_DEFAULT 0

  Lexical states for SCLEX_GUI4CLI.
- #define wxSTC_GC_COMMENTLINE 1
- #define wxSTC_GC_COMMENTBLOCK 2
- #define wxSTC_GC_GLOBAL 3
- #define wxSTC_GC_EVENT 4
- #define wxSTC_GC_ATTRIBUTE 5
- #define wxSTC_GC_CONTROL 6
- #define wxSTC_GC_COMMAND 7
- #define wxSTC_GC_STRING 8
- #define wxSTC_GC_OPERATOR 9
- #define wxSTC_SN_DEFAULT 0

  Lexical states for SCLEX_SPECMAN.
- #define wxSTC_SN_CODE 1
- #define wxSTC_SN_COMMENTLINE 2
- #define wxSTC_SN_COMMENTLINEBANG 3
- #define wxSTC_SN_NUMBER 4
- #define wxSTC_SN_WORD 5
- #define wxSTC_SN_STRING 6
- #define wxSTC_SN_WORD2 7
- #define wxSTC_SN_WORD3 8
- #define wxSTC_SN_PREPROCESSOR 9
- #define wxSTC_SN_OPERATOR 10
- #define wxSTC_SN_IDENTIFIER 11
- #define wxSTC_SN_STRINGEOL 12
- #define wxSTC_SN_REGEXTAG 13
- #define wxSTC_SN_SIGNAL 14
- #define wxSTC_SN_USER 19
- #define wxSTC_AU3_DEFAULT 0

  Lexical states for SCLEX_AU3.
- #define wxSTC_AU3_COMMENT 1
- #define wxSTC_AU3_COMMENTBLOCK 2
- #define wxSTC_AU3_NUMBER 3
- #define wxSTC_AU3_FUNCTION 4
- #define wxSTC_AU3_KEYWORD 5
- #define wxSTC_AU3_MACRO 6

- #define wxSTC_AU3_STRING 7
- #define wxSTC_AU3_OPERATOR 8
- #define wxSTC_AU3_VARIABLE 9
- #define wxSTC_AU3_SENT 10
- #define wxSTC_AU3_PREPROCESSOR 11
- #define wxSTC_AU3_SPECIAL 12
- #define wxSTC_AU3_EXPAND 13
- #define wxSTC_AU3_COMOBJ 14
- #define wxSTC_AU3_UDF 15
- #define wxSTC_APDL_DEFAULT 0

  Lexical states for SCLEX_APDL.
- #define wxSTC_APDL_COMMENT 1
- #define wxSTC_APDL_COMMENTBLOCK 2
- #define wxSTC_APDL_NUMBER 3
- #define wxSTC_APDL_STRING 4
- #define wxSTC_APDL_OPERATOR 5
- #define wxSTC_APDL_WORD 6
- #define wxSTC_APDL_PROCESSOR 7
- #define wxSTC_APDL_COMMAND 8
- #define wxSTC_APDL_SLASHCOMMAND 9
- #define wxSTC_APDL_STARCOMMAND 10
- #define wxSTC_APDL_ARGUMENT 11
- #define wxSTC_APDL_FUNCTION 12
- #define wxSTC_SH_DEFAULT 0

  Lexical states for SCLEX_BASH.
- #define wxSTC_SH_ERROR 1
- #define wxSTC_SH_COMMENTLINE 2
- #define wxSTC_SH_NUMBER 3
- #define wxSTC_SH_WORD 4
- #define wxSTC_SH_STRING 5
- #define wxSTC_SH_CHARACTER 6
- #define wxSTC_SH_OPERATOR 7
- #define wxSTC_SH_IDENTIFIER 8
- #define wxSTC_SH_SCALAR 9
- #define wxSTC_SH_PARAM 10
- #define wxSTC_SH_BACKTICKS 11
- #define wxSTC_SH_HERE_DELIM 12
- #define wxSTC_SH_HERE_Q 13
- #define wxSTC_ASN1_DEFAULT 0

  Lexical states for SCLEX_ASN1.
- #define wxSTC_ASN1_COMMENT 1
- #define wxSTC_ASN1_IDENTIFIER 2
- #define wxSTC_ASN1_STRING 3
- #define wxSTC_ASN1_OID 4
- #define wxSTC_ASN1_SCALAR 5
- #define wxSTC_ASN1_KEYWORD 6
- #define wxSTC_ASN1_ATTRIBUTE 7
- #define wxSTC_ASN1_DESCRIPTOR 8
- #define wxSTC_ASN1_TYPE 9
- #define wxSTC_ASN1_OPERATOR 10
- #define wxSTC_VHDL_DEFAULT 0

  Lexical states for SCLEX_VHDL.
- #define wxSTC_VHDL_COMMENT 1
- #define wxSTC_VHDL_COMMENTLINEBANG 2

- #define wxSTC_VHDL_NUMBER 3
- #define wxSTC_VHDL_STRING 4
- #define wxSTC_VHDL_OPERATOR 5
- #define wxSTC_VHDL_IDENTIFIER 6
- #define wxSTC_VHDL_STRINGEOL 7
- #define wxSTC_VHDL_KEYWORD 8
- #define wxSTC_VHDL_STDOPERATOR 9
- #define wxSTC_VHDL_ATTRIBUTE 10
- #define wxSTC_VHDL_STDFUNCTION 11
- #define wxSTC_VHDL_STDPACKAGE 12
- #define wxSTC_VHDL_STDTYPE 13
- #define wxSTC_VHDL_USERWORD 14
- #define wxSTC_CAML_DEFAULT 0

  Lexical states for SCLEX_CAML.
- #define wxSTC_CAML_IDENTIFIER 1
- #define wxSTC_CAML_TAGNAME 2
- #define wxSTC_CAML_KEYWORD 3
- #define wxSTC_CAML_KEYWORD2 4
- #define wxSTC_CAML_KEYWORD3 5
- #define wxSTC_CAML_LINENUM 6
- #define wxSTC_CAML_OPERATOR 7
- #define wxSTC_CAML_NUMBER 8
- #define wxSTC_CAML_CHAR 9
- #define wxSTC_CAML_WHITE 10
- #define wxSTC_CAML_STRING 11
- #define wxSTC_CAML_COMMENT 12
- #define wxSTC_CAML_COMMENT1 13
- #define wxSTC_CAML_COMMENT2 14
- #define wxSTC_CAML_COMMENT3 15
- #define wxSTC_HA_DEFAULT 0

  Lexical states for SCLEX_HASKELL.
- #define wxSTC_HA_IDENTIFIER 1
- #define wxSTC_HA_KEYWORD 2
- #define wxSTC_HA_NUMBER 3
- #define wxSTC_HA_STRING 4
- #define wxSTC_HA_CHARACTER 5
- #define wxSTC_HA_CLASS 6
- #define wxSTC_HA_MODULE 7
- #define wxSTC_HA_CAPITAL 8
- #define wxSTC_HA_DATA 9
- #define wxSTC_HA_IMPORT 10
- #define wxSTC_HA_OPERATOR 11
- #define wxSTC_HA_INSTANCE 12
- #define wxSTC_HA_COMMENTLINE 13
- #define wxSTC_HA_COMMENTBLOCK 14
- #define wxSTC_HA_COMMENTBLOCK2 15
- #define wxSTC_HA_COMMENTBLOCK3 16
- #define wxSTC_HA_PRAGMA 17
- #define wxSTC_HA_PREPROCESSOR 18
- #define wxSTC_HA_STRINGEOL 19
- #define wxSTC_HA_RESERVED_OPERATOR 20
- #define wxSTC_HA_LITERATE_COMMENT 21
- #define wxSTC_HA_LITERATE_CODEDELIM 22
- #define wxSTC_T3_DEFAULT 0

Lexical states of SCLEX_TADS3.

- #define wxSTC_T3_X_DEFAULT 1
- #define wxSTC_T3_PREPROCESSOR 2
- #define wxSTC_T3_BLOCK_COMMENT 3
- #define wxSTC_T3_LINE_COMMENT 4
- #define wxSTC_T3_OPERATOR 5
- #define wxSTC_T3_KEYWORD 6
- #define wxSTC_T3_NUMBER 7
- #define wxSTC_T3_IDENTIFIER 8
- #define wxSTC_T3_S_STRING 9
- #define wxSTC_T3_D_STRING 10
- #define wxSTC_T3_X_STRING 11
- #define wxSTC_T3_LIB_DIRECTIVE 12
- #define wxSTC_T3_MSG_PARAM 13
- #define wxSTC_T3_HTML_TAG 14
- #define wxSTC_T3_HTML_DEFAULT 15
- #define wxSTC_T3_HTML_STRING 16
- #define wxSTC_T3_USER1 17
- #define wxSTC_T3_USER2 18
- #define wxSTC_T3_USER3 19
- #define wxSTC_T3_BRACE 20
- #define wxSTC_REBOL_DEFAULT 0

Lexical states for SCLEX_REBOL.

- #define wxSTC_REBOL_COMMENTLINE 1
- #define wxSTC_REBOL_COMMENTBLOCK 2
- #define wxSTC_REBOL_PREFACE 3
- #define wxSTC_REBOL_OPERATOR 4
- #define wxSTC_REBOL_CHARACTER 5
- #define wxSTC_REBOL_QUOTEDSTRING 6
- #define wxSTC_REBOL_BRACEDSTRING 7
- #define wxSTC_REBOL_NUMBER 8
- #define wxSTC_REBOL_PAIR 9
- #define wxSTC_REBOL_TUPLE 10
- #define wxSTC_REBOL_BINARY 11
- #define wxSTC_REBOL_MONEY 12
- #define wxSTC_REBOL_ISSUE 13
- #define wxSTC_REBOL_TAG 14
- #define wxSTC_REBOL_FILE 15
- #define wxSTC_REBOL_EMAIL 16
- #define wxSTC_REBOL_URL 17
- #define wxSTC_REBOL_DATE 18
- #define wxSTC_REBOL_TIME 19
- #define wxSTC_REBOL_IDENTIFIER 20
- #define wxSTC_REBOL_WORD 21
- #define wxSTC_REBOL_WORD2 22
- #define wxSTC_REBOL_WORD3 23
- #define wxSTC_REBOL_WORD4 24
- #define wxSTC_REBOL_WORD5 25
- #define wxSTC_REBOL_WORD6 26
- #define wxSTC_REBOL_WORD7 27
- #define wxSTC_REBOL_WORD8 28
- #define wxSTC_SQL_DEFAULT 0

Lexical states for SCLEX_SQL.

- #define wxSTC_SQL_COMMENT 1

- #define wxSTC_SQL_COMMENTLINE 2
- #define wxSTC_SQL_COMMENTDOC 3
- #define wxSTC_SQL_NUMBER 4
- #define wxSTC_SQL_WORD 5
- #define wxSTC_SQL_STRING 6
- #define wxSTC_SQL_CHARACTER 7
- #define wxSTC_SQL_SQLPLUS 8
- #define wxSTC_SQL_SQLPLUS_PROMPT 9
- #define wxSTC_SQL_OPERATOR 10
- #define wxSTC_SQL_IDENTIFIER 11
- #define wxSTC_SQL_SQLPLUS_COMMENT 13
- #define wxSTC_SQL_COMMENTLINEDOC 15
- #define wxSTC_SQL_WORD2 16
- #define wxSTC_SQL_COMMENTDOCKEYWORD 17
- #define wxSTC_SQL_COMMENTDOCKEYWORDERROR 18
- #define wxSTC_SQL_USER1 19
- #define wxSTC_SQL_USER2 20
- #define wxSTC_SQL_USER3 21
- #define wxSTC_SQL_USER4 22
- #define wxSTC_SQL_QUOTEDIDENTIFIER 23
- #define wxSTC_ST_DEFAULT 0

  Lexical states for SCLEX_SMALLTALK.
- #define wxSTC_ST_STRING 1
- #define wxSTC_ST_NUMBER 2
- #define wxSTC_ST_COMMENT 3
- #define wxSTC_ST_SYMBOL 4
- #define wxSTC_ST_BINARY 5
- #define wxSTC_ST_BOOL 6
- #define wxSTC_ST_SELF 7
- #define wxSTC_ST_SUPER 8
- #define wxSTC_ST_NIL 9
- #define wxSTC_ST_GLOBAL 10
- #define wxSTC_ST_RETURN 11
- #define wxSTC_ST_SPECIAL 12
- #define wxSTC_ST_KWSEND 13
- #define wxSTC_ST_ASSIGN 14
- #define wxSTC_ST_CHARACTER 15
- #define wxSTC_ST_SPEC_SEL 16
- #define wxSTC_FS_DEFAULT 0

  Lexical states for SCLEX_FLAGSHIP (clipper)
- #define wxSTC_FS_COMMENT 1
- #define wxSTC_FS_COMMENTLINE 2
- #define wxSTC_FS_COMMENTDOC 3
- #define wxSTC_FS_COMMENTLINEDOC 4
- #define wxSTC_FS_COMMENTDOCKEYWORD 5
- #define wxSTC_FS_COMMENTDOCKEYWORDERROR 6
- #define wxSTC_FS_KEYWORD 7
- #define wxSTC_FS_KEYWORD2 8
- #define wxSTC_FS_KEYWORD3 9
- #define wxSTC_FS_KEYWORD4 10
- #define wxSTC_FS_NUMBER 11
- #define wxSTC_FS_STRING 12
- #define wxSTC_FS_PREPROCESSOR 13
- #define wxSTC_FS_OPERATOR 14

- #define wxSTC_FS_IDENTIFIER 15
- #define wxSTC_FS_DATE 16
- #define wxSTC_FS_STRINGEOL 17
- #define wxSTC_FS_CONSTANT 18
- #define wxSTC_FS_WORDOPERATOR 19
- #define wxSTC_FS_DISABLEDCODE 20
- #define wxSTC_FS_DEFAULT_C 21
- #define wxSTC_FS_COMMENTDOC_C 22
- #define wxSTC_FS_COMMENTLINEDOC_C 23
- #define wxSTC_FS_KEYWORD_C 24
- #define wxSTC_FS_KEYWORD2_C 25
- #define wxSTC_FS_NUMBER_C 26
- #define wxSTC_FS_STRING_C 27
- #define wxSTC_FS_PREPROCESSOR_C 28
- #define wxSTC_FS_OPERATOR_C 29
- #define wxSTC_FS_IDENTIFIER_C 30
- #define wxSTC_FS_STRINGEOL_C 31
- #define wxSTC_CSOUND_DEFAULT 0

  Lexical states for SCLEX_CSOUND.
- #define wxSTC_CSOUND_COMMENT 1
- #define wxSTC_CSOUND_NUMBER 2
- #define wxSTC_CSOUND_OPERATOR 3
- #define wxSTC_CSOUND_INSTR 4
- #define wxSTC_CSOUND_IDENTIFIER 5
- #define wxSTC_CSOUND_OPCODE 6
- #define wxSTC_CSOUND_HEADERSTMT 7
- #define wxSTC_CSOUND_USERKEYWORD 8
- #define wxSTC_CSOUND_COMMENTBLOCK 9
- #define wxSTC_CSOUND_PARAM 10
- #define wxSTC_CSOUND_ARATE_VAR 11
- #define wxSTC_CSOUND_KRATE_VAR 12
- #define wxSTC_CSOUND_IRATE_VAR 13
- #define wxSTC_CSOUND_GLOBAL_VAR 14
- #define wxSTC_CSOUND_STRINGEOL 15
- #define wxSTC_INNO_DEFAULT 0

  Lexical states for SCLEX_INNOSETUP.
- #define wxSTC_INNO_COMMENT 1
- #define wxSTC_INNO_KEYWORD 2
- #define wxSTC_INNO_PARAMETER 3
- #define wxSTC_INNO_SECTION 4
- #define wxSTC_INNO_PREPROC 5
- #define wxSTC_INNO_INLINE_EXPANSION 6
- #define wxSTC_INNO_COMMENT_PASCAL 7
- #define wxSTC_INNO_KEYWORD_PASCAL 8
- #define wxSTC_INNO_KEYWORD_USER 9
- #define wxSTC_INNO_STRING_DOUBLE 10
- #define wxSTC_INNO_STRING_SINGLE 11
- #define wxSTC_INNO_IDENTIFIER 12
- #define wxSTC_OPAL_SPACE 0

  Lexical states for SCLEX_OPAL.
- #define wxSTC_OPAL_COMMENT_BLOCK 1
- #define wxSTC_OPAL_COMMENT_LINE 2
- #define wxSTC_OPAL_INTEGER 3
- #define wxSTC_OPAL_KEYWORD 4

- #define wxSTC_OPAL_SORT 5
- #define wxSTC_OPAL_STRING 6
- #define wxSTC_OPAL_PAR 7
- #define wxSTC_OPAL_BOOL_CONST 8
- #define wxSTC_OPAL_DEFAULT 32
- #define wxSTC_SPICE_DEFAULT 0

  Lexical states for SCLEX_SPICE.
- #define wxSTC_SPICE_IDENTIFIER 1
- #define wxSTC_SPICE_KEYWORD 2
- #define wxSTC_SPICE_KEYWORD2 3
- #define wxSTC_SPICE_KEYWORD3 4
- #define wxSTC_SPICE_NUMBER 5
- #define wxSTC_SPICE_DELIMITER 6
- #define wxSTC_SPICE_VALUE 7
- #define wxSTC_SPICE_COMMENTLINE 8
- #define wxSTC_CMAKE_DEFAULT 0

  Lexical states for SCLEX_CMAKE.
- #define wxSTC_CMAKE_COMMENT 1
- #define wxSTC_CMAKE_STRINGDQ 2
- #define wxSTC_CMAKE_STRINGLQ 3
- #define wxSTC_CMAKE_STRINGRQ 4
- #define wxSTC_CMAKE_COMMANDS 5
- #define wxSTC_CMAKE_PARAMETERS 6
- #define wxSTC_CMAKE_VARIABLE 7
- #define wxSTC_CMAKE_USERDEFINED 8
- #define wxSTC_CMAKE_WHILEDEF 9
- #define wxSTC_CMAKE_FOREACHDEF 10
- #define wxSTC_CMAKE_IFDEFINEDEF 11
- #define wxSTC_CMAKE_MACRODEF 12
- #define wxSTC_CMAKE_STRINGVAR 13
- #define wxSTC_CMAKE_NUMBER 14
- #define wxSTC_GAP_DEFAULT 0

  Lexical states for SCLEX_GAP.
- #define wxSTC_GAP_IDENTIFIER 1
- #define wxSTC_GAP_KEYWORD 2
- #define wxSTC_GAP_KEYWORD2 3
- #define wxSTC_GAP_KEYWORD3 4
- #define wxSTC_GAP_KEYWORD4 5
- #define wxSTC_GAP_STRING 6
- #define wxSTC_GAP_CHAR 7
- #define wxSTC_GAP_OPERATOR 8
- #define wxSTC_GAP_COMMENT 9
- #define wxSTC_GAP_NUMBER 10
- #define wxSTC_GAP_STRINGEOL 11
- #define wxSTC_PLM_DEFAULT 0

  Lexical state for SCLEX_PLM.
- #define wxSTC_PLM_COMMENT 1
- #define wxSTC_PLM_STRING 2
- #define wxSTC_PLM_NUMBER 3
- #define wxSTC_PLM_IDENTIFIER 4
- #define wxSTC_PLM_OPERATOR 5
- #define wxSTC_PLM_CONTROL 6
- #define wxSTC_PLM_KEYWORD 7
- #define wxSTC_4GL_DEFAULT 0

Lexical state for SCLEX_PROGRESS.

- #define wxSTC_4GL_NUMBER 1
- #define wxSTC_4GL_WORD 2
- #define wxSTC_4GL_STRING 3
- #define wxSTC_4GL_CHARACTER 4
- #define wxSTC_4GL_PREPROCESSOR 5
- #define wxSTC_4GL_OPERATOR 6
- #define wxSTC_4GL_IDENTIFIER 7
- #define wxSTC_4GL_BLOCK 8
- #define wxSTC_4GL_END 9
- #define wxSTC_4GL_COMMENT1 10
- #define wxSTC_4GL_COMMENT2 11
- #define wxSTC_4GL_COMMENT3 12
- #define wxSTC_4GL_COMMENT4 13
- #define wxSTC_4GL_COMMENT5 14
- #define wxSTC_4GL_COMMENT6 15
- #define wxSTC_4GL_DEFAULT_ 16
- #define wxSTC_4GL_NUMBER_ 17
- #define wxSTC_4GL_WORD_ 18
- #define wxSTC_4GL_STRING_ 19
- #define wxSTC_4GL_CHARACTER_ 20
- #define wxSTC_4GL_PREPROCESSOR_ 21
- #define wxSTC_4GL_OPERATOR_ 22
- #define wxSTC_4GL_IDENTIFIER_ 23
- #define wxSTC_4GL_BLOCK_ 24
- #define wxSTC_4GL_END_ 25
- #define wxSTC_4GL_COMMENT1_ 26
- #define wxSTC_4GL_COMMENT2_ 27
- #define wxSTC_4GL_COMMENT3_ 28
- #define wxSTC_4GL_COMMENT4_ 29
- #define wxSTC_4GL_COMMENT5_ 30
- #define wxSTC_4GL_COMMENT6_ 31
- #define wxSTC_ABAQUS_DEFAULT 0

Lexical states for SCLEX_ABAQUS.

- #define wxSTC_ABAQUS_COMMENT 1
- #define wxSTC_ABAQUS_COMMENTBLOCK 2
- #define wxSTC_ABAQUS_NUMBER 3
- #define wxSTC_ABAQUS_STRING 4
- #define wxSTC_ABAQUS_OPERATOR 5
- #define wxSTC_ABAQUS_WORD 6
- #define wxSTC_ABAQUS_PROCESSOR 7
- #define wxSTC_ABAQUS_COMMAND 8
- #define wxSTC_ABAQUS_SLASHCOMMAND 9
- #define wxSTC_ABAQUS_STARCOMMAND 10
- #define wxSTC_ABAQUS_ARGUMENT 11
- #define wxSTC_ABAQUS_FUNCTION 12
- #define wxSTC_ASY_DEFAULT 0

Lexical states for SCLEX_ASYMPTOTE.

- #define wxSTC_ASY_COMMENT 1
- #define wxSTC_ASY_COMMENTLINE 2
- #define wxSTC_ASY_NUMBER 3
- #define wxSTC_ASY_WORD 4
- #define wxSTC_ASY_STRING 5
- #define wxSTC_ASY_CHARACTER 6

- #define wxSTC_ASY_OPERATOR 7
- #define wxSTC_ASY_IDENTIFIER 8
- #define wxSTC_ASY_STRINGEOL 9
- #define wxSTC_ASY_COMMENTLINEDOC 10
- #define wxSTC_ASY_WORD2 11
- #define wxSTC_R_DEFAULT 0

  Lexical states for SCLEX_R.
- #define wxSTC_R_COMMENT 1
- #define wxSTC_R_KWORD 2
- #define wxSTC_R_BASEKWORD 3
- #define wxSTC_R_OTHERKWORD 4
- #define wxSTC_R_NUMBER 5
- #define wxSTC_R_STRING 6
- #define wxSTC_R_STRING2 7
- #define wxSTC_R_OPERATOR 8
- #define wxSTC_R_IDENTIFIER 9
- #define wxSTC_R_INFIX 10
- #define wxSTC_R_INFIXEOL 11
- #define wxSTC_MAGIK_DEFAULT 0

  Lexical state for SCLEX_MAGIK.
- #define wxSTC_MAGIK_COMMENT 1
- #define wxSTC_MAGIK_HYPER_COMMENT 16
- #define wxSTC_MAGIK_STRING 2
- #define wxSTC_MAGIK_CHARACTER 3
- #define wxSTC_MAGIK_NUMBER 4
- #define wxSTC_MAGIK_IDENTIFIER 5
- #define wxSTC_MAGIK_OPERATOR 6
- #define wxSTC_MAGIK_FLOW 7
- #define wxSTC_MAGIK_CONTAINER 8
- #define wxSTC_MAGIK_BRACKET_BLOCK 9
- #define wxSTC_MAGIK_BRACE_BLOCK 10
- #define wxSTC_MAGIK_SQBRACKET_BLOCK 11
- #define wxSTC_MAGIK_UNKNOWN_KEYWORD 12
- #define wxSTC_MAGIK_KEYWORD 13
- #define wxSTC_MAGIK_PRAGMA 14
- #define wxSTC_MAGIK_SYMBOL 15
- #define wxSTC_POWERSHELL_DEFAULT 0

  Lexical state for SCLEX_POWERSHELL.
- #define wxSTC_POWERSHELL_COMMENT 1
- #define wxSTC_POWERSHELL_STRING 2
- #define wxSTC_POWERSHELL_CHARACTER 3
- #define wxSTC_POWERSHELL_NUMBER 4
- #define wxSTC_POWERSHELL_VARIABLE 5
- #define wxSTC_POWERSHELL_OPERATOR 6
- #define wxSTC_POWERSHELL_IDENTIFIER 7
- #define wxSTC_POWERSHELL_KEYWORD 8
- #define wxSTC_POWERSHELL_CMDLET 9
- #define wxSTC_POWERSHELL_ALIAS 10
- #define wxSTC_POWERSHELL_FUNCTION 11
- #define wxSTC_POWERSHELL_USER1 12
- #define wxSTC_POWERSHELL_COMMENTSTREAM 13
- #define wxSTC_POWERSHELL_HERE_STRING 14
- #define wxSTC_POWERSHELL_HERE_CHARACTER 15
- #define wxSTC_POWERSHELL_COMMENTDOCKEYWORD 16

- #define wxSTC_MYSQL_DEFAULT 0

  Lexical state for SCLEX_MYSQL.
- #define wxSTC_MYSQL_COMMENT 1
- #define wxSTC_MYSQL_COMMENTLINE 2
- #define wxSTC_MYSQL_VARIABLE 3
- #define wxSTC_MYSQL_SYSTEMVARIABLE 4
- #define wxSTC_MYSQL_KNOWNSYSTEMVARIABLE 5
- #define wxSTC_MYSQL_NUMBER 6
- #define wxSTC_MYSQL_MAJORKEYWORD 7
- #define wxSTC_MYSQL_KEYWORD 8
- #define wxSTC_MYSQL_DATABASEOBJECT 9
- #define wxSTC_MYSQL_PROCEDUREKEYWORD 10
- #define wxSTC_MYSQL_STRING 11
- #define wxSTC_MYSQL_SQSTRING 12
- #define wxSTC_MYSQL_DQSTRING 13
- #define wxSTC_MYSQL_OPERATOR 14
- #define wxSTC_MYSQL_FUNCTION 15
- #define wxSTC_MYSQL_IDENTIFIER 16
- #define wxSTC_MYSQL_QUOTEDIDENTIFIER 17
- #define wxSTC_MYSQL_USER1 18
- #define wxSTC_MYSQL_USER2 19
- #define wxSTC_MYSQL_USER3 20
- #define wxSTC_MYSQL_HIDDENCOMMAND 21
- #define wxSTC_MYSQL_PLACEHOLDER 22
- #define wxSTC_PO_DEFAULT 0

  Lexical state for SCLEX_PO.
- #define wxSTC_PO_COMMENT 1
- #define wxSTC_PO_MSGID 2
- #define wxSTC_PO_MSGID_TEXT 3
- #define wxSTC_PO_MSGSTR 4
- #define wxSTC_PO_MSGSTR_TEXT 5
- #define wxSTC_PO_MSGCTXT 6
- #define wxSTC_PO_MSGCTXT_TEXT 7
- #define wxSTC_PO_FUZZY 8
- #define wxSTC_PO_PROGRAMMER_COMMENT 9
- #define wxSTC_PO_REFERENCE 10
- #define wxSTC_PO_FLAGS 11
- #define wxSTC_PO_MSGID_TEXT_EOL 12
- #define wxSTC_PO_MSGSTR_TEXT_EOL 13
- #define wxSTC_PO_MSGCTXT_TEXT_EOL 14
- #define wxSTC_PO_ERROR 15
- #define wxSTC_PAS_DEFAULT 0

  Lexical states for SCLEX_PASCAL.
- #define wxSTC_PAS_IDENTIFIER 1
- #define wxSTC_PAS_COMMENT 2
- #define wxSTC_PAS_COMMENT2 3
- #define wxSTC_PAS_COMMENTLINE 4
- #define wxSTC_PAS_PREPROCESSOR 5
- #define wxSTC_PAS_PREPROCESSOR2 6
- #define wxSTC_PAS_NUMBER 7
- #define wxSTC_PAS_HEXNUMBER 8
- #define wxSTC_PAS_WORD 9
- #define wxSTC_PAS_STRING 10
- #define wxSTC_PAS_STRINGEOL 11

- #define wxSTC_PAS_CHARACTER 12
- #define wxSTC_PAS_OPERATOR 13
- #define wxSTC_PAS_ASM 14
- #define wxSTC_SORCUS_DEFAULT 0

  Lexical state for SCLEX_SORCUS.
- #define wxSTC_SORCUS_COMMAND 1
- #define wxSTC_SORCUS_PARAMETER 2
- #define wxSTC_SORCUS_COMMENTLINE 3
- #define wxSTC_SORCUS_STRING 4
- #define wxSTC_SORCUS_STRINGEOL 5
- #define wxSTC_SORCUS_IDENTIFIER 6
- #define wxSTC_SORCUS_OPERATOR 7
- #define wxSTC_SORCUS_NUMBER 8
- #define wxSTC_SORCUS_CONSTANT 9
- #define wxSTC_POWERPRO_DEFAULT 0

  Lexical state for SCLEX_POWERPRO.
- #define wxSTC_POWERPRO_COMMENTBLOCK 1
- #define wxSTC_POWERPRO_COMMENTLINE 2
- #define wxSTC_POWERPRO_NUMBER 3
- #define wxSTC_POWERPRO_WORD 4
- #define wxSTC_POWERPRO_WORD2 5
- #define wxSTC_POWERPRO_WORD3 6
- #define wxSTC_POWERPRO_WORD4 7
- #define wxSTC_POWERPRO_DOUBLEQUOTEDSTRING 8
- #define wxSTC_POWERPRO_SINGLEQUOTEDSTRING 9
- #define wxSTC_POWERPRO_LINECONTINUE 10
- #define wxSTC_POWERPRO_OPERATOR 11
- #define wxSTC_POWERPRO_IDENTIFIER 12
- #define wxSTC_POWERPRO_STRINGEOL 13
- #define wxSTC_POWERPRO_VERBATIM 14
- #define wxSTC_POWERPRO_ALTQUOTE 15
- #define wxSTC_POWERPRO_FUNCTION 16
- #define wxSTC_SML_DEFAULT 0

  Lexical states for SCLEX_SML.
- #define wxSTC_SML_IDENTIFIER 1
- #define wxSTC_SML_TAGNAME 2
- #define wxSTC_SML_KEYWORD 3
- #define wxSTC_SML_KEYWORD2 4
- #define wxSTC_SML_KEYWORD3 5
- #define wxSTC_SML_LINENUM 6
- #define wxSTC_SML_OPERATOR 7
- #define wxSTC_SML_NUMBER 8
- #define wxSTC_SML_CHAR 9
- #define wxSTC_SML_STRING 11
- #define wxSTC_SML_COMMENT 12
- #define wxSTC_SML_COMMENT1 13
- #define wxSTC_SML_COMMENT2 14
- #define wxSTC_SML_COMMENT3 15
- #define wxSTC_MARKDOWN_DEFAULT 0

  Lexical state for SCLEX_MARKDOWN.
- #define wxSTC_MARKDOWN_LINE_BEGIN 1
- #define wxSTC_MARKDOWN_STRONG1 2
- #define wxSTC_MARKDOWN_STRONG2 3
- #define wxSTC_MARKDOWN_EM1 4

- #define wxSTC_MARKDOWN_EM2 5
- #define wxSTC_MARKDOWN_HEADER1 6
- #define wxSTC_MARKDOWN_HEADER2 7
- #define wxSTC_MARKDOWN_HEADER3 8
- #define wxSTC_MARKDOWN_HEADER4 9
- #define wxSTC_MARKDOWN_HEADER5 10
- #define wxSTC_MARKDOWN_HEADER6 11
- #define wxSTC_MARKDOWN_PRECHAR 12
- #define wxSTC_MARKDOWN_ULIST_ITEM 13
- #define wxSTC_MARKDOWN_OLIST_ITEM 14
- #define wxSTC_MARKDOWN_BLOCKQUOTE 15
- #define wxSTC_MARKDOWN_STRIKEOUT 16
- #define wxSTC_MARKDOWN_HRULE 17
- #define wxSTC_MARKDOWN_LINK 18
- #define wxSTC_MARKDOWN_CODE 19
- #define wxSTC_MARKDOWN_CODE2 20
- #define wxSTC_MARKDOWN_CODEBK 21
- #define wxSTC_TXT2TAGS_DEFAULT 0

  Lexical state for SCLEX_TXT2TAGS.
- #define wxSTC_TXT2TAGS_LINE_BEGIN 1
- #define wxSTC_TXT2TAGS_STRONG1 2
- #define wxSTC_TXT2TAGS_STRONG2 3
- #define wxSTC_TXT2TAGS_EM1 4
- #define wxSTC_TXT2TAGS_EM2 5
- #define wxSTC_TXT2TAGS_HEADER1 6
- #define wxSTC_TXT2TAGS_HEADER2 7
- #define wxSTC_TXT2TAGS_HEADER3 8
- #define wxSTC_TXT2TAGS_HEADER4 9
- #define wxSTC_TXT2TAGS_HEADER5 10
- #define wxSTC_TXT2TAGS_HEADER6 11
- #define wxSTC_TXT2TAGS_PRECHAR 12
- #define wxSTC_TXT2TAGS_ULIST_ITEM 13
- #define wxSTC_TXT2TAGS_OLIST_ITEM 14
- #define wxSTC_TXT2TAGS_BLOCKQUOTE 15
- #define wxSTC_TXT2TAGS_STRIKEOUT 16
- #define wxSTC_TXT2TAGS_HRULE 17
- #define wxSTC_TXT2TAGS_LINK 18
- #define wxSTC_TXT2TAGS_CODE 19
- #define wxSTC_TXT2TAGS_CODE2 20
- #define wxSTC_TXT2TAGS_CODEBK 21
- #define wxSTC_TXT2TAGS_COMMENT 22
- #define wxSTC_TXT2TAGS_OPTION 23
- #define wxSTC_TXT2TAGS_PREPROC 24
- #define wxSTC_TXT2TAGS_POSTPROC 25
- #define wxSTC_A68K_DEFAULT 0

  Lexical states for SCLEX_A68K.
- #define wxSTC_A68K_COMMENT 1
- #define wxSTC_A68K_NUMBER_DEC 2
- #define wxSTC_A68K_NUMBER_BIN 3
- #define wxSTC_A68K_NUMBER_HEX 4
- #define wxSTC_A68K_STRING1 5
- #define wxSTC_A68K_OPERATOR 6
- #define wxSTC_A68K_CPUINSTRUCTION 7
- #define wxSTC_A68K_EXTINSTRUCTION 8

- #define wxSTC_A68K_REGISTER 9
- #define wxSTC_A68K_DIRECTIVE 10
- #define wxSTC_A68K_MACRO_ARG 11
- #define wxSTC_A68K_LABEL 12
- #define wxSTC_A68K_STRING2 13
- #define wxSTC_A68K_IDENTIFIER 14
- #define wxSTC_A68K_MACRO_DECLARATION 15
- #define wxSTC_A68K_COMMENT_WORD 16
- #define wxSTC_A68K_COMMENT_SPECIAL 17
- #define wxSTC_A68K_COMMENT_DOXYGEN 18
- #define wxSTC_MODULA_DEFAULT 0

  Lexical states for SCLEX_MODULA.
- #define wxSTC_MODULA_COMMENT 1
- #define wxSTC_MODULA_DOXYCOMM 2
- #define wxSTC_MODULA_DOXYKEY 3
- #define wxSTC_MODULA_KEYWORD 4
- #define wxSTC_MODULA_RESERVED 5
- #define wxSTC_MODULA_NUMBER 6
- #define wxSTC_MODULA_BASENUM 7
- #define wxSTC_MODULA_FLOAT 8
- #define wxSTC_MODULA_STRING 9
- #define wxSTC_MODULA_STRSPEC 10
- #define wxSTC_MODULA_CHAR 11
- #define wxSTC_MODULA_CHARSPEC 12
- #define wxSTC_MODULA_PROC 13
- #define wxSTC_MODULA_PRAGMA 14
- #define wxSTC_MODULA_PRGKEY 15
- #define wxSTC_MODULA_OPERATOR 16
- #define wxSTC_MODULA_BADSTR 17
- #define wxSTC_COFFEESCRIPT_DEFAULT 0

  Lexical states for SCLEX_COFFEESCRIPT.
- #define wxSTC_COFFEESCRIPT_COMMENT 1
- #define wxSTC_COFFEESCRIPT_COMMENTLINE 2
- #define wxSTC_COFFEESCRIPT_COMMENTDOC 3
- #define wxSTC_COFFEESCRIPT_NUMBER 4
- #define wxSTC_COFFEESCRIPT_WORD 5
- #define wxSTC_COFFEESCRIPT_STRING 6
- #define wxSTC_COFFEESCRIPT_CHARACTER 7
- #define wxSTC_COFFEESCRIPT_UUID 8
- #define wxSTC_COFFEESCRIPT_PREPROCESSOR 9
- #define wxSTC_COFFEESCRIPT_OPERATOR 10
- #define wxSTC_COFFEESCRIPT_IDENTIFIER 11
- #define wxSTC_COFFEESCRIPT_STRINGEOL 12
- #define wxSTC_COFFEESCRIPT_VERBATIM 13
- #define wxSTC_COFFEESCRIPT_REGEX 14
- #define wxSTC_COFFEESCRIPT_COMMENTLINEDOC 15
- #define wxSTC_COFFEESCRIPT_WORD2 16
- #define wxSTC_COFFEESCRIPT_COMMENTDOCKEYWORD 17
- #define wxSTC_COFFEESCRIPT_COMMENTDOCKEYWORDERROR 18
- #define wxSTC_COFFEESCRIPT_GLOBALCLASS 19
- #define wxSTC_COFFEESCRIPT_STRINGRAW 20
- #define wxSTC_COFFEESCRIPT_TRIPLEVERBATIM 21
- #define wxSTC_COFFEESCRIPT_COMMENTBLOCK 22
- #define wxSTC_COFFEESCRIPT_VERBOSE_REGEX 23

- #define wxSTC_COFFEESCRIPT_VERBOSE_REGEX_COMMENT 24
- #define wxSTC_AVS_DEFAULT 0

  Lexical states for SCLEX_AVS.
- #define wxSTC_AVS_COMMENTBLOCK 1
- #define wxSTC_AVS_COMMENTBLOCKN 2
- #define wxSTC_AVS_COMMENTLINE 3
- #define wxSTC_AVS_NUMBER 4
- #define wxSTC_AVS_OPERATOR 5
- #define wxSTC_AVS_IDENTIFIER 6
- #define wxSTC_AVS_STRING 7
- #define wxSTC_AVS_TRIPLESTRING 8
- #define wxSTC_AVS_KEYWORD 9
- #define wxSTC_AVS_FILTER 10
- #define wxSTC_AVS_PLUGIN 11
- #define wxSTC_AVS_FUNCTION 12
- #define wxSTC_AVS_CLIPPROP 13
- #define wxSTC_AVS_USERDFN 14
- #define wxSTC_ECL_DEFAULT 0

  Lexical states for SCLEX_ECL.
- #define wxSTC_ECL_COMMENT 1
- #define wxSTC_ECL_COMMENTLINE 2
- #define wxSTC_ECL_NUMBER 3
- #define wxSTC_ECL_STRING 4
- #define wxSTC_ECL_WORD0 5
- #define wxSTC_ECL_OPERATOR 6
- #define wxSTC_ECL_CHARACTER 7
- #define wxSTC_ECL_UUID 8
- #define wxSTC_ECL_PREPROCESSOR 9
- #define wxSTC_ECL_UNKNOWN 10
- #define wxSTC_ECL_IDENTIFIER 11
- #define wxSTC_ECL_STRINGEOL 12
- #define wxSTC_ECL_VERBATIM 13
- #define wxSTC_ECL_REGEX 14
- #define wxSTC_ECL_COMMENTLINEDOC 15
- #define wxSTC_ECL_WORD1 16
- #define wxSTC_ECL_COMMENTDOCKEYWORD 17
- #define wxSTC_ECL_COMMENTDOCKEYWORDERROR 18
- #define wxSTC_ECL_WORD2 19
- #define wxSTC_ECL_WORD3 20
- #define wxSTC_ECL_WORD4 21
- #define wxSTC_ECL_WORD5 22
- #define wxSTC_ECL_COMMENTDOC 23
- #define wxSTC_ECL_ADDED 24
- #define wxSTC_ECL_DELETED 25
- #define wxSTC_ECL_CHANGED 26
- #define wxSTC_ECL_MOVED 27
- #define wxSTC_OSCRIPT_DEFAULT 0

  Lexical states for SCLEX_OSCRIPT.
- #define wxSTC_OSCRIPT_LINE_COMMENT 1
- #define wxSTC_OSCRIPT_BLOCK_COMMENT 2
- #define wxSTC_OSCRIPT_DOC_COMMENT 3
- #define wxSTC_OSCRIPT_PREPROCESSOR 4
- #define wxSTC_OSCRIPT_NUMBER 5
- #define wxSTC_OSCRIPT_SINGLEQUOTE_STRING 6

- #define wxSTC_OSCRIPT_DOUBLEQUOTE_STRING 7
- #define wxSTC_OSCRIPT_CONSTANT 8
- #define wxSTC_OSCRIPT_IDENTIFIER 9
- #define wxSTC_OSCRIPT_GLOBAL 10
- #define wxSTC_OSCRIPT_KEYWORD 11
- #define wxSTC_OSCRIPT_OPERATOR 12
- #define wxSTC_OSCRIPT_LABEL 13
- #define wxSTC_OSCRIPT_TYPE 14
- #define wxSTC_OSCRIPT_FUNCTION 15
- #define wxSTC_OSCRIPT_OBJECT 16
- #define wxSTC_OSCRIPT_PROPERTY 17
- #define wxSTC_OSCRIPT_METHOD 18
- #define wxSTC_VISUALPROLOG_DEFAULT 0

  Lexical states for SCLEX_VISUALPROLOG.
- #define wxSTC_VISUALPROLOG_KEY_MAJOR 1
- #define wxSTC_VISUALPROLOG_KEY_MINOR 2
- #define wxSTC_VISUALPROLOG_KEY_DIRECTIVE 3
- #define wxSTC_VISUALPROLOG_COMMENT_BLOCK 4
- #define wxSTC_VISUALPROLOG_COMMENT_LINE 5
- #define wxSTC_VISUALPROLOG_COMMENT_KEY 6
- #define wxSTC_VISUALPROLOG_COMMENT_KEY_ERROR 7
- #define wxSTC_VISUALPROLOG_IDENTIFIER 8
- #define wxSTC_VISUALPROLOG_VARIABLE 9
- #define wxSTC_VISUALPROLOG_ANONYMOUS 10
- #define wxSTC_VISUALPROLOG_NUMBER 11
- #define wxSTC_VISUALPROLOG_OPERATOR 12
- #define wxSTC_VISUALPROLOG_CHARACTER 13
- #define wxSTC_VISUALPROLOG_CHARACTER_TOO_MANY 14
- #define wxSTC_VISUALPROLOG_CHARACTER_ESCAPE_ERROR 15
- #define wxSTC_VISUALPROLOG_STRING 16
- #define wxSTC_VISUALPROLOG_STRING_ESCAPE 17
- #define wxSTC_VISUALPROLOG_STRING_ESCAPE_ERROR 18
- #define wxSTC_VISUALPROLOG_STRING_EOL_OPEN 19
- #define wxSTC_VISUALPROLOG_STRING_VERBATIM 20
- #define wxSTC_VISUALPROLOG_STRING_VERBATIM_SPECIAL 21
- #define wxSTC_VISUALPROLOG_STRING_VERBATIM_EOL 22
- #define wxSTC_STTXT_DEFAULT 0

  Lexical states for SCLEX_STTXT.
- #define wxSTC_STTXT_COMMENT 1
- #define wxSTC_STTXT_COMMENTLINE 2
- #define wxSTC_STTXT_KEYWORD 3
- #define wxSTC_STTXT_TYPE 4
- #define wxSTC_STTXT_FUNCTION 5
- #define wxSTC_STTXT_FB 6
- #define wxSTC_STTXT_NUMBER 7
- #define wxSTC_STTXT_HEXNUMBER 8
- #define wxSTC_STTXT_PRAGMA 9
- #define wxSTC_STTXT_OPERATOR 10
- #define wxSTC_STTXT_CHARACTER 11
- #define wxSTC_STTXT_STRING1 12
- #define wxSTC_STTXT_STRING2 13
- #define wxSTC_STTXT_STRINGEOL 14
- #define wxSTC_STTXT_IDENTIFIER 15
- #define wxSTC_STTXT_DATETIME 16

- #define wxSTC_STTXT_VARS 17
- #define wxSTC_STTXT_PRAGMAS 18
- #define wxSTC_KVIRC_DEFAULT 0

  Lexical states for SCLEX_KVIRC.
- #define wxSTC_KVIRC_COMMENT 1
- #define wxSTC_KVIRC_COMMENTBLOCK 2
- #define wxSTC_KVIRC_STRING 3
- #define wxSTC_KVIRC_WORD 4
- #define wxSTC_KVIRC_KEYWORD 5
- #define wxSTC_KVIRC_FUNCTION_KEYWORD 6
- #define wxSTC_KVIRC_FUNCTION 7
- #define wxSTC_KVIRC_VARIABLE 8
- #define wxSTC_KVIRC_NUMBER 9
- #define wxSTC_KVIRC_OPERATOR 10
- #define wxSTC_KVIRC_STRING_FUNCTION 11
- #define wxSTC_KVIRC_STRING_VARIABLE 12
- #define wxSTC_RUST_DEFAULT 0

  Lexical states for SCLEX_RUST.
- #define wxSTC_RUST_COMMENTBLOCK 1
- #define wxSTC_RUST_COMMENTLINE 2
- #define wxSTC_RUST_COMMENTBLOCKDOC 3
- #define wxSTC_RUST_COMMENTLINEDOC 4
- #define wxSTC_RUST_NUMBER 5
- #define wxSTC_RUST_WORD 6
- #define wxSTC_RUST_WORD2 7
- #define wxSTC_RUST_WORD3 8
- #define wxSTC_RUST_WORD4 9
- #define wxSTC_RUST_WORD5 10
- #define wxSTC_RUST_WORD6 11
- #define wxSTC_RUST_WORD7 12
- #define wxSTC_RUST_STRING 13
- #define wxSTC_RUST_STRINGR 14
- #define wxSTC_RUST_CHARACTER 15
- #define wxSTC_RUST_OPERATOR 16
- #define wxSTC_RUST_IDENTIFIER 17
- #define wxSTC_RUST_LIFETIME 18
- #define wxSTC_RUST_MACRO 19
- #define wxSTC_RUST_LEXERROR 20
- #define wxSTC_DMAP_DEFAULT 0

  Lexical states for SCLEX_DMAP.
- #define wxSTC_DMAP_COMMENT 1
- #define wxSTC_DMAP_NUMBER 2
- #define wxSTC_DMAP_STRING1 3
- #define wxSTC_DMAP_STRING2 4
- #define wxSTC_DMAP_STRINGEOL 5
- #define wxSTC_DMAP_OPERATOR 6
- #define wxSTC_DMAP_IDENTIFIER 7
- #define wxSTC_DMAP_WORD 8
- #define wxSTC_DMAP_WORD2 9
- #define wxSTC_DMAP_WORD3 10
- #define wxSTC_CMD_REDO 2011

  Redoes the next action on the undo history.
- #define wxSTC_CMD_SELECTALL 2013

  Select all the text in the document.

- #define wxSTC_CMD_UNDO 2176

  Undo one action in the undo history.
- #define wxSTC_CMD_CUT 2177

  Cut the selection to the clipboard.
- #define wxSTC_CMD_COPY 2178

  Copy the selection to the clipboard.
- #define wxSTC_CMD_PASTE 2179

  Paste the contents of the clipboard into the document replacing the selection.
- #define wxSTC_CMD_CLEAR 2180

  Clear the selection.
- #define wxSTC_CMD_LINEDOWN 2300

  Move caret down one line.
- #define wxSTC_CMD_LINEDOWNEXTEND 2301

  Move caret down one line extending selection to new caret position.
- #define wxSTC_CMD_LINEUP 2302

  Move caret up one line.
- #define wxSTC_CMD_LINEUPEXTEND 2303

  Move caret up one line extending selection to new caret position.
- #define wxSTC_CMD_CHARLEFT 2304

  Move caret left one character.
- #define wxSTC_CMD_CHARLEFTEXTEND 2305

  Move caret left one character extending selection to new caret position.
- #define wxSTC_CMD_CHARRIGHT 2306

  Move caret right one character.
- #define wxSTC_CMD_CHARRIGHTEXTEND 2307

  Move caret right one character extending selection to new caret position.
- #define wxSTC_CMD_WORDLEFT 2308

  Move caret left one word.
- #define wxSTC_CMD_WORDLEFTEXTEND 2309

  Move caret left one word extending selection to new caret position.
- #define wxSTC_CMD_WORDRIGHT 2310

  Move caret right one word.
- #define wxSTC_CMD_WORDRIGHTEXTEND 2311

  Move caret right one word extending selection to new caret position.
- #define wxSTC_CMD_HOME 2312

  Move caret to first position on line.
- #define wxSTC_CMD_HOMEEXTEND 2313

  Move caret to first position on line extending selection to new caret position.
- #define wxSTC_CMD_LINEEND 2314

  Move caret to last position on line.
- #define wxSTC_CMD_LINEENDEXTEND 2315

  Move caret to last position on line extending selection to new caret position.
- #define wxSTC_CMD_DOCUMENTSTART 2316

  Move caret to first position in document.
- #define wxSTC_CMD_DOCUMENTSTARTEXTEND 2317

  Move caret to first position in document extending selection to new caret position.
- #define wxSTC_CMD_DOCUMENTEND 2318

  Move caret to last position in document.
- #define wxSTC_CMD_DOCUMENTENDEXTEND 2319

  Move caret to last position in document extending selection to new caret position.
- #define wxSTC_CMD_PAGEUP 2320

Move caret one page up.

- #define wxSTC_CMD_PAGEUPEXTEND 2321

  Move caret one page up extending selection to new caret position.
- #define wxSTC_CMD_PAGEDOWN 2322

  Move caret one page down.
- #define wxSTC_CMD_PAGEDOWNEXTEND 2323

  Move caret one page down extending selection to new caret position.
- #define wxSTC_CMD_EDITTOGGLEOVERTYPE 2324

  Switch from insert to overtype mode or the reverse.
- #define wxSTC_CMD_CANCEL 2325

  Cancel any modes such as call tip or auto-completion list display.
- #define wxSTC_CMD_DELETEBACK 2326

  Delete the selection or if no selection, the character before the caret.
- #define wxSTC_CMD_TAB 2327

  If selection is empty or all on one line replace the selection with a tab character.
- #define wxSTC_CMD_BACKTAB 2328

  Dedent the selected lines.
- #define wxSTC_CMD_NEWLINE 2329

  Insert a new line, may use a CRLF, CR or LF depending on EOL mode.
- #define wxSTC_CMD_FORMFEED 2330

  Insert a Form Feed character.
- #define wxSTC_CMD_VCHOME 2331

  Move caret to before first visible character on line.
- #define wxSTC_CMD_VCHOMEEXTEND 2332

  Like VCHome but extending selection to new caret position.
- #define wxSTC_CMD_ZOOMIN 2333

  Magnify the displayed text by increasing the sizes by 1 point.
- #define wxSTC_CMD_ZOOMOUT 2334

  Make the displayed text smaller by decreasing the sizes by 1 point.
- #define wxSTC_CMD_DELWORDLEFT 2335

  Delete the word to the left of the caret.
- #define wxSTC_CMD_DELWORDRIGHT 2336

  Delete the word to the right of the caret.
- #define wxSTC_CMD_DELWORDRIGHTEND 2518

  Delete the word to the right of the caret, but not the trailing non-word characters.
- #define wxSTC_CMD_LINECUT 2337

  Cut the line containing the caret.
- #define wxSTC_CMD_LINEDELETE 2338

  Delete the line containing the caret.
- #define wxSTC_CMD_LINETRANSPOSE 2339

  Switch the current line with the previous.
- #define wxSTC_CMD_LINEDUPLICATE 2404

  Duplicate the current line.
- #define wxSTC_CMD_LOWERCASE 2340

  Transform the selection to lower case.
- #define wxSTC_CMD_UPPERCASE 2341

  Transform the selection to upper case.
- #define wxSTC_CMD_LINESCROLLDOWN 2342

  Scroll the document down, keeping the caret visible.
- #define wxSTC_CMD_LINESCROLLUP 2343

  Scroll the document up, keeping the caret visible.

- #define wxSTC_CMD_DELETEBACKNOTLINE 2344

  Delete the selection or if no selection, the character before the caret.
- #define wxSTC_CMD_HOMEDISPLAY 2345

  Move caret to first position on display line.
- #define wxSTC_CMD_HOMEDISPLAYEXTEND 2346

  Move caret to first position on display line extending selection to new caret position.
- #define wxSTC_CMD_LINEENDDISPLAY 2347

  Move caret to last position on display line.
- #define wxSTC_CMD_LINEENDDISPLAYEXTEND 2348

  Move caret to last position on display line extending selection to new caret position.
- #define wxSTC_CMD_HOMEWRAP 2349

  These are like their namesakes Home(Extend)?, LineEnd(Extend)?, VCHome(Extend)? except they behave differently when word-wrap is enabled: They go first to the start / end of the display line, like (Home|LineEnd)Display The difference is that, the cursor is already at the point, it goes on to the start or end of the document line, as appropriate for (Home|LineEnd|VCHome)(Extend)?.
- #define wxSTC_CMD_HOMEWRAPEXTEND 2450
- #define wxSTC_CMD_LINEENDWRAP 2451
- #define wxSTC_CMD_LINEENDWRAPEXTEND 2452
- #define wxSTC_CMD_VCHOMEWRAP 2453
- #define wxSTC_CMD_VCHOMEWRAPEXTEND 2454
- #define wxSTC_CMD_LINECOPY 2455

  Copy the line containing the caret.
- #define wxSTC_CMD_WORDPARTLEFT 2390

  Move to the previous change in capitalisation.
- #define wxSTC_CMD_WORDPARTLEFTEXTEND 2391

  Move to the previous change in capitalisation extending selection to new caret position.
- #define wxSTC_CMD_WORDPARTRIGHT 2392

  Move to the change next in capitalisation.
- #define wxSTC_CMD_WORDPARTRIGHTEXTEND 2393

  Move to the next change in capitalisation extending selection to new caret position.
- #define wxSTC_CMD_DELLINELEFT 2395

  Delete back from the current position to the start of the line.
- #define wxSTC_CMD_DELLINERIGHT 2396

  Delete forwards from the current position to the end of the line.
- #define wxSTC_CMD_PARADOWN 2413

  Move caret between paragraphs (delimited by empty lines).
- #define wxSTC_CMD_PARADOWNEXTEND 2414
- #define wxSTC_CMD_PARAUP 2415
- #define wxSTC_CMD_PARAUPEXTEND 2416
- #define wxSTC_CMD_LINEDOWNRECTEXTEND 2426

  Move caret down one line, extending rectangular selection to new caret position.
- #define wxSTC_CMD_LINEUPRECTEXTEND 2427

  Move caret up one line, extending rectangular selection to new caret position.
- #define wxSTC_CMD_CHARLEFTRECTEXTEND 2428

  Move caret left one character, extending rectangular selection to new caret position.
- #define wxSTC_CMD_CHARRIGHTRECTEXTEND 2429

  Move caret right one character, extending rectangular selection to new caret position.
- #define wxSTC_CMD_HOMERECTEXTEND 2430

  Move caret to first position on line, extending rectangular selection to new caret position.
- #define wxSTC_CMD_VCHOMERECTEXTEND 2431

  Move caret to before first visible character on line.
- #define wxSTC_CMD_LINEENDRECTEXTEND 2432

Move caret to last position on line, extending rectangular selection to new caret position.

- #define wxSTC_CMD_PAGEUPRECTEXTEND 2433

  Move caret one page up, extending rectangular selection to new caret position.
- #define wxSTC_CMD_PAGEDOWNRECTEXTEND 2434

  Move caret one page down, extending rectangular selection to new caret position.
- #define wxSTC_CMD_STUTTEREDPAGEUP 2435

  Move caret to top of page, or one page up if already at top of page.
- #define wxSTC_CMD_STUTTEREDPAGEUPEXTEND 2436

  Move caret to top of page, or one page up if already at top of page, extending selection to new caret position.
- #define wxSTC_CMD_STUTTEREDPAGEDOWN 2437

  Move caret to bottom of page, or one page down if already at bottom of page.
- #define wxSTC_CMD_STUTTEREDPAGEDOWNEXTEND 2438

  Move caret to bottom of page, or one page down if already at bottom of page, extending selection to new caret position.
- #define wxSTC_CMD_WORDLEFTEND 2439

  Move caret left one word, position cursor at end of word.
- #define wxSTC_CMD_WORDLEFTENDEXTEND 2440

  Move caret left one word, position cursor at end of word, extending selection to new caret position.
- #define wxSTC_CMD_WORDRIGHTEND 2441

  Move caret right one word, position cursor at end of word.
- #define wxSTC_CMD_WORDRIGHTENDEXTEND 2442

  Move caret right one word, position cursor at end of word, extending selection to new caret position.
- #define wxSTC_CMD_VERTICALCENTRECARET 2619

  Centre current line in window.
- #define wxSTC_CMD_MOVESELECTEDLINESUP 2620

  Move the selected lines up one line, shifting the line above after the selection.
- #define wxSTC_CMD_MOVESELECTEDLINESDOWN 2621

  Move the selected lines down one line, shifting the line below before the selection.
- #define wxSTC_CMD_SCROLLTOSTART 2628

  Scroll to start of document.
- #define wxSTC_CMD_SCROLLTOEND 2629

  Scroll to end of document.
- #define wxSTC_CMD_VCHOMEDISPLAY 2652

  Move caret to before first visible character on display line.
- #define wxSTC_CMD_VCHOMEDISPLAYEXTEND 2653

  Like VCHomeDisplay but extending selection to new caret position.

Variables

- const wxEventType wxEVT_STC_CHANGE
- const wxEventType wxEVT_STC_STYLENEEDED
- const wxEventType wxEVT_STC_CHARADDED
- const wxEventType wxEVT_STC_SAVEPOINTREACHED
- const wxEventType wxEVT_STC_SAVEPOINTLEFT
- const wxEventType wxEVT_STC_ROMODIFYATTEMPT
- const wxEventType wxEVT_STC_KEY
- const wxEventType wxEVT_STC_DOUBLECLICK
- const wxEventType wxEVT_STC_UPDATEUI
- const wxEventType wxEVT_STC_MODIFIED
- const wxEventType wxEVT_STC_MACRORECORD
- const wxEventType wxEVT_STC_MARGINCLICK

- const wxEventType wxEVT_STC_NEEDSHOWN
- const wxEventType wxEVT_STC_PAINTED
- const wxEventType wxEVT_STC_USERLISTSELECTION
- const wxEventType wxEVT_STC_URIDROPPED
- const wxEventType wxEVT_STC_DWELLSTART
- const wxEventType wxEVT_STC_DWELLEND
- const wxEventType wxEVT_STC_START_DRAG
- const wxEventType wxEVT_STC_DRAG_OVER
- const wxEventType wxEVT_STC_DO_DROP
- const wxEventType wxEVT_STC_ZOOM
- const wxEventType wxEVT_STC_HOTSPOT_CLICK
- const wxEventType wxEVT_STC_HOTSPOT_DCLICK
- const wxEventType wxEVT_STC_CALLTIP_CLICK
- const wxEventType wxEVT_STC_AUTOCOMP_SELECTION
- const wxEventType wxEVT_STC_INDICATOR_CLICK
- const wxEventType wxEVT_STC_INDICATOR_RELEASE
- const wxEventType wxEVT_STC_AUTOCOMP_CANCELLED
- const wxEventType wxEVT_STC_AUTOCOMP_CHAR_DELETED
- const wxEventType wxEVT_STC_HOTSPOT_RELEASE_CLICK
- const wxEventType wxEVT_STC_CLIPBOARD_COPY
- const wxEventType wxEVT_STC_CLIPBOARD_PASTE

22.449.1 Macro Definition Documentation

#define wxSTC_4GL_BLOCK 8

#define wxSTC_4GL_BLOCK_ 24

#define wxSTC_4GL_CHARACTER 4

#define wxSTC_4GL_CHARACTER_ 20

#define wxSTC_4GL_COMMENT1 10

#define wxSTC_4GL_COMMENT1_ 26

#define wxSTC_4GL_COMMENT2 11

#define wxSTC_4GL_COMMENT2_ 27

#define wxSTC_4GL_COMMENT3 12

#define wxSTC_4GL_COMMENT3_ 28

#define wxSTC_4GL_COMMENT4 13

#define wxSTC_4GL_COMMENT4_ 29

#define wxSTC_4GL_COMMENT5 14

#define wxSTC_4GL_COMMENT5_ 30

#define wxSTC_4GL_COMMENT6 15

#define wxSTC_4GL_COMMENT6_ 31

#define wxSTC_4GL_DEFAULT 0

Lexical state for SCLEX_PROGRESS.

#define wxSTC_4GL_DEFAULT_ 16

#define wxSTC_4GL_END 9

#define wxSTC_4GL_END_ 25

#define wxSTC_4GL_IDENTIFIER 7

#define wxSTC_4GL_IDENTIFIER_ 23

#define wxSTC_4GL_NUMBER 1

#define wxSTC_4GL_NUMBER_ 17

#define wxSTC_4GL_OPERATOR 6

#define wxSTC_4GL_OPERATOR_ 22

#define wxSTC_4GL_PREPROCESSOR 5

#define wxSTC_4GL_PREPROCESSOR_ 21

#define wxSTC_4GL_STRING 3

#define wxSTC_4GL_STRING_ 19

#define wxSTC_4GL_WORD 2

#define wxSTC_4GL_WORD_ 18

#define wxSTC_A68K_COMMENT 1

#define wxSTC_A68K_COMMENT_DOXYGEN 18

#define wxSTC_A68K_COMMENT_SPECIAL 17

#define wxSTC_A68K_COMMENT_WORD 16

#define wxSTC_A68K_CPUINSTRUCTION 7

#define wxSTC_A68K_DEFAULT 0

Lexical states for SCLEX_A68K.

#define wxSTC_A68K_DIRECTIVE 10

#define wxSTC_A68K_EXTINSTRUCTION 8

#define wxSTC_A68K_IDENTIFIER 14

#define wxSTC_A68K_LABEL 12

#define wxSTC_A68K_MACRO_ARG 11

#define wxSTC_A68K_MACRO_DECLARATION 15

#define wxSTC_A68K_NUMBER_BIN 3

#define wxSTC_A68K_NUMBER_DEC 2

#define wxSTC_A68K_NUMBER_HEX 4

#define wxSTC_A68K_OPERATOR 6

#define wxSTC_A68K_REGISTER 9

#define wxSTC_A68K_STRING1 5

#define wxSTC_A68K_STRING2 13

#define wxSTC_ABAQUS_ARGUMENT 11

#define wxSTC_ABAQUS_COMMAND 8

#define wxSTC_ABAQUS_COMMENT 1

#define wxSTC_ABAQUS_COMMENTBLOCK 2

#define wxSTC_ABAQUS_DEFAULT 0

Lexical states for SCLEX_ABAQUS.

#define wxSTC_ABAQUS_FUNCTION 12

#define wxSTC_ABAQUS_NUMBER 3

#define wxSTC_ABAQUS_OPERATOR 5

#define wxSTC_ABAQUS_PROCESSOR 7

#define wxSTC_ABAQUS_SLASHCOMMAND 9

#define wxSTC_ABAQUS_STARCOMMAND 10

#define wxSTC_ABAQUS_STRING 4

#define wxSTC_ABAQUS_WORD 6

#define wxSTC_ADA_CHARACTER 5

#define wxSTC_ADA_CHARACTEREOL 6

#define wxSTC_ADA_COMMENTLINE 10

#define wxSTC_ADA_DEFAULT 0

Lexical states for SCLEX_ADA.

#define wxSTC_ADA_DELIMITER 4

#define wxSTC_ADA_IDENTIFIER 2

#define wxSTC_ADA_ILLEGAL 11

#define wxSTC_ADA_LABEL 9

#define wxSTC_ADA_NUMBER 3

#define wxSTC_ADA_STRING 7

#define wxSTC_ADA_STRINGEOL 8

#define wxSTC_ADA_WORD 1

#define wxSTC_ALPHA_NOALPHA 256

#define wxSTC_ALPHA_OPAQUE 255

#define wxSTC_ALPHA_TRANSPARENT 0

#define wxSTC_ANNOTATION_BOXED 2

#define wxSTC_ANNOTATION_HIDDEN 0

#define wxSTC_ANNOTATION_STANDARD 1

#define wxSTC_APDL_ARGUMENT 11

#define wxSTC_APDL_COMMAND 8

#define wxSTC_APDL_COMMENT 1

#define wxSTC_APDL_COMMENTBLOCK 2

#define wxSTC_APDL_DEFAULT 0

Lexical states for SCLEX_APDL.

#define wxSTC_APDL_FUNCTION 12

#define wxSTC_APDL_NUMBER 3

#define wxSTC_APDL_OPERATOR 5

#define wxSTC_APDL_PROCESSOR 7

#define wxSTC_APDL_SLASHCOMMAND 9

#define wxSTC_APDL_STARCOMMAND 10

#define wxSTC_APDL_STRING 4

#define wxSTC_APDL_WORD 6

#define wxSTC_ASM_CHARACTER 12

#define wxSTC_ASM_COMMENT 1

#define wxSTC_ASM_COMMENTBLOCK 11

#define wxSTC_ASM_COMMENTDIRECTIVE 15

#define wxSTC_ASM_CPUINSTRUCTION 6

#define wxSTC_ASM_DEFAULT 0

Lexical states for SCLEX_ASM, SCLEX_AS.

#define wxSTC_ASM_DIRECTIVE 9

#define wxSTC_ASM_DIRECTIVEOPERAND 10

#define wxSTC_ASM_EXTINSTRUCTION 14

#define wxSTC_ASM_IDENTIFIER 5

#define wxSTC_ASM_MATHINSTRUCTION 7

#define wxSTC_ASM_NUMBER 2

#define wxSTC_ASM_OPERATOR 4

#define wxSTC_ASM_REGISTER 8

#define wxSTC_ASM_STRING 3

#define wxSTC_ASM_STRINGEOL 13

#define wxSTC_ASN1_ATTRIBUTE 7

#define wxSTC_ASN1_COMMENT 1

#define wxSTC_ASN1_DEFAULT 0

Lexical states for SCLEX_ASN1.

#define wxSTC_ASN1_DESCRIPTOR 8

#define wxSTC_ASN1_IDENTIFIER 2

#define wxSTC_ASN1_KEYWORD 6

#define wxSTC_ASN1_OID 4

#define wxSTC_ASN1_OPERATOR 10

#define wxSTC_ASN1_SCALAR 5

#define wxSTC_ASN1_STRING 3

#define wxSTC_ASN1_TYPE 9

#define wxSTC_ASY_CHARACTER 6

#define wxSTC_ASY_COMMENT 1

#define wxSTC_ASY_COMMENTLINE 2

#define wxSTC_ASY_COMMENTLINEDOC 10

#define wxSTC_ASY_DEFAULT 0

Lexical states for SCLEX_ASYMPTOTE.

#define wxSTC_ASY_IDENTIFIER 8

#define wxSTC_ASY_NUMBER 3

#define wxSTC_ASY_OPERATOR 7

#define wxSTC_ASY_STRING 5

#define wxSTC_ASY_STRINGEOL 9

#define wxSTC_ASY_WORD 4

#define wxSTC_ASY_WORD2 11

#define wxSTC_AU3_COMMENT 1

#define wxSTC_AU3_COMMENTBLOCK 2

#define wxSTC_AU3_COMOBJ 14

#define wxSTC_AU3_DEFAULT 0

Lexical states for SCLEX_AU3.

#define wxSTC_AU3_EXPAND 13

#define wxSTC_AU3_FUNCTION 4

#define wxSTC_AU3_KEYWORD 5

#define wxSTC_AU3_MACRO 6

#define wxSTC_AU3_NUMBER 3

#define wxSTC_AU3_OPERATOR 8

#define wxSTC_AU3_PREPROCESSOR 11

#define wxSTC_AU3_SENT 10

#define wxSTC_AU3_SPECIAL 12

#define wxSTC_AU3_STRING 7

#define wxSTC_AU3_UDF 15

#define wxSTC_AU3_VARIABLE 9

#define wxSTC_AUTOMATICFOLD_CHANGE 0x0004

#define wxSTC_AUTOMATICFOLD_CLICK 0x0002

#define wxSTC_AUTOMATICFOLD_SHOW 0x0001

#define wxSTC_AVE_COMMENT 1

#define wxSTC_AVE_DEFAULT 0

Lexical states for SCLEX_AVE, Avenue.

#define wxSTC_AVE_ENUM 7

#define wxSTC_AVE_IDENTIFIER 9

#define wxSTC_AVE_NUMBER 2

#define wxSTC_AVE_OPERATOR 10

#define wxSTC_AVE_STRING 6

#define wxSTC_AVE_STRINGEOL 8

#define wxSTC_AVE_WORD 3

#define wxSTC_AVE_WORD1 11

#define wxSTC_AVE_WORD2 12

#define wxSTC_AVE_WORD3 13

#define wxSTC_AVE_WORD4 14

#define wxSTC_AVE_WORD5 15

#define wxSTC_AVE_WORD6 16

#define wxSTC_AVS_CLIPPROP 13

#define wxSTC_AVS_COMMENTBLOCK 1

#define wxSTC_AVS_COMMENTBLOCKN 2

#define wxSTC_AVS_COMMENTLINE 3

#define wxSTC_AVS_DEFAULT 0

Lexical states for SCLEX_AVS.

#define wxSTC_AVS_FILTER 10

#define wxSTC_AVS_FUNCTION 12

#define wxSTC_AVS_IDENTIFIER 6

#define wxSTC_AVS_KEYWORD 9

#define wxSTC_AVS_NUMBER 4

#define wxSTC_AVS_OPERATOR 5

#define wxSTC_AVS_PLUGIN 11

#define wxSTC_AVS_STRING 7

#define wxSTC_AVS_TRIPLESTRING 8

#define wxSTC_AVS_USERDFN 14

#define wxSTC_B_ASM 14

#define wxSTC_B_BINNUMBER 18

#define wxSTC_B_COMMENT 1

#define wxSTC_B_COMMENTBLOCK 19

#define wxSTC_B_CONSTANT 13

#define wxSTC_B_DATE 8

#define wxSTC_B_DEFAULT 0

Lexical states for SCLEX_VB, SCLEX_VBSCRIPT, SCLEX_POWERBASIC.

#define wxSTC_B_DOCBLOCK 21

#define wxSTC_B_DOCKEYWORD 22

#define wxSTC_B_DOCLINE 20

#define wxSTC_B_ERROR 16

#define wxSTC_B_HEXNUMBER 17

#define wxSTC_B_IDENTIFIER 7

#define wxSTC_B_KEYWORD 3

#define wxSTC_B_KEYWORD2 10

#define wxSTC_B_KEYWORD3 11

#define wxSTC_B_KEYWORD4 12

#define wxSTC_B_LABEL 15

#define wxSTC_B_NUMBER 2

#define wxSTC_B_OPERATOR 6

#define wxSTC_B_PREPROCESSOR 5

#define wxSTC_B_STRING 4

#define wxSTC_B_STRINGEOL 9

#define wxSTC_BAAN_COMMENT 1

#define wxSTC_BAAN_COMMENTDOC 2

#define wxSTC_BAAN_DEFAULT 0

Lexical states for SCLEX_BAAN.

#define wxSTC_BAAN_IDENTIFIER 8

#define wxSTC_BAAN_NUMBER 3

#define wxSTC_BAAN_OPERATOR 7

#define wxSTC_BAAN_PREPROCESSOR 6

#define wxSTC_BAAN_STRING 5

#define wxSTC_BAAN_STRINGEOL 9

#define wxSTC_BAAN_WORD 4

#define wxSTC_BAAN_WORD2 10

#define wxSTC_BAT_COMMAND 5

#define wxSTC_BAT_COMMENT 1

#define wxSTC_BAT_DEFAULT 0

Lexical states for SCLEX_BATCH.

#define wxSTC_BAT_HIDE 4

#define wxSTC_BAT_IDENTIFIER 6

#define wxSTC_BAT_LABEL 3

#define wxSTC_BAT_OPERATOR 7

#define wxSTC_BAT_WORD 2

#define wxSTC_C_CHARACTER 7

#define wxSTC_C_COMMENT 1

#define wxSTC_C_COMMENTDOC 3

#define wxSTC_C_COMMENTDOCKEYWORD 17

#define wxSTC_C_COMMENTDOCKEYWORDERROR 18

#define wxSTC_C_COMMENTLINE 2

#define wxSTC_C_COMMENTLINEDOC 15

#define wxSTC_C_DEFAULT 0

Lexical states for SCLEX_CPP.

#define wxSTC_C_GLOBALCLASS 19

#define wxSTC_C_HASHQUOTEDSTRING 22

#define wxSTC_C_IDENTIFIER 11

#define wxSTC_C_NUMBER 4

#define wxSTC_C_OPERATOR 10

#define wxSTC_C_PREPROCESSOR 9

#define wxSTC_C_PREPROCESSORCOMMENT 23

#define wxSTC_C_PREPROCESSORCOMMENTDOC 24

#define wxSTC_C_REGEX 14

#define wxSTC_C_STRING 6

#define wxSTC_C_STRINGEOL 12

#define wxSTC_C_STRINGRAW 20

#define wxSTC_C_TRIPLEVERBATIM 21

#define wxSTC_C_USERLITERAL 25

#define wxSTC_C_UUID 8

#define wxSTC_C_VERBATIM 13

#define wxSTC_C_WORD 5

#define wxSTC_C_WORD2 16

#define wxSTC_CACHE_CARET 1

#define wxSTC_CACHE_DOCUMENT 3

#define wxSTC_CACHE_NONE 0

#define wxSTC_CACHE_PAGE 2

#define wxSTC_CAML_CHAR 9

#define wxSTC_CAML_COMMENT 12

#define wxSTC_CAML_COMMENT1 13

#define wxSTC_CAML_COMMENT2 14

#define wxSTC_CAML_COMMENT3 15

#define wxSTC_CAML_DEFAULT 0

Lexical states for SCLEX_CAML.

#define wxSTC_CAML_IDENTIFIER 1

#define wxSTC_CAML_KEYWORD 3

#define wxSTC_CAML_KEYWORD2 4

#define wxSTC_CAML_KEYWORD3 5

#define wxSTC_CAML_LINENUM 6

#define wxSTC_CAML_NUMBER 8

#define wxSTC_CAML_OPERATOR 7

#define wxSTC_CAML_STRING 11

#define wxSTC_CAML_TAGNAME 2

#define wxSTC_CAML_WHITE 10

#define wxSTC_CARET_EVEN 0x08

If CARET_EVEN is not set, instead of having symmetrical UZs, the left and bottom UZs are extended up to right and top UZs respectively.

This way, we favour the displaying of useful information: the begining of lines, where most code reside, and the lines after the caret, eg. the body of a function.

#define wxSTC_CARET_JUMPS 0x10

If CARET_JUMPS is set, the display is moved more energetically so the caret can move in the same direction longer before the policy is applied again.

#define wxSTC_CARET_SLOP 0x01

Caret policy, used by SetXCaretPolicy and SetYCaretPolicy.

If CARET_SLOP is set, we can define a slop value: caretSlop. This value defines an unwanted zone (UZ) where the caret is... unwanted. This zone is defined as a number of pixels near the vertical margins, and as a number of lines near the horizontal margins. By keeping the caret away from the edges, it is seen within its context, so it is likely that the identifier that the caret is on can be completely seen, and that the current line is seen with some of the lines following it which are often dependent on that line.

#define wxSTC_CARET_STRICT 0x04

If CARET_STRICT is set, the policy is enforced...

strictly. The caret is centred on the display if slop is not set, and cannot go in the UZ if slop is set.

#define wxSTC_CARETSTICKY_OFF 0

#define wxSTC_CARETSTICKY_ON 1

#define wxSTC_CARETSTICKY_WHITESPACE 2

#define wxSTC_CARETSTYLE_BLOCK 2

#define wxSTC_CARETSTYLE_INVISIBLE 0

#define wxSTC_CARETSTYLE_LINE 1

#define wxSTC_CASE_LOWER 2

#define wxSTC_CASE_MIXED 0

#define wxSTC_CASE_UPPER 1

#define wxSTC_CASEINSENSITIVEBEHAVIOUR_IGNORECASE 1

#define wxSTC_CASEINSENSITIVEBEHAVIOUR_RESPECTCASE 0

#define wxSTC_CHARSET_8859_15 1000

#define wxSTC_CHARSET_ANSI 0

Character set identifiers are used in StyleSetCharacterSet.

The values are the same as the Windows $*$_CHARSET values.

#define wxSTC_CHARSET_ARABIC 178

#define wxSTC_CHARSET_BALTIC 186

#define wxSTC_CHARSET_CHINESEBIG5 136

#define wxSTC_CHARSET_CYRILLIC 1251

#define wxSTC_CHARSET_DEFAULT 1

#define wxSTC_CHARSET_EASTEUROPE 238

#define wxSTC_CHARSET_GB2312 134

#define wxSTC_CHARSET_GREEK 161

#define wxSTC_CHARSET_HANGUL 129

#define wxSTC_CHARSET_HEBREW 177

#define wxSTC_CHARSET_JOHAB 130

#define wxSTC_CHARSET_MAC 77

#define wxSTC_CHARSET_OEM 255

#define wxSTC_CHARSET_RUSSIAN 204

#define wxSTC_CHARSET_SHIFTJIS 128

#define wxSTC_CHARSET_SYMBOL 2

#define wxSTC_CHARSET_THAI 222

#define wxSTC_CHARSET_TURKISH 162

#define wxSTC_CHARSET_VIETNAMESE 163

#define wxSTC_CLW_ATTRIBUTE 13

#define wxSTC_CLW_BUILTIN_PROCEDURES_FUNCTION 11

#define wxSTC_CLW_COMMENT 2

#define wxSTC_CLW_COMPILER_DIRECTIVE 9

#define wxSTC_CLW_DEFAULT 0

Lexical states for SCLEX_CLW.

#define wxSTC_CLW_DEPRECATED 16

#define wxSTC_CLW_ERROR 15

#define wxSTC_CLW_INTEGER_CONSTANT 5

#define wxSTC_CLW_KEYWORD 8

#define wxSTC_CLW_LABEL 1

#define wxSTC_CLW_PICTURE_STRING 7

#define wxSTC_CLW_REAL_CONSTANT 6

#define wxSTC_CLW_RUNTIME_EXPRESSIONS 10

#define wxSTC_CLW_STANDARD_EQUATE 14

#define wxSTC_CLW_STRING 3

#define wxSTC_CLW_STRUCTURE_DATA_TYPE 12

#define wxSTC_CLW_USER_IDENTIFIER 4

#define wxSTC_CMAKE_COMMANDS 5

#define wxSTC_CMAKE_COMMENT 1

#define wxSTC_CMAKE_DEFAULT 0

Lexical states for SCLEX_CMAKE.

#define wxSTC_CMAKE_FOREACHDEF 10

#define wxSTC_CMAKE_IFDEFINEDEF 11

#define wxSTC_CMAKE_MACRODEF 12

#define wxSTC_CMAKE_NUMBER 14

#define wxSTC_CMAKE_PARAMETERS 6

#define wxSTC_CMAKE_STRINGDQ 2

#define wxSTC_CMAKE_STRINGLQ 3

#define wxSTC_CMAKE_STRINGRQ 4

#define wxSTC_CMAKE_STRINGVAR 13

#define wxSTC_CMAKE_USERDEFINED 8

#define wxSTC_CMAKE_VARIABLE 7

#define wxSTC_CMAKE_WHILEDEF 9

#define wxSTC_CMD_BACKTAB 2328

Dedent the selected lines.

#define wxSTC_CMD_CANCEL 2325

Cancel any modes such as call tip or auto-completion list display.

#define wxSTC_CMD_CHARLEFT 2304

Move caret left one character.

#define wxSTC_CMD_CHARLEFTEXTEND 2305

Move caret left one character extending selection to new caret position.

#define wxSTC_CMD_CHARLEFTRECTEXTEND 2428

Move caret left one character, extending rectangular selection to new caret position.

#define wxSTC_CMD_CHARRIGHT 2306

Move caret right one character.

#define wxSTC_CMD_CHARRIGHTEXTEND 2307

Move caret right one character extending selection to new caret position.

#define wxSTC_CMD_CHARRIGHTRECTEXTEND 2429

Move caret right one character, extending rectangular selection to new caret position.

#define wxSTC_CMD_CLEAR 2180

Clear the selection.

#define wxSTC_CMD_COPY 2178

Copy the selection to the clipboard.

#define wxSTC_CMD_CUT 2177

Cut the selection to the clipboard.

#define wxSTC_CMD_DELETEBACK 2326

Delete the selection or if no selection, the character before the caret.

#define wxSTC_CMD_DELETEBACKNOTLINE 2344

Delete the selection or if no selection, the character before the caret.
Will not delete the character before at the start of a line.

#define wxSTC_CMD_DELLINELEFT 2395

Delete back from the current position to the start of the line.

#define wxSTC_CMD_DELLINERIGHT 2396

Delete forwards from the current position to the end of the line.

#define wxSTC_CMD_DELWORDLEFT 2335

Delete the word to the left of the caret.

#define wxSTC_CMD_DELWORDRIGHT 2336

Delete the word to the right of the caret.

#define wxSTC_CMD_DELWORDRIGHTEND 2518

Delete the word to the right of the caret, but not the trailing non-word characters.

#define wxSTC_CMD_DOCUMENTEND 2318

Move caret to last position in document.

#define wxSTC_CMD_DOCUMENTENDEXTEND 2319

Move caret to last position in document extending selection to new caret position.

#define wxSTC_CMD_DOCUMENTSTART 2316

Move caret to first position in document.

#define wxSTC_CMD_DOCUMENTSTARTEXTEND 2317

Move caret to first position in document extending selection to new caret position.

#define wxSTC_CMD_EDITTOGGLEOVERTYPE 2324

Switch from insert to overtype mode or the reverse.

#define wxSTC_CMD_FORMFEED 2330

Insert a Form Feed character.

#define wxSTC_CMD_HOME 2312

Move caret to first position on line.

#define wxSTC_CMD_HOMEDISPLAY 2345

Move caret to first position on display line.

#define wxSTC_CMD_HOMEDISPLAYEXTEND 2346

Move caret to first position on display line extending selection to new caret position.

#define wxSTC_CMD_HOMEEXTEND 2313

Move caret to first position on line extending selection to new caret position.

#define wxSTC_CMD_HOMERECTEXTEND 2430

Move caret to first position on line, extending rectangular selection to new caret position.

#define wxSTC_CMD_HOMEWRAP 2349

These are like their namesakes Home(Extend)?, LineEnd(Extend)?, VCHome(Extend)? except they behave differently when word-wrap is enabled: They go first to the start / end of the display line, like (Home|LineEnd)Display The difference is that, the cursor is already at the point, it goes on to the start or end of the document line, as appropriate for (Home|LineEnd|VCHome)(Extend)?.

#define wxSTC_CMD_HOMEWRAPEXTEND 2450

#define wxSTC_CMD_LINECOPY 2455

Copy the line containing the caret.

#define wxSTC_CMD_LINECUT 2337

Cut the line containing the caret.

#define wxSTC_CMD_LINEDELETE 2338

Delete the line containing the caret.

#define wxSTC_CMD_LINEDOWN 2300

Move caret down one line.

#define wxSTC_CMD_LINEDOWNEXTEND 2301

Move caret down one line extending selection to new caret position.

#define wxSTC_CMD_LINEDOWNRECTEXTEND 2426

Move caret down one line, extending rectangular selection to new caret position.

#define wxSTC_CMD_LINEDUPLICATE 2404

Duplicate the current line.

#define wxSTC_CMD_LINEEND 2314

Move caret to last position on line.

#define wxSTC_CMD_LINEENDDISPLAY 2347

Move caret to last position on display line.

#define wxSTC_CMD_LINEENDDISPLAYEXTEND 2348

Move caret to last position on display line extending selection to new caret position.

#define wxSTC_CMD_LINEENDEXTEND 2315

Move caret to last position on line extending selection to new caret position.

#define wxSTC_CMD_LINEENDRECTEXTEND 2432

Move caret to last position on line, extending rectangular selection to new caret position.

#define wxSTC_CMD_LINEENDWRAP 2451

#define wxSTC_CMD_LINEENDWRAPEXTEND 2452

#define wxSTC_CMD_LINESCROLLDOWN 2342

Scroll the document down, keeping the caret visible.

#define wxSTC_CMD_LINESCROLLUP 2343

Scroll the document up, keeping the caret visible.

#define wxSTC_CMD_LINETRANSPOSE 2339

Switch the current line with the previous.

#define wxSTC_CMD_LINEUP 2302

Move caret up one line.

#define wxSTC_CMD_LINEUPEXTEND 2303

Move caret up one line extending selection to new caret position.

#define wxSTC_CMD_LINEUPRECTEXTEND 2427

Move caret up one line, extending rectangular selection to new caret position.

#define wxSTC_CMD_LOWERCASE 2340

Transform the selection to lower case.

#define wxSTC_CMD_MOVESELECTEDLINESDOWN 2621

Move the selected lines down one line, shifting the line below before the selection.

#define wxSTC_CMD_MOVESELECTEDLINESUP 2620

Move the selected lines up one line, shifting the line above after the selection.

#define wxSTC_CMD_NEWLINE 2329

Insert a new line, may use a CRLF, CR or LF depending on EOL mode.

#define wxSTC_CMD_PAGEDOWN 2322

Move caret one page down.

#define wxSTC_CMD_PAGEDOWNEXTEND 2323

Move caret one page down extending selection to new caret position.

#define wxSTC_CMD_PAGEDOWNRECTEXTEND 2434

Move caret one page down, extending rectangular selection to new caret position.

#define wxSTC_CMD_PAGEUP 2320

Move caret one page up.

#define wxSTC_CMD_PAGEUPEXTEND 2321

Move caret one page up extending selection to new caret position.

#define wxSTC_CMD_PAGEUPRECTEXTEND 2433

Move caret one page up, extending rectangular selection to new caret position.

#define wxSTC_CMD_PARADOWN 2413

Move caret between paragraphs (delimited by empty lines).

#define wxSTC_CMD_PARADOWNEXTEND 2414

#define wxSTC_CMD_PARAUP 2415

#define wxSTC_CMD_PARAUPEXTEND 2416

#define wxSTC_CMD_PASTE 2179

Paste the contents of the clipboard into the document replacing the selection.

#define wxSTC_CMD_REDO 2011

Redoes the next action on the undo history.

#define wxSTC_CMD_SCROLLTOEND 2629

Scroll to end of document.

#define wxSTC_CMD_SCROLLTOSTART 2628

Scroll to start of document.

#define wxSTC_CMD_SELECTALL 2013

Select all the text in the document.

#define wxSTC_CMD_STUTTEREDPAGEDOWN 2437

Move caret to bottom of page, or one page down if already at bottom of page.

#define wxSTC_CMD_STUTTEREDPAGEDOWNEXTEND 2438

Move caret to bottom of page, or one page down if already at bottom of page, extending selection to new caret position.

#define wxSTC_CMD_STUTTEREDPAGEUP 2435

Move caret to top of page, or one page up if already at top of page.

#define wxSTC_CMD_STUTTEREDPAGEUPEXTEND 2436

Move caret to top of page, or one page up if already at top of page, extending selection to new caret position.

#define wxSTC_CMD_TAB 2327

If selection is empty or all on one line replace the selection with a tab character.

If more than one line selected, indent the lines.

#define wxSTC_CMD_UNDO 2176

Undo one action in the undo history.

#define wxSTC_CMD_UPPERCASE 2341

Transform the selection to upper case.

#define wxSTC_CMD_VCHOME 2331

Move caret to before first visible character on line.

If already there move to first character on line.

#define wxSTC_CMD_VCHOMEDISPLAY 2652

Move caret to before first visible character on display line.

If already there move to first character on display line.

#define wxSTC_CMD_VCHOMEDISPLAYEXTEND 2653

Like VCHomeDisplay but extending selection to new caret position.

#define wxSTC_CMD_VCHOMEEXTEND 2332

Like VCHome but extending selection to new caret position.

#define wxSTC_CMD_VCHOMERECTEXTEND 2431

Move caret to before first visible character on line.

If already there move to first character on line. In either case, extend rectangular selection to new caret position.

#define wxSTC_CMD_VCHOMEWRAP 2453

#define wxSTC_CMD_VCHOMEWRAPEXTEND 2454

#define wxSTC_CMD_VERTICALCENTRECARET 2619

Centre current line in window.

#define wxSTC_CMD_WORDLEFT 2308

Move caret left one word.

#define wxSTC_CMD_WORDLEFTEND 2439

Move caret left one word, position cursor at end of word.

#define wxSTC_CMD_WORDLEFTENDEXTEND 2440

Move caret left one word, position cursor at end of word, extending selection to new caret position.

#define wxSTC_CMD_WORDLEFTEXTEND 2309

Move caret left one word extending selection to new caret position.

#define wxSTC_CMD_WORDPARTLEFT 2390

Move to the previous change in capitalisation.

#define wxSTC_CMD_WORDPARTLEFTEXTEND 2391

Move to the previous change in capitalisation extending selection to new caret position.

#define wxSTC_CMD_WORDPARTRIGHT 2392

Move to the change next in capitalisation.

#define wxSTC_CMD_WORDPARTRIGHTEXTEND 2393

Move to the next change in capitalisation extending selection to new caret position.

#define wxSTC_CMD_WORDRIGHT 2310

Move caret right one word.

#define wxSTC_CMD_WORDRIGHTEND 2441

Move caret right one word, position cursor at end of word.

#define wxSTC_CMD_WORDRIGHTENDEXTEND 2442

Move caret right one word, position cursor at end of word, extending selection to new caret position.

#define wxSTC_CMD_WORDRIGHTEXTEND 2311

Move caret right one word extending selection to new caret position.

#define wxSTC_CMD_ZOOMIN 2333

Magnify the displayed text by increasing the sizes by 1 point.

#define wxSTC_CMD_ZOOMOUT 2334

Make the displayed text smaller by decreasing the sizes by 1 point.

#define wxSTC_COFFEESCRIPT_CHARACTER 7

#define wxSTC_COFFEESCRIPT_COMMENT 1

#define wxSTC_COFFEESCRIPT_COMMENTBLOCK 22

#define wxSTC_COFFEESCRIPT_COMMENTDOC 3

#define wxSTC_COFFEESCRIPT_COMMENTDOCKEYWORD 17

#define wxSTC_COFFEESCRIPT_COMMENTDOCKEYWORDERROR 18

#define wxSTC_COFFEESCRIPT_COMMENTLINE 2

#define wxSTC_COFFEESCRIPT_COMMENTLINEDOC 15

#define wxSTC_COFFEESCRIPT_DEFAULT 0

Lexical states for SCLEX_COFFEESCRIPT.

#define wxSTC_COFFEESCRIPT_GLOBALCLASS 19

#define wxSTC_COFFEESCRIPT_IDENTIFIER 11

#define wxSTC_COFFEESCRIPT_NUMBER 4

#define wxSTC_COFFEESCRIPT_OPERATOR 10

#define wxSTC_COFFEESCRIPT_PREPROCESSOR 9

#define wxSTC_COFFEESCRIPT_REGEX 14

#define wxSTC_COFFEESCRIPT_STRING 6

#define wxSTC_COFFEESCRIPT_STRINGEOL 12

#define wxSTC_COFFEESCRIPT_STRINGRAW 20

#define wxSTC_COFFEESCRIPT_TRIPLEVERBATIM 21

#define wxSTC_COFFEESCRIPT_UUID 8

#define wxSTC_COFFEESCRIPT_VERBATIM 13

#define wxSTC_COFFEESCRIPT_VERBOSE_REGEX 23

#define wxSTC_COFFEESCRIPT_VERBOSE_REGEX_COMMENT 24

#define wxSTC_COFFEESCRIPT_WORD 5

#define wxSTC_COFFEESCRIPT_WORD2 16

#define wxSTC_CONF_COMMENT 1

#define wxSTC_CONF_DEFAULT 0

Lexical states for SCLEX_CONF (Apache Configuration Files Lexer)

#define wxSTC_CONF_DIRECTIVE 9

#define wxSTC_CONF_EXTENSION 4

#define wxSTC_CONF_IDENTIFIER 3

#define wxSTC_CONF_IP 8

#define wxSTC_CONF_NUMBER 2

#define wxSTC_CONF_OPERATOR 7

#define wxSTC_CONF_PARAMETER 5

#define wxSTC_CONF_STRING 6

#define wxSTC_CP_UTF8 65001

The SC_CP_UTF8 value can be used to enter Unicode mode.

This is the same value as CP_UTF8 in Windows

#define wxSTC_CSOUND_ARATE_VAR 11

#define wxSTC_CSOUND_COMMENT 1

#define wxSTC_CSOUND_COMMENTBLOCK 9

#define wxSTC_CSOUND_DEFAULT 0

Lexical states for SCLEX_CSOUND.

#define wxSTC_CSOUND_GLOBAL_VAR 14

#define wxSTC_CSOUND_HEADERSTMT 7

#define wxSTC_CSOUND_IDENTIFIER 5

#define wxSTC_CSOUND_INSTR 4

#define wxSTC_CSOUND_IRATE_VAR 13

#define wxSTC_CSOUND_KRATE_VAR 12

#define wxSTC_CSOUND_NUMBER 2

#define wxSTC_CSOUND_OPCODE 6

#define wxSTC_CSOUND_OPERATOR 3

#define wxSTC_CSOUND_PARAM 10

#define wxSTC_CSOUND_STRINGEOL 15

#define wxSTC_CSOUND_USERKEYWORD 8

#define wxSTC_CSS_ATTRIBUTE 16

#define wxSTC_CSS_CLASS 2

#define wxSTC_CSS_COMMENT 9

#define wxSTC_CSS_DEFAULT 0

Lexical states for SCLEX_CSS.

#define wxSTC_CSS_DIRECTIVE 12

#define wxSTC_CSS_DOUBLESTRING 13

#define wxSTC_CSS_EXTENDED_IDENTIFIER 19

#define wxSTC_CSS_EXTENDED_PSEUDOCLASS 20

#define wxSTC_CSS_EXTENDED_PSEUDOELEMENT 21

#define wxSTC_CSS_ID 10

#define wxSTC_CSS_IDENTIFIER 6

#define wxSTC_CSS_IDENTIFIER2 15

#define wxSTC_CSS_IDENTIFIER3 17

#define wxSTC_CSS_IMPORTANT 11

#define wxSTC_CSS_MEDIA 22

#define wxSTC_CSS_OPERATOR 5

#define wxSTC_CSS_PSEUDOCLASS 3

#define wxSTC_CSS_PSEUDOELEMENT 18

#define wxSTC_CSS_SINGLESTRING 14

#define wxSTC_CSS_TAG 1

#define wxSTC_CSS_UNKNOWN_IDENTIFIER 7

#define wxSTC_CSS_UNKNOWN_PSEUDOCLASS 4

#define wxSTC_CSS_VALUE 8

#define wxSTC_CSS_VARIABLE 23

#define wxSTC_CURSORARROW 2

#define wxSTC_CURSORNORMAL -1

#define wxSTC_CURSORREVERSEARROW 7

#define wxSTC_CURSORWAIT 4

#define wxSTC_D_CHARACTER 12

#define wxSTC_D_COMMENT 1

#define wxSTC_D_COMMENTDOC 3

#define wxSTC_D_COMMENTDOCKEYWORD 16

#define wxSTC_D_COMMENTDOCKEYWORDERROR 17

#define wxSTC_D_COMMENTLINE 2

#define wxSTC_D_COMMENTLINEDOC 15

#define wxSTC_D_COMMENTNESTED 4

#define wxSTC_D_DEFAULT 0

Lexical states for SCLEX_D.

#define wxSTC_D_IDENTIFIER 14

#define wxSTC_D_NUMBER 5

#define wxSTC_D_OPERATOR 13

#define wxSTC_D_STRING 10

#define wxSTC_D_STRINGB 18

#define wxSTC_D_STRINGEOL 11

#define wxSTC_D_STRINGR 19

#define wxSTC_D_TYPEDEF 9

#define wxSTC_D_WORD 6

#define wxSTC_D_WORD2 7

#define wxSTC_D_WORD3 8

#define wxSTC_D_WORD5 20

#define wxSTC_D_WORD6 21

#define wxSTC_D_WORD7 22

#define wxSTC_DIFF_ADDED 6

#define wxSTC_DIFF_CHANGED 7

#define wxSTC_DIFF_COMMAND 2

#define wxSTC_DIFF_COMMENT 1

#define wxSTC_DIFF_DEFAULT 0

Lexical states for SCLEX_DIFF.

#define wxSTC_DIFF_DELETED 5

#define wxSTC_DIFF_HEADER 3

#define wxSTC_DIFF_POSITION 4

#define wxSTC_DMAP_COMMENT 1

#define wxSTC_DMAP_DEFAULT 0

Lexical states for SCLEX_DMAP.

#define wxSTC_DMAP_IDENTIFIER 7

#define wxSTC_DMAP_NUMBER 2

#define wxSTC_DMAP_OPERATOR 6

#define wxSTC_DMAP_STRING1 3

#define wxSTC_DMAP_STRING2 4

#define wxSTC_DMAP_STRINGEOL 5

#define wxSTC_DMAP_WORD 8

#define wxSTC_DMAP_WORD2 9

#define wxSTC_DMAP_WORD3 10

#define wxSTC_ECL_ADDED 24

#define wxSTC_ECL_CHANGED 26

#define wxSTC_ECL_CHARACTER 7

#define wxSTC_ECL_COMMENT 1

#define wxSTC_ECL_COMMENTDOC 23

#define wxSTC_ECL_COMMENTDOCKEYWORD 17

#define wxSTC_ECL_COMMENTDOCKEYWORDERROR 18

#define wxSTC_ECL_COMMENTLINE 2

#define wxSTC_ECL_COMMENTLINEDOC 15

#define wxSTC_ECL_DEFAULT 0

Lexical states for SCLEX_ECL.

#define wxSTC_ECL_DELETED 25

#define wxSTC_ECL_IDENTIFIER 11

#define wxSTC_ECL_MOVED 27

#define wxSTC_ECL_NUMBER 3

#define wxSTC_ECL_OPERATOR 6

#define wxSTC_ECL_PREPROCESSOR 9

#define wxSTC_ECL_REGEX 14

#define wxSTC_ECL_STRING 4

#define wxSTC_ECL_STRINGEOL 12

#define wxSTC_ECL_UNKNOWN 10

#define wxSTC_ECL_UUID 8

#define wxSTC_ECL_VERBATIM 13

#define wxSTC_ECL_WORD0 5

#define wxSTC_ECL_WORD1 16

#define wxSTC_ECL_WORD2 19

#define wxSTC_ECL_WORD3 20

#define wxSTC_ECL_WORD4 21

#define wxSTC_ECL_WORD5 22

#define wxSTC_EDGE_BACKGROUND 2

#define wxSTC_EDGE_LINE 1

#define wxSTC_EDGE_NONE 0

#define wxSTC_EFF_QUALITY_ANTIALIASED 2

#define wxSTC_EFF_QUALITY_DEFAULT 0

#define wxSTC_EFF_QUALITY_LCD_OPTIMIZED 3

#define wxSTC_EFF_QUALITY_MASK 0xF

Control font anti-aliasing.

#define wxSTC_EFF_QUALITY_NON_ANTIALIASED 1

#define wxSTC_EIFFEL_CHARACTER 5

#define wxSTC_EIFFEL_COMMENTLINE 1

#define wxSTC_EIFFEL_DEFAULT 0

Lexical states for SCLEX_EIFFEL and SCLEX_EIFFELKW.

#define wxSTC_EIFFEL_IDENTIFIER 7

#define wxSTC_EIFFEL_NUMBER 2

#define wxSTC_EIFFEL_OPERATOR 6

#define wxSTC_EIFFEL_STRING 4

#define wxSTC_EIFFEL_STRINGEOL 8

#define wxSTC_EIFFEL_WORD 3

#define wxSTC_EOL_CR 1

#define wxSTC_EOL_CRLF 0

#define wxSTC_EOL_LF 2

#define wxSTC_ERLANG_ATOM 7

#define wxSTC_ERLANG_ATOM_QUOTED 18

#define wxSTC_ERLANG_BIFS 22

#define wxSTC_ERLANG_CHARACTER 9

#define wxSTC_ERLANG_COMMENT 1

#define wxSTC_ERLANG_COMMENT_DOC 16

#define wxSTC_ERLANG_COMMENT_DOC_MACRO 17

#define wxSTC_ERLANG_COMMENT_FUNCTION 14

#define wxSTC_ERLANG_COMMENT_MODULE 15

#define wxSTC_ERLANG_DEFAULT 0

Lexical states for SCLEX_ERLANG.

#define wxSTC_ERLANG_FUNCTION_NAME 8

#define wxSTC_ERLANG_KEYWORD 4

#define wxSTC_ERLANG_MACRO 10

#define wxSTC_ERLANG_MACRO_QUOTED 19

#define wxSTC_ERLANG_MODULES 23

#define wxSTC_ERLANG_MODULES_ATT 24

#define wxSTC_ERLANG_NODE_NAME 13

#define wxSTC_ERLANG_NODE_NAME_QUOTED 21

#define wxSTC_ERLANG_NUMBER 3

#define wxSTC_ERLANG_OPERATOR 6

#define wxSTC_ERLANG_PREPROC 12

#define wxSTC_ERLANG_RECORD 11

#define wxSTC_ERLANG_RECORD_QUOTED 20

#define wxSTC_ERLANG_STRING 5

#define wxSTC_ERLANG_UNKNOWN 31

#define wxSTC_ERLANG_VARIABLE 2

#define wxSTC_ERR_ABSF 18

#define wxSTC_ERR_BORLAND 5

#define wxSTC_ERR_CMD 4

#define wxSTC_ERR_CTAG 9

#define wxSTC_ERR_DEFAULT 0

Lexical states for SCLEX_ERRORLIST.

#define wxSTC_ERR_DIFF_ADDITION 11

#define wxSTC_ERR_DIFF_CHANGED 10

#define wxSTC_ERR_DIFF_DELETION 12

#define wxSTC_ERR_DIFF_MESSAGE 13

#define wxSTC_ERR_ELF 15

#define wxSTC_ERR_GCC 2

#define wxSTC_ERR_GCC_INCLUDED_FROM 22

#define wxSTC_ERR_IFC 16

#define wxSTC_ERR_IFORT 17

#define wxSTC_ERR_JAVA_STACK 20

#define wxSTC_ERR_LUA 8

#define wxSTC_ERR_MS 3

#define wxSTC_ERR_NET 7

#define wxSTC_ERR_PERL 6

#define wxSTC_ERR_PHP 14

#define wxSTC_ERR_PYTHON 1

#define wxSTC_ERR_TIDY 19

#define wxSTC_ERR_VALUE 21

#define wxSTC_ESCRIPT_BRACE 9

#define wxSTC_ESCRIPT_COMMENT 1

#define wxSTC_ESCRIPT_COMMENTDOC 3

#define wxSTC_ESCRIPT_COMMENTLINE 2

#define wxSTC_ESCRIPT_DEFAULT 0

Lexical states for SCLEX_ESCRIPT.

#define wxSTC_ESCRIPT_IDENTIFIER 8

#define wxSTC_ESCRIPT_NUMBER 4

#define wxSTC_ESCRIPT_OPERATOR 7

#define wxSTC_ESCRIPT_STRING 6

#define wxSTC_ESCRIPT_WORD 5

#define wxSTC_ESCRIPT_WORD2 10

#define wxSTC_ESCRIPT_WORD3 11

#define wxSTC_F_COMMENT 1

#define wxSTC_F_CONTINUATION 14

#define wxSTC_F_DEFAULT 0

Lexical states for SCLEX_FORTRAN.

#define wxSTC_F_IDENTIFIER 7

#define wxSTC_F_LABEL 13

#define wxSTC_F_NUMBER 2

#define wxSTC_F_OPERATOR 6

#define wxSTC_F_OPERATOR2 12

#define wxSTC_F_PREPROCESSOR 11

#define wxSTC_F_STRING1 3

#define wxSTC_F_STRING2 4

#define wxSTC_F_STRINGEOL 5

#define wxSTC_F_WORD 8

#define wxSTC_F_WORD2 9

#define wxSTC_F_WORD3 10

#define wxSTC_FIND_MATCHCASE 0x4

#define wxSTC_FIND_POSIX 0x00400000

#define wxSTC_FIND_REGEXP 0x00200000

#define wxSTC_FIND_WHOLEWORD 0x2

#define wxSTC_FIND_WORDSTART 0x00100000

#define wxSTC_FOLDACTION_CONTRACT 0

#define wxSTC_FOLDACTION_EXPAND 1

#define wxSTC_FOLDACTION_TOGGLE 2

#define wxSTC_FOLDFLAG_LEVELNUMBERS 0x0040

#define wxSTC_FOLDFLAG_LINEAFTER_CONTRACTED 0x0010

#define wxSTC_FOLDFLAG_LINEAFTER_EXPANDED 0x0008

#define wxSTC_FOLDFLAG_LINEBEFORE_CONTRACTED 0x0004

#define wxSTC_FOLDFLAG_LINEBEFORE_EXPANDED 0x0002

#define wxSTC_FOLDLEVELBASE 0x400

#define wxSTC_FOLDLEVELHEADERFLAG 0x2000

#define wxSTC_FOLDLEVELNUMBERMASK 0x0FFF

#define wxSTC_FOLDLEVELWHITEFLAG 0x1000

#define wxSTC_FONT_SIZE_MULTIPLIER 100

#define wxSTC_FORTH_COMMENT 1

#define wxSTC_FORTH_COMMENT_ML 2

#define wxSTC_FORTH_CONTROL 4

#define wxSTC_FORTH_DEFAULT 0

Lexical states for SCLEX_FORTH (Forth Lexer)

#define wxSTC_FORTH_DEFWORD 6

#define wxSTC_FORTH_IDENTIFIER 3

#define wxSTC_FORTH_KEYWORD 5

#define wxSTC_FORTH_LOCALE 11

#define wxSTC_FORTH_NUMBER 9

#define wxSTC_FORTH_PREWORD1 7

#define wxSTC_FORTH_PREWORD2 8

#define wxSTC_FORTH_STRING 10

#define wxSTC_FS_COMMENT 1

#define wxSTC_FS_COMMENTDOC 3

#define wxSTC_FS_COMMENTDOC_C 22

#define wxSTC_FS_COMMENTDOCKEYWORD 5

#define wxSTC_FS_COMMENTDOCKEYWORDERROR 6

#define wxSTC_FS_COMMENTLINE 2

#define wxSTC_FS_COMMENTLINEDOC 4

#define wxSTC_FS_COMMENTLINEDOC_C 23

#define wxSTC_FS_CONSTANT 18

#define wxSTC_FS_DATE 16

#define wxSTC_FS_DEFAULT 0

Lexical states for SCLEX_FLAGSHIP (clipper)

#define wxSTC_FS_DEFAULT_C 21

#define wxSTC_FS_DISABLEDCODE 20

#define wxSTC_FS_IDENTIFIER 15

#define wxSTC_FS_IDENTIFIER_C 30

#define wxSTC_FS_KEYWORD 7

#define wxSTC_FS_KEYWORD2 8

#define wxSTC_FS_KEYWORD2_C 25

#define wxSTC_FS_KEYWORD3 9

#define wxSTC_FS_KEYWORD4 10

#define wxSTC_FS_KEYWORD_C 24

#define wxSTC_FS_NUMBER 11

#define wxSTC_FS_NUMBER_C 26

#define wxSTC_FS_OPERATOR 14

#define wxSTC_FS_OPERATOR_C 29

#define wxSTC_FS_PREPROCESSOR 13

#define wxSTC_FS_PREPROCESSOR_C 28

#define wxSTC_FS_STRING 12

#define wxSTC_FS_STRING_C 27

#define wxSTC_FS_STRINGEOL 17

#define wxSTC_FS_STRINGEOL_C 31

#define wxSTC_FS_WORDOPERATOR 19

#define wxSTC_GAP_CHAR 7

#define wxSTC_GAP_COMMENT 9

#define wxSTC_GAP_DEFAULT 0

Lexical states for SCLEX_GAP.

#define wxSTC_GAP_IDENTIFIER 1

#define wxSTC_GAP_KEYWORD 2

#define wxSTC_GAP_KEYWORD2 3

#define wxSTC_GAP_KEYWORD3 4

#define wxSTC_GAP_KEYWORD4 5

#define wxSTC_GAP_NUMBER 10

#define wxSTC_GAP_OPERATOR 8

#define wxSTC_GAP_STRING 6

#define wxSTC_GAP_STRINGEOL 11

#define wxSTC_GC_ATTRIBUTE 5

#define wxSTC_GC_COMMAND 7

#define wxSTC_GC_COMMENTBLOCK 2

#define wxSTC_GC_COMMENTLINE 1

#define wxSTC_GC_CONTROL 6

#define wxSTC_GC_DEFAULT 0

Lexical states for SCLEX_GUI4CLI.

#define wxSTC_GC_EVENT 4

#define wxSTC_GC_GLOBAL 3

#define wxSTC_GC_OPERATOR 9

#define wxSTC_GC_STRING 8

#define wxSTC_H_ASP 15

#define wxSTC_H_ASPAT 16

#define wxSTC_H_ATTRIBUTE 3

#define wxSTC_H_ATTRIBUTEUNKNOWN 4

#define wxSTC_H_CDATA 17

#define wxSTC_H_COMMENT 9

#define wxSTC_H_DEFAULT 0

Lexical states for SCLEX_HTML, SCLEX_XML.

#define wxSTC_H_DOUBLESTRING 6

#define wxSTC_H_ENTITY 10

#define wxSTC_H_NUMBER 5

#define wxSTC_H_OTHER 8

#define wxSTC_H_QUESTION 18

#define wxSTC_H_SCRIPT 14

#define wxSTC_H_SGML_1ST_PARAM 23

#define wxSTC_H_SGML_1ST_PARAM_COMMENT 30

#define wxSTC_H_SGML_BLOCK_DEFAULT 31

#define wxSTC_H_SGML_COMMAND 22

#define wxSTC_H_SGML_COMMENT 29

#define wxSTC_H_SGML_DEFAULT 21

SGML.

#define wxSTC_H_SGML_DOUBLESTRING 24

#define wxSTC_H_SGML_ENTITY 28

#define wxSTC_H_SGML_ERROR 26

#define wxSTC_H_SGML_SIMPLESTRING 25

#define wxSTC_H_SGML_SPECIAL 27

#define wxSTC_H_SINGLESTRING 7

#define wxSTC_H_TAG 1

#define wxSTC_H_TAGEND 11

XML and ASP.

#define wxSTC_H_TAGUNKNOWN 2

#define wxSTC_H_VALUE 19

More HTML.

#define wxSTC_H_XCCOMMENT 20

X-Code.

#define wxSTC_H_XMLEND 13

#define wxSTC_H_XMLSTART 12

#define wxSTC_HA_CAPITAL 8

#define wxSTC_HA_CHARACTER 5

#define wxSTC_HA_CLASS 6

#define wxSTC_HA_COMMENTBLOCK 14

#define wxSTC_HA_COMMENTBLOCK2 15

#define wxSTC_HA_COMMENTBLOCK3 16

#define wxSTC_HA_COMMENTLINE 13

#define wxSTC_HA_DATA 9

#define wxSTC_HA_DEFAULT 0

Lexical states for SCLEX_HASKELL.

#define wxSTC_HA_IDENTIFIER 1

#define wxSTC_HA_IMPORT 10

#define wxSTC_HA_INSTANCE 12

#define wxSTC_HA_KEYWORD 2

#define wxSTC_HA_LITERATE_CODEDELIM 22

#define wxSTC_HA_LITERATE_COMMENT 21

#define wxSTC_HA_MODULE 7

#define wxSTC_HA_NUMBER 3

#define wxSTC_HA_OPERATOR 11

#define wxSTC_HA_PRAGMA 17

#define wxSTC_HA_PREPROCESSOR 18

#define wxSTC_HA_RESERVED_OPERATOR 20

#define wxSTC_HA_STRING 4

#define wxSTC_HA_STRINGEOL 19

#define wxSTC_HB_COMMENTLINE 72

#define wxSTC_HB_DEFAULT 71

#define wxSTC_HB_IDENTIFIER 76

#define wxSTC_HB_NUMBER 73

#define wxSTC_HB_START 70

Embedded VBScript.

#define wxSTC_HB_STRING 75

#define wxSTC_HB_STRINGEOL 77

#define wxSTC_HB_WORD 74

#define wxSTC_HBA_COMMENTLINE 82

#define wxSTC_HBA_DEFAULT 81

#define wxSTC_HBA_IDENTIFIER 86

#define wxSTC_HBA_NUMBER 83

#define wxSTC_HBA_START 80

ASP VBScript.

#define wxSTC_HBA_STRING 85

#define wxSTC_HBA_STRINGEOL 87

#define wxSTC_HBA_WORD 84

#define wxSTC_HJ_COMMENT 42

#define wxSTC_HJ_COMMENTDOC 44

#define wxSTC_HJ_COMMENTLINE 43

#define wxSTC_HJ_DEFAULT 41

#define wxSTC_HJ_DOUBLESTRING 48

#define wxSTC_HJ_KEYWORD 47

#define wxSTC_HJ_NUMBER 45

#define wxSTC_HJ_REGEX 52

#define wxSTC_HJ_SINGLESTRING 49

#define wxSTC_HJ_START 40

Embedded Javascript.

#define wxSTC_HJ_STRINGEOL 51

#define wxSTC_HJ_SYMBOLS 50

#define wxSTC_HJ_WORD 46

#define wxSTC_HJA_COMMENT 57

#define wxSTC_HJA_COMMENTDOC 59

#define wxSTC_HJA_COMMENTLINE 58

#define wxSTC_HJA_DEFAULT 56

#define wxSTC_HJA_DOUBLESTRING 63

#define wxSTC_HJA_KEYWORD 62

#define wxSTC_HJA_NUMBER 60

#define wxSTC_HJA_REGEX 67

#define wxSTC_HJA_SINGLESTRING 64

#define wxSTC_HJA_START 55

ASP Javascript.

#define wxSTC_HJA_STRINGEOL 66

#define wxSTC_HJA_SYMBOLS 65

#define wxSTC_HJA_WORD 61

#define wxSTC_HP_CHARACTER 95

#define wxSTC_HP_CLASSNAME 99

#define wxSTC_HP_COMMENTLINE 92

#define wxSTC_HP_DEFAULT 91

#define wxSTC_HP_DEFNAME 100

#define wxSTC_HP_IDENTIFIER 102

#define wxSTC_HP_NUMBER 93

#define wxSTC_HP_OPERATOR 101

#define wxSTC_HP_START 90

Embedded Python.

#define wxSTC_HP_STRING 94

#define wxSTC_HP_TRIPLE 97

#define wxSTC_HP_TRIPLEDOUBLE 98

#define wxSTC_HP_WORD 96

#define wxSTC_HPA_CHARACTER 110

#define wxSTC_HPA_CLASSNAME 114

#define wxSTC_HPA_COMMENTLINE 107

#define wxSTC_HPA_DEFAULT 106

#define wxSTC_HPA_DEFNAME 115

#define wxSTC_HPA_IDENTIFIER 117

#define wxSTC_HPA_NUMBER 108

#define wxSTC_HPA_OPERATOR 116

#define wxSTC_HPA_START 105

ASP Python.

#define wxSTC_HPA_STRING 109

#define wxSTC_HPA_TRIPLE 112

#define wxSTC_HPA_TRIPLEDOUBLE 113

#define wxSTC_HPA_WORD 111

#define wxSTC_HPHP_COMMENT 124

#define wxSTC_HPHP_COMMENTLINE 125

#define wxSTC_HPHP_COMPLEX_VARIABLE 104

PHP.

#define wxSTC_HPHP_DEFAULT 118

PHP.

#define wxSTC_HPHP_HSTRING 119

#define wxSTC_HPHP_HSTRING_VARIABLE 126

#define wxSTC_HPHP_NUMBER 122

#define wxSTC_HPHP_OPERATOR 127

#define wxSTC_HPHP_SIMPLESTRING 120

#define wxSTC_HPHP_VARIABLE 123

#define wxSTC_HPHP_WORD 121

#define wxSTC_INDIC0_MASK 0x20

#define wxSTC_INDIC1_MASK 0x40

#define wxSTC_INDIC2_MASK 0x80

#define wxSTC_INDIC_BOX 6

#define wxSTC_INDIC_COMPOSITIONTHICK 14

#define wxSTC_INDIC_CONTAINER 8

#define wxSTC_INDIC_DASH 9

#define wxSTC_INDIC_DIAGONAL 3

#define wxSTC_INDIC_DOTBOX 12

#define wxSTC_INDIC_DOTS 10

#define wxSTC_INDIC_HIDDEN 5

#define wxSTC_INDIC_MAX 31

#define wxSTC_INDIC_PLAIN 0

Indicator style enumeration and some constants.

#define wxSTC_INDIC_ROUNDBOX 7

#define wxSTC_INDIC_SQUIGGLE 1

#define wxSTC_INDIC_SQUIGGLELOW 11

#define wxSTC_INDIC_SQUIGGLEPIXMAP 13

#define wxSTC_INDIC_STRAIGHTBOX 8

#define wxSTC_INDIC_STRIKE 4

#define wxSTC_INDIC_TT 2

#define wxSTC_INDICS_MASK 0xE0

#define wxSTC_INNO_COMMENT 1

#define wxSTC_INNO_COMMENT_PASCAL 7

#define wxSTC_INNO_DEFAULT 0

Lexical states for SCLEX_INNOSETUP.

#define wxSTC_INNO_IDENTIFIER 12

#define wxSTC_INNO_INLINE_EXPANSION 6

#define wxSTC_INNO_KEYWORD 2

#define wxSTC_INNO_KEYWORD_PASCAL 8

#define wxSTC_INNO_KEYWORD_USER 9

#define wxSTC_INNO_PARAMETER 3

#define wxSTC_INNO_PREPROC 5

#define wxSTC_INNO_SECTION 4

#define wxSTC_INNO_STRING_DOUBLE 10

#define wxSTC_INNO_STRING_SINGLE 11

#define wxSTC_INVALID_POSITION -1

#define wxSTC_IV_LOOKBOTH 3

#define wxSTC_IV_LOOKFORWARD 2

#define wxSTC_IV_NONE 0

#define wxSTC_IV_REAL 1

#define wxSTC_KEY_ADD 310

#define wxSTC_KEY_BACK 8

#define wxSTC_KEY_DELETE 308

#define wxSTC_KEY_DIVIDE 312

#define wxSTC_KEY_DOWN 300

Symbolic key codes and modifier flags.

ASCII and other printable characters below 256. Extended keys above 300.

#define wxSTC_KEY_END 305

#define wxSTC_KEY_ESCAPE 7

#define wxSTC_KEY_HOME 304

#define wxSTC_KEY_INSERT 309

#define wxSTC_KEY_LEFT 302

#define wxSTC_KEY_MENU 315

#define wxSTC_KEY_NEXT 307

#define wxSTC_KEY_PRIOR 306

#define wxSTC_KEY_RETURN 13

#define wxSTC_KEY_RIGHT 303

#define wxSTC_KEY_RWIN 314

#define wxSTC_KEY_SUBTRACT 311

#define wxSTC_KEY_TAB 9

#define wxSTC_KEY_UP 301

#define wxSTC_KEY_WIN 313

#define wxSTC_KEYWORDSET_MAX 8

Maximum value of keywordSet parameter of SetKeyWords.

#define wxSTC_KIX_COMMENT 1

#define wxSTC_KIX_DEFAULT 0

Lexical states for SCLEX_KIX.

#define wxSTC_KIX_FUNCTIONS 8

#define wxSTC_KIX_IDENTIFIER 31

#define wxSTC_KIX_KEYWORD 7

#define wxSTC_KIX_MACRO 6

#define wxSTC_KIX_NUMBER 4

#define wxSTC_KIX_OPERATOR 9

#define wxSTC_KIX_STRING1 2

#define wxSTC_KIX_STRING2 3

#define wxSTC_KIX_VAR 5

#define wxSTC_KVIRC_COMMENT 1

#define wxSTC_KVIRC_COMMENTBLOCK 2

#define wxSTC_KVIRC_DEFAULT 0

Lexical states for SCLEX_KVIRC.

#define wxSTC_KVIRC_FUNCTION 7

#define wxSTC_KVIRC_FUNCTION_KEYWORD 6

#define wxSTC_KVIRC_KEYWORD 5

#define wxSTC_KVIRC_NUMBER 9

#define wxSTC_KVIRC_OPERATOR 10

#define wxSTC_KVIRC_STRING 3

#define wxSTC_KVIRC_STRING_FUNCTION 11

#define wxSTC_KVIRC_STRING_VARIABLE 12

#define wxSTC_KVIRC_VARIABLE 8

#define wxSTC_KVIRC_WORD 4

#define wxSTC_L_CMDOPT 11

#define wxSTC_L_COMMAND 1

#define wxSTC_L_COMMENT 4

#define wxSTC_L_COMMENT2 7

#define wxSTC_L_DEFAULT 0

Lexical states for SCLEX_LATEX.

#define wxSTC_L_ERROR 12

#define wxSTC_L_MATH 3

#define wxSTC_L_MATH2 6

#define wxSTC_L_SHORTCMD 9

#define wxSTC_L_SPECIAL 10

#define wxSTC_L_TAG 2

#define wxSTC_L_TAG2 5

#define wxSTC_L_VERBATIM 8

#define wxSTC_LASTSTEPINUNDOREDO 0x100

#define wxSTC_LEX_A68K 100

#define wxSTC_LEX_ABAQUS 84

#define wxSTC_LEX_ADA 20

#define wxSTC_LEX_APDL 61

#define wxSTC_LEX_AS 113

#define wxSTC_LEX_ASM 34

#define wxSTC_LEX_ASN1 63

#define wxSTC_LEX_ASYMPTOTE 85

#define wxSTC_LEX_AU3 60

#define wxSTC_LEX_AUTOMATIC 1000

When a lexer specifies its language as SCLEX_AUTOMATIC it receives a value assigned in sequence from SCL↩ EX_AUTOMATIC+1.

#define wxSTC_LEX_AVE 19

#define wxSTC_LEX_AVS 104

#define wxSTC_LEX_BAAN 31

#define wxSTC_LEX_BASH 62

#define wxSTC_LEX_BATCH 12

#define wxSTC_LEX_BLITZBASIC 66

#define wxSTC_LEX_BULLANT 27

#define wxSTC_LEX_CAML 65

#define wxSTC_LEX_CLW 45

#define wxSTC_LEX_CLWNOCASE 46

#define wxSTC_LEX_CMAKE 80

#define wxSTC_LEX_COBOL 92

#define wxSTC_LEX_COFFEESCRIPT 102

#define wxSTC_LEX_CONF 17

#define wxSTC_LEX_CONTAINER 0

For SciLexer.h.

#define wxSTC_LEX_CPP 3

#define wxSTC_LEX_CPPNOCASE 35

#define wxSTC_LEX_CSOUND 74

#define wxSTC_LEX_CSS 38

#define wxSTC_LEX_D 79

#define wxSTC_LEX_DIFF 16

#define wxSTC_LEX_DMAP 112

#define wxSTC_LEX_ECL 105

#define wxSTC_LEX_EIFFEL 23

#define wxSTC_LEX_EIFFELKW 24

#define wxSTC_LEX_ERLANG 53

#define wxSTC_LEX_ERRORLIST 10

#define wxSTC_LEX_ESCRIPT 41

#define wxSTC_LEX_F77 37

#define wxSTC_LEX_FLAGSHIP 73

#define wxSTC_LEX_FORTH 52

#define wxSTC_LEX_FORTRAN 36

#define wxSTC_LEX_FREEBASIC 75

#define wxSTC_LEX_GAP 81

#define wxSTC_LEX_GUI4CLI 58

#define wxSTC_LEX_HASKELL 68

#define wxSTC_LEX_HTML 4

#define wxSTC_LEX_INNOSETUP 76

#define wxSTC_LEX_KIX 57

#define wxSTC_LEX_KVIRC 110

#define wxSTC_LEX_LATEX 14

#define wxSTC_LEX_LISP 21

#define wxSTC_LEX_LITERATEHASKELL 108

#define wxSTC_LEX_LOT 47

#define wxSTC_LEX_LOUT 40

#define wxSTC_LEX_LUA 15

#define wxSTC_LEX_MAGIK 87

#define wxSTC_LEX_MAKEFILE 11

#define wxSTC_LEX_MARKDOWN 98

#define wxSTC_LEX_MATLAB 32

#define wxSTC_LEX_METAPOST 50

#define wxSTC_LEX_MMIXAL 44

#define wxSTC_LEX_MODULA 101

#define wxSTC_LEX_MSSQL 55

#define wxSTC_LEX_MYSQL 89

#define wxSTC_LEX_NIMROD 96

#define wxSTC_LEX_NNCRONTAB 26

#define wxSTC_LEX_NSIS 43

#define wxSTC_LEX_NULL 1

#define wxSTC_LEX_OCTAVE 54

#define wxSTC_LEX_OPAL 77

#define wxSTC_LEX_OSCRIPT 106

#define wxSTC_LEX_PASCAL 18

#define wxSTC_LEX_PERL 6

#define wxSTC_LEX_PHPSCRIPT 69

#define wxSTC_LEX_PLM 82

#define wxSTC_LEX_PO 90

#define wxSTC_LEX_POV 39

#define wxSTC_LEX_POWERBASIC 51

#define wxSTC_LEX_POWERPRO 95

#define wxSTC_LEX_POWERSHELL 88

#define wxSTC_LEX_PROGRESS 83

#define wxSTC_LEX_PROPERTIES 9

#define wxSTC_LEX_PS 42

#define wxSTC_LEX_PUREBASIC 67

#define wxSTC_LEX_PYTHON 2

#define wxSTC_LEX_R 86

#define wxSTC_LEX_REBOL 71

#define wxSTC_LEX_RUBY 22

#define wxSTC_LEX_RUST 111

#define wxSTC_LEX_SCRIPTOL 33

#define wxSTC_LEX_SMALLTALK 72

#define wxSTC_LEX_SML 97

#define wxSTC_LEX_SORCUS 94

#define wxSTC_LEX_SPECMAN 59

#define wxSTC_LEX_SPICE 78

#define wxSTC_LEX_SQL 7

#define wxSTC_LEX_STTXT 109

#define wxSTC_LEX_TACL 93

#define wxSTC_LEX_TADS3 70

#define wxSTC_LEX_TAL 91

#define wxSTC_LEX_TCL 25

#define wxSTC_LEX_TCMD 103

#define wxSTC_LEX_TEX 49

#define wxSTC_LEX_TXT2TAGS 99

#define wxSTC_LEX_VB 8

#define wxSTC_LEX_VBSCRIPT 28

#define wxSTC_LEX_VERILOG 56

#define wxSTC_LEX_VHDL 64

#define wxSTC_LEX_VISUALPROLOG 107

#define wxSTC_LEX_XCODE 13

#define wxSTC_LEX_XML 5

#define wxSTC_LEX_YAML 48

#define wxSTC_LEXER_START 4000

#define wxSTC_LINE_END_TYPE_DEFAULT 0

Line end types which may be used in addition to LF, CR, and CRLF SC_LINE_END_TYPE_UNICODE includes U+2028 Line Separator, U+2029 Paragraph Separator, and U+0085 Next Line.

#define wxSTC_LINE_END_TYPE_UNICODE 1

#define wxSTC_LISP_COMMENT 1

#define wxSTC_LISP_DEFAULT 0

Lexical states for SCLEX_LISP.

#define wxSTC_LISP_IDENTIFIER 9

#define wxSTC_LISP_KEYWORD 3

#define wxSTC_LISP_KEYWORD_KW 4

#define wxSTC_LISP_MULTI_COMMENT 12

#define wxSTC_LISP_NUMBER 2

#define wxSTC_LISP_OPERATOR 10

#define wxSTC_LISP_SPECIAL 11

#define wxSTC_LISP_STRING 6

#define wxSTC_LISP_STRINGEOL 8

#define wxSTC_LISP_SYMBOL 5

#define wxSTC_LOT_ABORT 6

#define wxSTC_LOT_BREAK 2

#define wxSTC_LOT_DEFAULT 0

Lexical states for SCLEX_LOT.

#define wxSTC_LOT_FAIL 5

#define wxSTC_LOT_HEADER 1

#define wxSTC_LOT_PASS 4

#define wxSTC_LOT_SET 3

#define wxSTC_LOUT_COMMENT 1

#define wxSTC_LOUT_DEFAULT 0

Lexical states for SCLEX_LOUT.

#define wxSTC_LOUT_IDENTIFIER 9

#define wxSTC_LOUT_NUMBER 2

#define wxSTC_LOUT_OPERATOR 8

#define wxSTC_LOUT_STRING 7

#define wxSTC_LOUT_STRINGEOL 10

#define wxSTC_LOUT_WORD 3

#define wxSTC_LOUT_WORD2 4

#define wxSTC_LOUT_WORD3 5

#define wxSTC_LOUT_WORD4 6

#define wxSTC_LUA_CHARACTER 7

#define wxSTC_LUA_COMMENT 1

#define wxSTC_LUA_COMMENTDOC 3

#define wxSTC_LUA_COMMENTLINE 2

#define wxSTC_LUA_DEFAULT 0

Lexical states for SCLEX_LUA.

#define wxSTC_LUA_IDENTIFIER 11

#define wxSTC_LUA_LABEL 20

#define wxSTC_LUA_LITERALSTRING 8

#define wxSTC_LUA_NUMBER 4

#define wxSTC_LUA_OPERATOR 10

#define wxSTC_LUA_PREPROCESSOR 9

#define wxSTC_LUA_STRING 6

#define wxSTC_LUA_STRINGEOL 12

#define wxSTC_LUA_WORD 5

#define wxSTC_LUA_WORD2 13

#define wxSTC_LUA_WORD3 14

#define wxSTC_LUA_WORD4 15

#define wxSTC_LUA_WORD5 16

#define wxSTC_LUA_WORD6 17

#define wxSTC_LUA_WORD7 18

#define wxSTC_LUA_WORD8 19

#define wxSTC_MAGIK_BRACE_BLOCK 10

#define wxSTC_MAGIK_BRACKET_BLOCK 9

#define wxSTC_MAGIK_CHARACTER 3

#define wxSTC_MAGIK_COMMENT 1

#define wxSTC_MAGIK_CONTAINER 8

#define wxSTC_MAGIK_DEFAULT 0

Lexical state for SCLEX_MAGIK.

#define wxSTC_MAGIK_FLOW 7

#define wxSTC_MAGIK_HYPER_COMMENT 16

#define wxSTC_MAGIK_IDENTIFIER 5

#define wxSTC_MAGIK_KEYWORD 13

#define wxSTC_MAGIK_NUMBER 4

#define wxSTC_MAGIK_OPERATOR 6

#define wxSTC_MAGIK_PRAGMA 14

#define wxSTC_MAGIK_SQBRACKET_BLOCK 11

#define wxSTC_MAGIK_STRING 2

#define wxSTC_MAGIK_SYMBOL 15

#define wxSTC_MAGIK_UNKNOWN_KEYWORD 12

#define wxSTC_MAKE_COMMENT 1

#define wxSTC_MAKE_DEFAULT 0

Lexical states for SCLEX_MAKEFILE.

#define wxSTC_MAKE_IDENTIFIER 3

#define wxSTC_MAKE_IDEOL 9

#define wxSTC_MAKE_OPERATOR 4

#define wxSTC_MAKE_PREPROCESSOR 2

#define wxSTC_MAKE_TARGET 5

#define wxSTC_MARGIN_BACK 2

#define wxSTC_MARGIN_FORE 3

#define wxSTC_MARGIN_NUMBER 1

#define wxSTC_MARGIN_RTEXT 5

#define wxSTC_MARGIN_SYMBOL 0

#define wxSTC_MARGIN_TEXT 4

#define wxSTC_MARGINOPTION_NONE 0

#define wxSTC_MARGINOPTION_SUBLINESELECT 1

#define wxSTC_MARK_ARROW 2

#define wxSTC_MARK_ARROWDOWN 6

#define wxSTC_MARK_ARROWS 24

#define wxSTC_MARK_AVAILABLE 28

#define wxSTC_MARK_BACKGROUND 22

Invisible mark that only sets the line background colour.

#define wxSTC_MARK_BOOKMARK 31

#define wxSTC_MARK_BOXMINUS 14

#define wxSTC_MARK_BOXMINUSCONNECTED 15

#define wxSTC_MARK_BOXPLUS 12

#define wxSTC_MARK_BOXPLUSCONNECTED 13

#define wxSTC_MARK_CHARACTER 10000

#define wxSTC_MARK_CIRCLE 0

#define wxSTC_MARK_CIRCLEMINUS 20

#define wxSTC_MARK_CIRCLEMINUSCONNECTED 21

#define wxSTC_MARK_CIRCLEPLUS 18

#define wxSTC_MARK_CIRCLEPLUSCONNECTED 19

#define wxSTC_MARK_DOTDOTDOT 23

#define wxSTC_MARK_EMPTY 5

#define wxSTC_MARK_FULLRECT 26

#define wxSTC_MARK_LCORNER 10

#define wxSTC_MARK_LCORNERCURVE 16

#define wxSTC_MARK_LEFTRECT 27

#define wxSTC_MARK_MINUS 7

#define wxSTC_MARK_PIXMAP 25

#define wxSTC_MARK_PLUS 8

#define wxSTC_MARK_RGBAIMAGE 30

#define wxSTC_MARK_ROUNDRECT 1

#define wxSTC_MARK_SHORTARROW 4

#define wxSTC_MARK_SMALLRECT 3

#define wxSTC_MARK_TCORNER 11

#define wxSTC_MARK_TCORNERCURVE 17

#define wxSTC_MARK_UNDERLINE 29

#define wxSTC_MARK_VLINE 9

Shapes used for outlining column.

#define wxSTC_MARKDOWN_BLOCKQUOTE 15

#define wxSTC_MARKDOWN_CODE 19

#define wxSTC_MARKDOWN_CODE2 20

#define wxSTC_MARKDOWN_CODEBK 21

#define wxSTC_MARKDOWN_DEFAULT 0

Lexical state for SCLEX_MARKDOWN.

#define wxSTC_MARKDOWN_EM1 4

#define wxSTC_MARKDOWN_EM2 5

#define wxSTC_MARKDOWN_HEADER1 6

#define wxSTC_MARKDOWN_HEADER2 7

#define wxSTC_MARKDOWN_HEADER3 8

#define wxSTC_MARKDOWN_HEADER4 9

#define wxSTC_MARKDOWN_HEADER5 10

#define wxSTC_MARKDOWN_HEADER6 11

#define wxSTC_MARKDOWN_HRULE 17

#define wxSTC_MARKDOWN_LINE_BEGIN 1

#define wxSTC_MARKDOWN_LINK 18

#define wxSTC_MARKDOWN_OLIST_ITEM 14

#define wxSTC_MARKDOWN_PRECHAR 12

#define wxSTC_MARKDOWN_STRIKEOUT 16

#define wxSTC_MARKDOWN_STRONG1 2

#define wxSTC_MARKDOWN_STRONG2 3

#define wxSTC_MARKDOWN_ULIST_ITEM 13

#define wxSTC_MARKER_MAX 31

#define wxSTC_MARKNUM_FOLDER 30

#define wxSTC_MARKNUM_FOLDEREND 25

Markers used for outlining column.

#define wxSTC_MARKNUM_FOLDERMIDTAIL 27

#define wxSTC_MARKNUM_FOLDEROPEN 31

#define wxSTC_MARKNUM_FOLDEROPENMID 26

#define wxSTC_MARKNUM_FOLDERSUB 29

#define wxSTC_MARKNUM_FOLDERTAIL 28

#define wxSTC_MASK_FOLDERS 0xFE000000

#define wxSTC_MATLAB_COMMAND 2

#define wxSTC_MATLAB_COMMENT 1

#define wxSTC_MATLAB_DEFAULT 0

Lexical states for SCLEX_MATLAB.

#define wxSTC_MATLAB_DOUBLEQUOTESTRING 8

#define wxSTC_MATLAB_IDENTIFIER 7

#define wxSTC_MATLAB_KEYWORD 4

#define wxSTC_MATLAB_NUMBER 3

#define wxSTC_MATLAB_OPERATOR 6

#define wxSTC_MATLAB_STRING 5

single quoted string

#define wxSTC_MAX_MARGIN 4

#define wxSTC_METAPOST_COMMAND 4

#define wxSTC_METAPOST_DEFAULT 0

#define wxSTC_METAPOST_EXTRA 6

#define wxSTC_METAPOST_GROUP 2

#define wxSTC_METAPOST_SPECIAL 1

#define wxSTC_METAPOST_SYMBOL 3

#define wxSTC_METAPOST_TEXT 5

#define wxSTC_MMIXAL_CHAR 11

#define wxSTC_MMIXAL_COMMENT 1

#define wxSTC_MMIXAL_HEX 14

#define wxSTC_MMIXAL_INCLUDE 17

#define wxSTC_MMIXAL_LABEL 2

#define wxSTC_MMIXAL_LEADWS 0

Lexical states for SCLEX_MMIXAL.

#define wxSTC_MMIXAL_NUMBER 9

#define wxSTC_MMIXAL_OPCODE 3

#define wxSTC_MMIXAL_OPCODE_POST 7

#define wxSTC_MMIXAL_OPCODE_PRE 4

#define wxSTC_MMIXAL_OPCODE_UNKNOWN 6

#define wxSTC_MMIXAL_OPCODE_VALID 5

#define wxSTC_MMIXAL_OPERANDS 8

#define wxSTC_MMIXAL_OPERATOR 15

#define wxSTC_MMIXAL_REF 10

#define wxSTC_MMIXAL_REGISTER 13

#define wxSTC_MMIXAL_STRING 12

#define wxSTC_MMIXAL_SYMBOL 16

#define wxSTC_MOD_BEFOREDELETE 0x800

#define wxSTC_MOD_BEFOREINSERT 0x400

#define wxSTC_MOD_CHANGEANNOTATION 0x20000

#define wxSTC_MOD_CHANGEFOLD 0x8

#define wxSTC_MOD_CHANGEINDICATOR 0x4000

#define wxSTC_MOD_CHANGELINESTATE 0x8000

#define wxSTC_MOD_CHANGEMARGIN 0x10000

#define wxSTC_MOD_CHANGEMARKER 0x200

#define wxSTC_MOD_CHANGESTYLE 0x4

#define wxSTC_MOD_CONTAINER 0x40000

#define wxSTC_MOD_DELETETEXT 0x2

#define wxSTC_MOD_INSERTTEXT 0x1

Notifications Type of modification and the action which caused the modification.

These are defined as a bit mask to make it easy to specify which notifications are wanted. One bit is set from each of SC_MOD_∗ and SC_PERFORMED_∗.

#define wxSTC_MOD_LEXERSTATE 0x80000

#define wxSTC_MODEVENTMASKALL 0xFFFFF

#define wxSTC_MODULA_BADSTR 17

#define wxSTC_MODULA_BASENUM 7

#define wxSTC_MODULA_CHAR 11

#define wxSTC_MODULA_CHARSPEC 12

#define wxSTC_MODULA_COMMENT 1

#define wxSTC_MODULA_DEFAULT 0

Lexical states for SCLEX_MODULA.

#define wxSTC_MODULA_DOXYCOMM 2

#define wxSTC_MODULA_DOXYKEY 3

#define wxSTC_MODULA_FLOAT 8

#define wxSTC_MODULA_KEYWORD 4

#define wxSTC_MODULA_NUMBER 6

#define wxSTC_MODULA_OPERATOR 16

#define wxSTC_MODULA_PRAGMA 14

#define wxSTC_MODULA_PRGKEY 15

#define wxSTC_MODULA_PROC 13

#define wxSTC_MODULA_RESERVED 5

#define wxSTC_MODULA_STRING 9

#define wxSTC_MODULA_STRSPEC 10

#define wxSTC_MSSQL_COLUMN_NAME 8

#define wxSTC_MSSQL_COLUMN_NAME_2 16

#define wxSTC_MSSQL_COMMENT 1

#define wxSTC_MSSQL_DATATYPE 10

#define wxSTC_MSSQL_DEFAULT 0

Lexical states for SCLEX_OCTAVE are identical to MatLab Lexical states for SCLEX_MSSQL.

#define wxSTC_MSSQL_DEFAULT_PREF_DATATYPE 15

#define wxSTC_MSSQL_FUNCTION 13

#define wxSTC_MSSQL_GLOBAL_VARIABLE 12

#define wxSTC_MSSQL_IDENTIFIER 6

#define wxSTC_MSSQL_LINE_COMMENT 2

#define wxSTC_MSSQL_NUMBER 3

#define wxSTC_MSSQL_OPERATOR 5

#define wxSTC_MSSQL_STATEMENT 9

#define wxSTC_MSSQL_STORED_PROCEDURE 14

#define wxSTC_MSSQL_STRING 4

#define wxSTC_MSSQL_SYSTABLE 11

#define wxSTC_MSSQL_VARIABLE 7

#define wxSTC_MULTILINEUNDOREDO 0x1000

#define wxSTC_MULTIPASTE_EACH 1

#define wxSTC_MULTIPASTE_ONCE 0

#define wxSTC_MULTISTEPUNDOREDO 0x80

#define wxSTC_MYSQL_COMMENT 1

#define wxSTC_MYSQL_COMMENTLINE 2

#define wxSTC_MYSQL_DATABASEOBJECT 9

#define wxSTC_MYSQL_DEFAULT 0

Lexical state for SCLEX_MYSQL.

#define wxSTC_MYSQL_DQSTRING 13

#define wxSTC_MYSQL_FUNCTION 15

#define wxSTC_MYSQL_HIDDENCOMMAND 21

#define wxSTC_MYSQL_IDENTIFIER 16

#define wxSTC_MYSQL_KEYWORD 8

#define wxSTC_MYSQL_KNOWNSYSTEMVARIABLE 5

#define wxSTC_MYSQL_MAJORKEYWORD 7

#define wxSTC_MYSQL_NUMBER 6

#define wxSTC_MYSQL_OPERATOR 14

#define wxSTC_MYSQL_PLACEHOLDER 22

#define wxSTC_MYSQL_PROCEDUREKEYWORD 10

#define wxSTC_MYSQL_QUOTEDIDENTIFIER 17

#define wxSTC_MYSQL_SQSTRING 12

#define wxSTC_MYSQL_STRING 11

#define wxSTC_MYSQL_SYSTEMVARIABLE 4

#define wxSTC_MYSQL_USER1 18

#define wxSTC_MYSQL_USER2 19

#define wxSTC_MYSQL_USER3 20

#define wxSTC_MYSQL_VARIABLE 3

#define wxSTC_NNCRONTAB_ASTERISK 6

#define wxSTC_NNCRONTAB_COMMENT 1

#define wxSTC_NNCRONTAB_DEFAULT 0

Lexical states for SCLEX_NNCRONTAB (nnCron crontab Lexer)

#define wxSTC_NNCRONTAB_ENVIRONMENT 9

#define wxSTC_NNCRONTAB_IDENTIFIER 10

#define wxSTC_NNCRONTAB_KEYWORD 4

#define wxSTC_NNCRONTAB_MODIFIER 5

#define wxSTC_NNCRONTAB_NUMBER 7

#define wxSTC_NNCRONTAB_SECTION 3

#define wxSTC_NNCRONTAB_STRING 8

#define wxSTC_NNCRONTAB_TASK 2

#define wxSTC_NSIS_COMMENT 1

#define wxSTC_NSIS_COMMENTBOX 18

#define wxSTC_NSIS_DEFAULT 0

Lexical states for SCLEX_NSIS.

#define wxSTC_NSIS_FUNCTION 5

#define wxSTC_NSIS_FUNCTIONDEF 17

#define wxSTC_NSIS_IFDEFINEDEF 11

#define wxSTC_NSIS_LABEL 7

#define wxSTC_NSIS_MACRODEF 12

#define wxSTC_NSIS_NUMBER 14

#define wxSTC_NSIS_PAGEEX 16

#define wxSTC_NSIS_SECTIONDEF 9

#define wxSTC_NSIS_SECTIONGROUP 15

#define wxSTC_NSIS_STRINGDQ 2

#define wxSTC_NSIS_STRINGLQ 3

#define wxSTC_NSIS_STRINGRQ 4

#define wxSTC_NSIS_STRINGVAR 13

#define wxSTC_NSIS_SUBSECTIONDEF 10

#define wxSTC_NSIS_USERDEFINED 8

#define wxSTC_NSIS_VARIABLE 6

#define wxSTC_OPAL_BOOL_CONST 8

#define wxSTC_OPAL_COMMENT_BLOCK 1

#define wxSTC_OPAL_COMMENT_LINE 2

#define wxSTC_OPAL_DEFAULT 32

#define wxSTC_OPAL_INTEGER 3

#define wxSTC_OPAL_KEYWORD 4

#define wxSTC_OPAL_PAR 7

#define wxSTC_OPAL_SORT 5

#define wxSTC_OPAL_SPACE 0

Lexical states for SCLEX_OPAL.

#define wxSTC_OPAL_STRING 6

#define wxSTC_OPTIONAL_START 3000

#define wxSTC_ORDER_CUSTOM 2

#define wxSTC_ORDER_PERFORMSORT 1

#define wxSTC_ORDER_PRESORTED 0

#define wxSTC_OSCRIPT_BLOCK_COMMENT 2

#define wxSTC_OSCRIPT_CONSTANT 8

#define wxSTC_OSCRIPT_DEFAULT 0

Lexical states for SCLEX_OSCRIPT.

#define wxSTC_OSCRIPT_DOC_COMMENT 3

#define wxSTC_OSCRIPT_DOUBLEQUOTE_STRING 7

#define wxSTC_OSCRIPT_FUNCTION 15

#define wxSTC_OSCRIPT_GLOBAL 10

#define wxSTC_OSCRIPT_IDENTIFIER 9

#define wxSTC_OSCRIPT_KEYWORD 11

#define wxSTC_OSCRIPT_LABEL 13

#define wxSTC_OSCRIPT_LINE_COMMENT 1

#define wxSTC_OSCRIPT_METHOD 18

#define wxSTC_OSCRIPT_NUMBER 5

#define wxSTC_OSCRIPT_OBJECT 16

#define wxSTC_OSCRIPT_OPERATOR 12

#define wxSTC_OSCRIPT_PREPROCESSOR 4

#define wxSTC_OSCRIPT_PROPERTY 17

#define wxSTC_OSCRIPT_SINGLEQUOTE_STRING 6

#define wxSTC_OSCRIPT_TYPE 14

#define wxSTC_P_CHARACTER 4

#define wxSTC_P_CLASSNAME 8

#define wxSTC_P_COMMENTBLOCK 12

#define wxSTC_P_COMMENTLINE 1

#define wxSTC_P_DECORATOR 15

#define wxSTC_P_DEFAULT 0

Lexical states for SCLEX_PYTHON.

#define wxSTC_P_DEFNAME 9

#define wxSTC_P_IDENTIFIER 11

#define wxSTC_P_NUMBER 2

#define wxSTC_P_OPERATOR 10

#define wxSTC_P_STRING 3

#define wxSTC_P_STRINGEOL 13

#define wxSTC_P_TRIPLE 6

#define wxSTC_P_TRIPLEDOUBLE 7

#define wxSTC_P_WORD 5

#define wxSTC_P_WORD2 14

#define wxSTC_PAS_ASM 14

#define wxSTC_PAS_CHARACTER 12

#define wxSTC_PAS_COMMENT 2

#define wxSTC_PAS_COMMENT2 3

#define wxSTC_PAS_COMMENTLINE 4

#define wxSTC_PAS_DEFAULT 0

Lexical states for SCLEX_PASCAL.

#define wxSTC_PAS_HEXNUMBER 8

#define wxSTC_PAS_IDENTIFIER 1

#define wxSTC_PAS_NUMBER 7

#define wxSTC_PAS_OPERATOR 13

#define wxSTC_PAS_PREPROCESSOR 5

#define wxSTC_PAS_PREPROCESSOR2 6

#define wxSTC_PAS_STRING 10

#define wxSTC_PAS_STRINGEOL 11

#define wxSTC_PAS_WORD 9

#define wxSTC_PERFORMED_REDO 0x40

#define wxSTC_PERFORMED_UNDO 0x20

#define wxSTC_PERFORMED_USER 0x10

#define wxSTC_PL_ARRAY 13

#define wxSTC_PL_BACKTICKS 20

#define wxSTC_PL_BACKTICKS_VAR 57

#define wxSTC_PL_CHARACTER 7

#define wxSTC_PL_COMMENTLINE 2

#define wxSTC_PL_DATASECTION 21

#define wxSTC_PL_DEFAULT 0

Lexical states for SCLEX_PERL.

#define wxSTC_PL_ERROR 1

#define wxSTC_PL_FORMAT 42

#define wxSTC_PL_FORMAT_IDENT 41

#define wxSTC_PL_HASH 14

#define wxSTC_PL_HERE_DELIM 22

#define wxSTC_PL_HERE_Q 23

#define wxSTC_PL_HERE_QQ 24

#define wxSTC_PL_HERE_QQ_VAR 61

#define wxSTC_PL_HERE_QX 25

#define wxSTC_PL_HERE_QX_VAR 62

#define wxSTC_PL_IDENTIFIER 11

#define wxSTC_PL_LONGQUOTE 19

#define wxSTC_PL_NUMBER 4

#define wxSTC_PL_OPERATOR 10

#define wxSTC_PL_POD 3

#define wxSTC_PL_POD_VERB 31

#define wxSTC_PL_PREPROCESSOR 9

#define wxSTC_PL_PUNCTUATION 8

#define wxSTC_PL_REGEX 17

#define wxSTC_PL_REGEX_VAR 54

#define wxSTC_PL_REGSUBST 18

#define wxSTC_PL_REGSUBST_VAR 55

#define wxSTC_PL_SCALAR 12

#define wxSTC_PL_STRING 6

#define wxSTC_PL_STRING_Q 26

#define wxSTC_PL_STRING_QQ 27

#define wxSTC_PL_STRING_QQ_VAR 64

#define wxSTC_PL_STRING_QR 29

#define wxSTC_PL_STRING_QR_VAR 66

#define wxSTC_PL_STRING_QW 30

#define wxSTC_PL_STRING_QX 28

#define wxSTC_PL_STRING_QX_VAR 65

#define wxSTC_PL_STRING_VAR 43

#define wxSTC_PL_SUB_PROTOTYPE 40

#define wxSTC_PL_SYMBOLTABLE 15

#define wxSTC_PL_VARIABLE_INDEXER 16

#define wxSTC_PL_WORD 5

#define wxSTC_PL_XLAT 44

#define wxSTC_PLM_COMMENT 1

#define wxSTC_PLM_CONTROL 6

#define wxSTC_PLM_DEFAULT 0

Lexical state for SCLEX_PLM.

#define wxSTC_PLM_IDENTIFIER 4

#define wxSTC_PLM_KEYWORD 7

#define wxSTC_PLM_NUMBER 3

#define wxSTC_PLM_OPERATOR 5

#define wxSTC_PLM_STRING 2

#define wxSTC_PO_COMMENT 1

#define wxSTC_PO_DEFAULT 0

Lexical state for SCLEX_PO.

#define wxSTC_PO_ERROR 15

#define wxSTC_PO_FLAGS 11

#define wxSTC_PO_FUZZY 8

#define wxSTC_PO_MSGCTXT 6

#define wxSTC_PO_MSGCTXT_TEXT 7

#define wxSTC_PO_MSGCTXT_TEXT_EOL 14

#define wxSTC_PO_MSGID 2

#define wxSTC_PO_MSGID_TEXT 3

#define wxSTC_PO_MSGID_TEXT_EOL 12

#define wxSTC_PO_MSGSTR 4

#define wxSTC_PO_MSGSTR_TEXT 5

#define wxSTC_PO_MSGSTR_TEXT_EOL 13

#define wxSTC_PO_PROGRAMMER_COMMENT 9

#define wxSTC_PO_REFERENCE 10

#define wxSTC_POV_BADDIRECTIVE 9

#define wxSTC_POV_COMMENT 1

#define wxSTC_POV_COMMENTLINE 2

#define wxSTC_POV_DEFAULT 0

Lexical states for SCLEX_POV.

#define wxSTC_POV_DIRECTIVE 8

#define wxSTC_POV_IDENTIFIER 5

#define wxSTC_POV_NUMBER 3

#define wxSTC_POV_OPERATOR 4

#define wxSTC_POV_STRING 6

#define wxSTC_POV_STRINGEOL 7

#define wxSTC_POV_WORD2 10

#define wxSTC_POV_WORD3 11

#define wxSTC_POV_WORD4 12

#define wxSTC_POV_WORD5 13

#define wxSTC_POV_WORD6 14

#define wxSTC_POV_WORD7 15

#define wxSTC_POV_WORD8 16

#define wxSTC_POWERPRO_ALTQUOTE 15

#define wxSTC_POWERPRO_COMMENTBLOCK 1

#define wxSTC_POWERPRO_COMMENTLINE 2

#define wxSTC_POWERPRO_DEFAULT 0

Lexical state for SCLEX_POWERPRO.

#define wxSTC_POWERPRO_DOUBLEQUOTEDSTRING 8

#define wxSTC_POWERPRO_FUNCTION 16

#define wxSTC_POWERPRO_IDENTIFIER 12

#define wxSTC_POWERPRO_LINECONTINUE 10

#define wxSTC_POWERPRO_NUMBER 3

#define wxSTC_POWERPRO_OPERATOR 11

#define wxSTC_POWERPRO_SINGLEQUOTEDSTRING 9

#define wxSTC_POWERPRO_STRINGEOL 13

#define wxSTC_POWERPRO_VERBATIM 14

#define wxSTC_POWERPRO_WORD 4

#define wxSTC_POWERPRO_WORD2 5

#define wxSTC_POWERPRO_WORD3 6

#define wxSTC_POWERPRO_WORD4 7

#define wxSTC_POWERSHELL_ALIAS 10

#define wxSTC_POWERSHELL_CHARACTER 3

#define wxSTC_POWERSHELL_CMDLET 9

#define wxSTC_POWERSHELL_COMMENT 1

#define wxSTC_POWERSHELL_COMMENTDOCKEYWORD 16

#define wxSTC_POWERSHELL_COMMENTSTREAM 13

#define wxSTC_POWERSHELL_DEFAULT 0

Lexical state for SCLEX_POWERSHELL.

#define wxSTC_POWERSHELL_FUNCTION 11

#define wxSTC_POWERSHELL_HERE_CHARACTER 15

#define wxSTC_POWERSHELL_HERE_STRING 14

#define wxSTC_POWERSHELL_IDENTIFIER 7

#define wxSTC_POWERSHELL_KEYWORD 8

#define wxSTC_POWERSHELL_NUMBER 4

#define wxSTC_POWERSHELL_OPERATOR 6

#define wxSTC_POWERSHELL_STRING 2

#define wxSTC_POWERSHELL_USER1 12

#define wxSTC_POWERSHELL_VARIABLE 5

#define wxSTC_PRINT_BLACKONWHITE 2

PrintColourMode - force black text on white background for printing.

#define wxSTC_PRINT_COLOURONWHITE 3

PrintColourMode - text stays coloured, but all background is forced to be white for printing.

#define wxSTC_PRINT_COLOURONWHITEDEFAULTBG 4

PrintColourMode - only the default-background is forced to be white for printing.

#define wxSTC_PRINT_INVERTLIGHT 1

PrintColourMode - invert the light value of each style for printing.

#define wxSTC_PRINT_NORMAL 0

PrintColourMode - use same colours as screen.

#define wxSTC_PROPS_ASSIGNMENT 3

#define wxSTC_PROPS_COMMENT 1

#define wxSTC_PROPS_DEFAULT 0

Lexical states for SCLEX_PROPERTIES.

#define wxSTC_PROPS_DEFVAL 4

#define wxSTC_PROPS_KEY 5

#define wxSTC_PROPS_SECTION 2

#define wxSTC_PS_BADSTRINGCHAR 15

#define wxSTC_PS_BASE85STRING 14

#define wxSTC_PS_COMMENT 1

#define wxSTC_PS_DEFAULT 0

Lexical states for SCLEX_PS.

#define wxSTC_PS_DSC_COMMENT 2

#define wxSTC_PS_DSC_VALUE 3

#define wxSTC_PS_HEXSTRING 13

#define wxSTC_PS_IMMEVAL 8

#define wxSTC_PS_KEYWORD 6

#define wxSTC_PS_LITERAL 7

#define wxSTC_PS_NAME 5

#define wxSTC_PS_NUMBER 4

#define wxSTC_PS_PAREN_ARRAY 9

#define wxSTC_PS_PAREN_DICT 10

#define wxSTC_PS_PAREN_PROC 11

#define wxSTC_PS_TEXT 12

#define wxSTC_R_BASEKWORD 3

#define wxSTC_R_COMMENT 1

#define wxSTC_R_DEFAULT 0

Lexical states for SCLEX_R.

#define wxSTC_R_IDENTIFIER 9

#define wxSTC_R_INFIX 10

#define wxSTC_R_INFIXEOL 11

#define wxSTC_R_KWORD 2

#define wxSTC_R_NUMBER 5

#define wxSTC_R_OPERATOR 8

#define wxSTC_R_OTHERKWORD 4

#define wxSTC_R_STRING 6

#define wxSTC_R_STRING2 7

#define wxSTC_RB_BACKTICKS 18

#define wxSTC_RB_CHARACTER 7

#define wxSTC_RB_CLASS_VAR 17

#define wxSTC_RB_CLASSNAME 8

#define wxSTC_RB_COMMENTLINE 2

#define wxSTC_RB_DATASECTION 19

#define wxSTC_RB_DEFAULT 0

Lexical states for SCLEX_RUBY.

#define wxSTC_RB_DEFNAME 9

#define wxSTC_RB_ERROR 1

#define wxSTC_RB_GLOBAL 13

#define wxSTC_RB_HERE_DELIM 20

#define wxSTC_RB_HERE_Q 21

#define wxSTC_RB_HERE_QQ 22

#define wxSTC_RB_HERE_QX 23

#define wxSTC_RB_IDENTIFIER 11

#define wxSTC_RB_INSTANCE_VAR 16

#define wxSTC_RB_MODULE_NAME 15

#define wxSTC_RB_NUMBER 4

#define wxSTC_RB_OPERATOR 10

#define wxSTC_RB_POD 3

#define wxSTC_RB_REGEX 12

#define wxSTC_RB_STDERR 40

#define wxSTC_RB_STDIN 30

#define wxSTC_RB_STDOUT 31

#define wxSTC_RB_STRING 6

#define wxSTC_RB_STRING_Q 24

#define wxSTC_RB_STRING_QQ 25

#define wxSTC_RB_STRING_QR 27

#define wxSTC_RB_STRING_QW 28

#define wxSTC_RB_STRING_QX 26

#define wxSTC_RB_SYMBOL 14

#define wxSTC_RB_UPPER_BOUND 41

#define wxSTC_RB_WORD 5

#define wxSTC_RB_WORD_DEMOTED 29

#define wxSTC_REBOL_BINARY 11

#define wxSTC_REBOL_BRACEDSTRING 7

#define wxSTC_REBOL_CHARACTER 5

#define wxSTC_REBOL_COMMENTBLOCK 2

#define wxSTC_REBOL_COMMENTLINE 1

#define wxSTC_REBOL_DATE 18

#define wxSTC_REBOL_DEFAULT 0

Lexical states for SCLEX_REBOL.

#define wxSTC_REBOL_EMAIL 16

#define wxSTC_REBOL_FILE 15

#define wxSTC_REBOL_IDENTIFIER 20

#define wxSTC_REBOL_ISSUE 13

#define wxSTC_REBOL_MONEY 12

#define wxSTC_REBOL_NUMBER 8

#define wxSTC_REBOL_OPERATOR 4

#define wxSTC_REBOL_PAIR 9

#define wxSTC_REBOL_PREFACE 3

#define wxSTC_REBOL_QUOTEDSTRING 6

#define wxSTC_REBOL_TAG 14

#define wxSTC_REBOL_TIME 19

#define wxSTC_REBOL_TUPLE 10

#define wxSTC_REBOL_URL 17

#define wxSTC_REBOL_WORD 21

#define wxSTC_REBOL_WORD2 22

#define wxSTC_REBOL_WORD3 23

#define wxSTC_REBOL_WORD4 24

#define wxSTC_REBOL_WORD5 25

#define wxSTC_REBOL_WORD6 26

#define wxSTC_REBOL_WORD7 27

#define wxSTC_REBOL_WORD8 28

#define wxSTC_RUST_CHARACTER 15

#define wxSTC_RUST_COMMENTBLOCK 1

#define wxSTC_RUST_COMMENTBLOCKDOC 3

#define wxSTC_RUST_COMMENTLINE 2

#define wxSTC_RUST_COMMENTLINEDOC 4

#define wxSTC_RUST_DEFAULT 0

Lexical states for SCLEX_RUST.

#define wxSTC_RUST_IDENTIFIER 17

#define wxSTC_RUST_LEXERROR 20

#define wxSTC_RUST_LIFETIME 18

#define wxSTC_RUST_MACRO 19

#define wxSTC_RUST_NUMBER 5

#define wxSTC_RUST_OPERATOR 16

#define wxSTC_RUST_STRING 13

#define wxSTC_RUST_STRINGR 14

#define wxSTC_RUST_WORD 6

#define wxSTC_RUST_WORD2 7

#define wxSTC_RUST_WORD3 8

#define wxSTC_RUST_WORD4 9

#define wxSTC_RUST_WORD5 10

#define wxSTC_RUST_WORD6 11

#define wxSTC_RUST_WORD7 12

#define wxSTC_SCMOD_ALT 4

#define wxSTC_SCMOD_CTRL 2

#define wxSTC_SCMOD_META 16

#define wxSTC_SCMOD_NORM 0

#define wxSTC_SCMOD_SHIFT 1

#define wxSTC_SCMOD_SUPER 8

#define wxSTC_SCRIPTOL_CHARACTER 8

#define wxSTC_SCRIPTOL_CLASSNAME 14

#define wxSTC_SCRIPTOL_COMMENTBLOCK 5

#define wxSTC_SCRIPTOL_COMMENTLINE 2

#define wxSTC_SCRIPTOL_CSTYLE 4

#define wxSTC_SCRIPTOL_DEFAULT 0

Lexical states for SCLEX_SCRIPTOL.

#define wxSTC_SCRIPTOL_IDENTIFIER 12

#define wxSTC_SCRIPTOL_KEYWORD 10

#define wxSTC_SCRIPTOL_NUMBER 6

#define wxSTC_SCRIPTOL_OPERATOR 11

#define wxSTC_SCRIPTOL_PERSISTENT 3

#define wxSTC_SCRIPTOL_PREPROCESSOR 15

#define wxSTC_SCRIPTOL_STRING 7

#define wxSTC_SCRIPTOL_STRINGEOL 9

#define wxSTC_SCRIPTOL_TRIPLE 13

#define wxSTC_SCRIPTOL_WHITE 1

#define wxSTC_SCVS_NONE 0

#define wxSTC_SCVS_RECTANGULARSELECTION 1

#define wxSTC_SCVS_USERACCESSIBLE 2

#define wxSTC_SEL_LINES 2

#define wxSTC_SEL_RECTANGLE 1

#define wxSTC_SEL_STREAM 0

#define wxSTC_SEL_THIN 3

#define wxSTC_SH_BACKTICKS 11

#define wxSTC_SH_CHARACTER 6

#define wxSTC_SH_COMMENTLINE 2

#define wxSTC_SH_DEFAULT 0

Lexical states for SCLEX_BASH.

#define wxSTC_SH_ERROR 1

#define wxSTC_SH_HERE_DELIM 12

#define wxSTC_SH_HERE_Q 13

#define wxSTC_SH_IDENTIFIER 8

#define wxSTC_SH_NUMBER 3

#define wxSTC_SH_OPERATOR 7

#define wxSTC_SH_PARAM 10

#define wxSTC_SH_SCALAR 9

#define wxSTC_SH_STRING 5

#define wxSTC_SH_WORD 4

#define wxSTC_SML_CHAR 9

#define wxSTC_SML_COMMENT 12

#define wxSTC_SML_COMMENT1 13

#define wxSTC_SML_COMMENT2 14

#define wxSTC_SML_COMMENT3 15

#define wxSTC_SML_DEFAULT 0

Lexical states for SCLEX_SML.

#define wxSTC_SML_IDENTIFIER 1

#define wxSTC_SML_KEYWORD 3

#define wxSTC_SML_KEYWORD2 4

#define wxSTC_SML_KEYWORD3 5

#define wxSTC_SML_LINENUM 6

#define wxSTC_SML_NUMBER 8

#define wxSTC_SML_OPERATOR 7

#define wxSTC_SML_STRING 11

#define wxSTC_SML_TAGNAME 2

#define wxSTC_SN_CODE 1

#define wxSTC_SN_COMMENTLINE 2

#define wxSTC_SN_COMMENTLINEBANG 3

#define wxSTC_SN_DEFAULT 0

Lexical states for SCLEX_SPECMAN.

#define wxSTC_SN_IDENTIFIER 11

#define wxSTC_SN_NUMBER 4

#define wxSTC_SN_OPERATOR 10

#define wxSTC_SN_PREPROCESSOR 9

#define wxSTC_SN_REGEXTAG 13

#define wxSTC_SN_SIGNAL 14

#define wxSTC_SN_STRING 6

#define wxSTC_SN_STRINGEOL 12

#define wxSTC_SN_USER 19

#define wxSTC_SN_WORD 5

#define wxSTC_SN_WORD2 7

#define wxSTC_SN_WORD3 8

#define wxSTC_SORCUS_COMMAND 1

#define wxSTC_SORCUS_COMMENTLINE 3

#define wxSTC_SORCUS_CONSTANT 9

#define wxSTC_SORCUS_DEFAULT 0

Lexical state for SCLEX_SORCUS.

#define wxSTC_SORCUS_IDENTIFIER 6

#define wxSTC_SORCUS_NUMBER 8

#define wxSTC_SORCUS_OPERATOR 7

#define wxSTC_SORCUS_PARAMETER 2

#define wxSTC_SORCUS_STRING 4

#define wxSTC_SORCUS_STRINGEOL 5

#define wxSTC_SPICE_COMMENTLINE 8

#define wxSTC_SPICE_DEFAULT 0

Lexical states for SCLEX_SPICE.

#define wxSTC_SPICE_DELIMITER 6

#define wxSTC_SPICE_IDENTIFIER 1

#define wxSTC_SPICE_KEYWORD 2

#define wxSTC_SPICE_KEYWORD2 3

#define wxSTC_SPICE_KEYWORD3 4

#define wxSTC_SPICE_NUMBER 5

#define wxSTC_SPICE_VALUE 7

#define wxSTC_SQL_CHARACTER 7

#define wxSTC_SQL_COMMENT 1

#define wxSTC_SQL_COMMENTDOC 3

#define wxSTC_SQL_COMMENTDOCKEYWORD 17

#define wxSTC_SQL_COMMENTDOCKEYWORDERROR 18

#define wxSTC_SQL_COMMENTLINE 2

#define wxSTC_SQL_COMMENTLINEDOC 15

#define wxSTC_SQL_DEFAULT 0

Lexical states for SCLEX_SQL.

#define wxSTC_SQL_IDENTIFIER 11

#define wxSTC_SQL_NUMBER 4

#define wxSTC_SQL_OPERATOR 10

#define wxSTC_SQL_QUOTEDIDENTIFIER 23

#define wxSTC_SQL_SQLPLUS 8

#define wxSTC_SQL_SQLPLUS_COMMENT 13

#define wxSTC_SQL_SQLPLUS_PROMPT 9

#define wxSTC_SQL_STRING 6

#define wxSTC_SQL_USER1 19

#define wxSTC_SQL_USER2 20

#define wxSTC_SQL_USER3 21

#define wxSTC_SQL_USER4 22

#define wxSTC_SQL_WORD 5

#define wxSTC_SQL_WORD2 16

#define wxSTC_ST_ASSIGN 14

#define wxSTC_ST_BINARY 5

#define wxSTC_ST_BOOL 6

#define wxSTC_ST_CHARACTER 15

#define wxSTC_ST_COMMENT 3

#define wxSTC_ST_DEFAULT 0

Lexical states for SCLEX_SMALLTALK.

#define wxSTC_ST_GLOBAL 10

#define wxSTC_ST_KWSEND 13

#define wxSTC_ST_NIL 9

#define wxSTC_ST_NUMBER 2

#define wxSTC_ST_RETURN 11

#define wxSTC_ST_SELF 7

#define wxSTC_ST_SPEC_SEL 16

#define wxSTC_ST_SPECIAL 12

#define wxSTC_ST_STRING 1

#define wxSTC_ST_SUPER 8

#define wxSTC_ST_SYMBOL 4

#define wxSTC_START 2000

Define start of Scintilla messages to be greater than all Windows edit (EM_∗) messages as many EM_ messages can be used although that use is deprecated.

#define wxSTC_STARTACTION 0x2000

#define wxSTC_STATUS_BADALLOC 2

#define wxSTC_STATUS_FAILURE 1

#define wxSTC_STATUS_OK 0

#define wxSTC_STTXT_CHARACTER 11

#define wxSTC_STTXT_COMMENT 1

#define wxSTC_STTXT_COMMENTLINE 2

#define wxSTC_STTXT_DATETIME 16

#define wxSTC_STTXT_DEFAULT 0

Lexical states for SCLEX_STTXT.

#define wxSTC_STTXT_FB 6

#define wxSTC_STTXT_FUNCTION 5

#define wxSTC_STTXT_HEXNUMBER 8

#define wxSTC_STTXT_IDENTIFIER 15

#define wxSTC_STTXT_KEYWORD 3

#define wxSTC_STTXT_NUMBER 7

#define wxSTC_STTXT_OPERATOR 10

#define wxSTC_STTXT_PRAGMA 9

#define wxSTC_STTXT_PRAGMAS 18

#define wxSTC_STTXT_STRING1 12

#define wxSTC_STTXT_STRING2 13

#define wxSTC_STTXT_STRINGEOL 14

#define wxSTC_STTXT_TYPE 4

#define wxSTC_STTXT_VARS 17

#define wxSTC_STYLE_BRACEBAD 35

#define wxSTC_STYLE_BRACELIGHT 34

#define wxSTC_STYLE_CALLTIP 38

#define wxSTC_STYLE_CONTROLCHAR 36

#define wxSTC_STYLE_DEFAULT 32

Styles in range 32..38 are predefined for parts of the UI and are not used as normal styles.

Style 39 is for future use.

#define wxSTC_STYLE_INDENTGUIDE 37

#define wxSTC_STYLE_LASTPREDEFINED 39

#define wxSTC_STYLE_LINENUMBER 33

#define wxSTC_STYLE_MAX 255

#define wxSTC_T3_BLOCK_COMMENT 3

#define wxSTC_T3_BRACE 20

#define wxSTC_T3_D_STRING 10

#define wxSTC_T3_DEFAULT 0

Lexical states of SCLEX_TADS3.

#define wxSTC_T3_HTML_DEFAULT 15

#define wxSTC_T3_HTML_STRING 16

#define wxSTC_T3_HTML_TAG 14

#define wxSTC_T3_IDENTIFIER 8

#define wxSTC_T3_KEYWORD 6

#define wxSTC_T3_LIB_DIRECTIVE 12

#define wxSTC_T3_LINE_COMMENT 4

#define wxSTC_T3_MSG_PARAM 13

#define wxSTC_T3_NUMBER 7

#define wxSTC_T3_OPERATOR 5

#define wxSTC_T3_PREPROCESSOR 2

#define wxSTC_T3_S_STRING 9

#define wxSTC_T3_USER1 17

#define wxSTC_T3_USER2 18

#define wxSTC_T3_USER3 19

#define wxSTC_T3_X_DEFAULT 1

#define wxSTC_T3_X_STRING 11

#define wxSTC_TCL_BLOCK_COMMENT 21

#define wxSTC_TCL_COMMENT 1

#define wxSTC_TCL_COMMENT_BOX 20

#define wxSTC_TCL_COMMENTLINE 2

#define wxSTC_TCL_DEFAULT 0

Lexical states for SCLEX_TCL.

#define wxSTC_TCL_EXPAND 11

#define wxSTC_TCL_IDENTIFIER 7

#define wxSTC_TCL_IN_QUOTE 5

#define wxSTC_TCL_MODIFIER 10

#define wxSTC_TCL_NUMBER 3

#define wxSTC_TCL_OPERATOR 6

#define wxSTC_TCL_SUB_BRACE 9

#define wxSTC_TCL_SUBSTITUTION 8

#define wxSTC_TCL_WORD 12

#define wxSTC_TCL_WORD2 13

#define wxSTC_TCL_WORD3 14

#define wxSTC_TCL_WORD4 15

#define wxSTC_TCL_WORD5 16

#define wxSTC_TCL_WORD6 17

#define wxSTC_TCL_WORD7 18

#define wxSTC_TCL_WORD8 19

#define wxSTC_TCL_WORD_IN_QUOTE 4

#define wxSTC_TCMD_CLABEL 10

#define wxSTC_TCMD_COMMAND 5

#define wxSTC_TCMD_COMMENT 1

#define wxSTC_TCMD_DEFAULT 0

Lexical states for SCLEX_TCMD.

#define wxSTC_TCMD_ENVIRONMENT 8

#define wxSTC_TCMD_EXPANSION 9

#define wxSTC_TCMD_HIDE 4

#define wxSTC_TCMD_IDENTIFIER 6

#define wxSTC_TCMD_LABEL 3

#define wxSTC_TCMD_OPERATOR 7

#define wxSTC_TCMD_WORD 2

#define wxSTC_TECHNOLOGY_DEFAULT 0

#define wxSTC_TECHNOLOGY_DIRECTWRITE 1

#define wxSTC_TEX_COMMAND 4

#define wxSTC_TEX_DEFAULT 0

Lexical states for SCLEX_TEX.

#define wxSTC_TEX_GROUP 2

#define wxSTC_TEX_SPECIAL 1

#define wxSTC_TEX_SYMBOL 3

#define wxSTC_TEX_TEXT 5

#define wxSTC_TIME_FOREVER 10000000

#define wxSTC_TXT2TAGS_BLOCKQUOTE 15

#define wxSTC_TXT2TAGS_CODE 19

#define wxSTC_TXT2TAGS_CODE2 20

#define wxSTC_TXT2TAGS_CODEBK 21

#define wxSTC_TXT2TAGS_COMMENT 22

#define wxSTC_TXT2TAGS_DEFAULT 0

Lexical state for SCLEX_TXT2TAGS.

#define wxSTC_TXT2TAGS_EM1 4

#define wxSTC_TXT2TAGS_EM2 5

#define wxSTC_TXT2TAGS_HEADER1 6

#define wxSTC_TXT2TAGS_HEADER2 7

#define wxSTC_TXT2TAGS_HEADER3 8

#define wxSTC_TXT2TAGS_HEADER4 9

#define wxSTC_TXT2TAGS_HEADER5 10

#define wxSTC_TXT2TAGS_HEADER6 11

#define wxSTC_TXT2TAGS_HRULE 17

#define wxSTC_TXT2TAGS_LINE_BEGIN 1

#define wxSTC_TXT2TAGS_LINK 18

#define wxSTC_TXT2TAGS_OLIST_ITEM 14

#define wxSTC_TXT2TAGS_OPTION 23

#define wxSTC_TXT2TAGS_POSTPROC 25

#define wxSTC_TXT2TAGS_PRECHAR 12

#define wxSTC_TXT2TAGS_PREPROC 24

#define wxSTC_TXT2TAGS_STRIKEOUT 16

#define wxSTC_TXT2TAGS_STRONG1 2

#define wxSTC_TXT2TAGS_STRONG2 3

#define wxSTC_TXT2TAGS_ULIST_ITEM 13

#define wxSTC_TYPE_BOOLEAN 0

#define wxSTC_TYPE_INTEGER 1

#define wxSTC_TYPE_STRING 2

#define wxSTC_UNDO_MAY_COALESCE 1

#define wxSTC_UPDATE_CONTENT 0x1

#define wxSTC_UPDATE_H_SCROLL 0x8

#define wxSTC_UPDATE_SELECTION 0x2

#define wxSTC_UPDATE_V_SCROLL 0x4

#define wxSTC_V_COMMENT 1

#define wxSTC_V_COMMENTLINE 2

#define wxSTC_V_COMMENTLINEBANG 3

#define wxSTC_V_DEFAULT 0

Lexical states for SCLEX_VERILOG.

#define wxSTC_V_IDENTIFIER 11

#define wxSTC_V_NUMBER 4

#define wxSTC_V_OPERATOR 10

#define wxSTC_V_PREPROCESSOR 9

#define wxSTC_V_STRING 6

#define wxSTC_V_STRINGEOL 12

#define wxSTC_V_USER 19

#define wxSTC_V_WORD 5

#define wxSTC_V_WORD2 7

#define wxSTC_V_WORD3 8

#define wxSTC_VHDL_ATTRIBUTE 10

#define wxSTC_VHDL_COMMENT 1

#define wxSTC_VHDL_COMMENTLINEBANG 2

#define wxSTC_VHDL_DEFAULT 0

Lexical states for SCLEX_VHDL.

#define wxSTC_VHDL_IDENTIFIER 6

#define wxSTC_VHDL_KEYWORD 8

#define wxSTC_VHDL_NUMBER 3

#define wxSTC_VHDL_OPERATOR 5

#define wxSTC_VHDL_STDFUNCTION 11

#define wxSTC_VHDL_STDOPERATOR 9

#define wxSTC_VHDL_STDPACKAGE 12

#define wxSTC_VHDL_STDTYPE 13

#define wxSTC_VHDL_STRING 4

#define wxSTC_VHDL_STRINGEOL 7

#define wxSTC_VHDL_USERWORD 14

#define wxSTC_VISIBLE_SLOP 0x01

Constants for use with SetVisiblePolicy, similar to SetCaretPolicy.

#define wxSTC_VISIBLE_STRICT 0x04

#define wxSTC_VISUALPROLOG_ANONYMOUS 10

#define wxSTC_VISUALPROLOG_CHARACTER 13

#define wxSTC_VISUALPROLOG_CHARACTER_ESCAPE_ERROR 15

#define wxSTC_VISUALPROLOG_CHARACTER_TOO_MANY 14

#define wxSTC_VISUALPROLOG_COMMENT_BLOCK 4

#define wxSTC_VISUALPROLOG_COMMENT_KEY 6

#define wxSTC_VISUALPROLOG_COMMENT_KEY_ERROR 7

#define wxSTC_VISUALPROLOG_COMMENT_LINE 5

#define wxSTC_VISUALPROLOG_DEFAULT 0

Lexical states for SCLEX_VISUALPROLOG.

#define wxSTC_VISUALPROLOG_IDENTIFIER 8

#define wxSTC_VISUALPROLOG_KEY_DIRECTIVE 3

#define wxSTC_VISUALPROLOG_KEY_MAJOR 1

#define wxSTC_VISUALPROLOG_KEY_MINOR 2

#define wxSTC_VISUALPROLOG_NUMBER 11

#define wxSTC_VISUALPROLOG_OPERATOR 12

#define wxSTC_VISUALPROLOG_STRING 16

#define wxSTC_VISUALPROLOG_STRING_EOL_OPEN 19

#define wxSTC_VISUALPROLOG_STRING_ESCAPE 17

#define wxSTC_VISUALPROLOG_STRING_ESCAPE_ERROR 18

#define wxSTC_VISUALPROLOG_STRING_VERBATIM 20

#define wxSTC_VISUALPROLOG_STRING_VERBATIM_EOL 22

#define wxSTC_VISUALPROLOG_STRING_VERBATIM_SPECIAL 21

#define wxSTC_VISUALPROLOG_VARIABLE 9

#define wxSTC_WEIGHT_BOLD 700

#define wxSTC_WEIGHT_NORMAL 400

#define wxSTC_WEIGHT_SEMIBOLD 600

#define wxSTC_WRAP_CHAR 2

#define wxSTC_WRAP_NONE 0

#define wxSTC_WRAP_WHITESPACE 3

#define wxSTC_WRAP_WORD 1

#define wxSTC_WRAPINDENT_FIXED 0

#define wxSTC_WRAPINDENT_INDENT 2

#define wxSTC_WRAPINDENT_SAME 1

#define wxSTC_WRAPVISUALFLAG_END 0x0001

#define wxSTC_WRAPVISUALFLAG_MARGIN 0x0004

#define wxSTC_WRAPVISUALFLAG_NONE 0x0000

#define wxSTC_WRAPVISUALFLAG_START 0x0002

#define wxSTC_WRAPVISUALFLAGLOC_DEFAULT 0x0000

#define wxSTC_WRAPVISUALFLAGLOC_END_BY_TEXT 0x0001

#define wxSTC_WRAPVISUALFLAGLOC_START_BY_TEXT 0x0002

#define wxSTC_WS_INVISIBLE 0

#define wxSTC_WS_VISIBLEAFTERINDENT 2

#define wxSTC_WS_VISIBLEALWAYS 1

#define wxSTC_YAML_COMMENT 1

#define wxSTC_YAML_DEFAULT 0

Lexical states for SCLEX_YAML.

#define wxSTC_YAML_DOCUMENT 6

#define wxSTC_YAML_ERROR 8

#define wxSTC_YAML_IDENTIFIER 2

#define wxSTC_YAML_KEYWORD 3

#define wxSTC_YAML_NUMBER 4

#define wxSTC_YAML_OPERATOR 9

#define wxSTC_YAML_REFERENCE 5

#define wxSTC_YAML_TEXT 7

22.449.2 Variable Documentation

const wxEventType wxEVT_STC_AUTOCOMP_CANCELLED

const wxEventType wxEVT_STC_AUTOCOMP_CHAR_DELETED

const wxEventType wxEVT_STC_AUTOCOMP_SELECTION

const wxEventType wxEVT_STC_CALLTIP_CLICK

const wxEventType wxEVT_STC_CHANGE

const wxEventType wxEVT_STC_CHARADDED

const wxEventType wxEVT_STC_CLIPBOARD_COPY

const wxEventType wxEVT_STC_CLIPBOARD_PASTE

const wxEventType wxEVT_STC_DO_DROP

const wxEventType wxEVT_STC_DOUBLECLICK

const wxEventType wxEVT_STC_DRAG_OVER

const wxEventType wxEVT_STC_DWELLEND

const wxEventType wxEVT_STC_DWELLSTART

const wxEventType wxEVT_STC_HOTSPOT_CLICK

const wxEventType wxEVT_STC_HOTSPOT_DCLICK

const wxEventType wxEVT_STC_HOTSPOT_RELEASE_CLICK

const wxEventType wxEVT_STC_INDICATOR_CLICK

const wxEventType wxEVT_STC_INDICATOR_RELEASE

const wxEventType wxEVT_STC_KEY

const wxEventType wxEVT_STC_MACRORECORD

const wxEventType wxEVT_STC_MARGINCLICK

const wxEventType wxEVT_STC_MODIFIED

const wxEventType wxEVT_STC_NEEDSHOWN

const wxEventType wxEVT_STC_PAINTED

const wxEventType wxEVT_STC_ROMODIFYATTEMPT

const wxEventType wxEVT_STC_SAVEPOINTLEFT

const wxEventType wxEVT_STC_SAVEPOINTREACHED

const wxEventType wxEVT_STC_START_DRAG

const wxEventType wxEVT_STC_STYLENEEDED

const wxEventType wxEVT_STC_UPDATEUI

const wxEventType wxEVT_STC_URIDROPPED

const wxEventType wxEVT_STC_USERLISTSELECTION

const wxEventType wxEVT_STC_ZOOM

22.450 interface/wx/stdpaths.h File Reference

Classes

- class wxStandardPaths

  wxStandardPaths returns the standard locations in the file system and should be used by applications to find their data files in a portable way.

22.451 interface/wx/stdstream.h File Reference

Classes

- class wxStdInputStreamBuffer

  wxStdInputStreamBuffer is a std::streambuf derived stream buffer which reads from a wxInputStream.
- class wxStdInputStream

  wxStdInputStream is a std::istream derived stream which reads from a wxInputStream.
- class wxStdOutputStreamBuffer

  wxStdOutputStreamBuffer is a std::streambuf derived stream buffer which writes to a wxOutputStream.
- class wxStdOutputStream

  wxStdOutputStream is a std::ostream derived stream which writes to a wxOutputStream.

22.452 interface/wx/stockitem.h File Reference

Enumerations

- enum wxStockLabelQueryFlag {
  wxSTOCK_NOFLAGS = 0,
  wxSTOCK_WITH_MNEMONIC = 1,
  wxSTOCK_WITH_ACCELERATOR = 2,
  wxSTOCK_WITHOUT_ELLIPSIS = 4,
  wxSTOCK_FOR_BUTTON = wxSTOCK_WITHOUT_ELLIPSIS | wxSTOCK_WITH_MNEMONIC }

  Possible values for flags parameter of wxGetStockLabel().

Functions

- wxString wxGetStockLabel (wxWindowID id, long flags=wxSTOCK_WITH_MNEMONIC)

  Returns label that should be used for given id element.

22.452.1 Enumeration Type Documentation

enum wxStockLabelQueryFlag

Possible values for flags parameter of wxGetStockLabel().

The elements of this enum are bit masks and may be combined with each other (except when specified otherwise).

Enumerator

wxSTOCK_NOFLAGS Indicates absence of wxSTOCK_WITH_MNEMONIC and wxSTOCK_WITH_ACCE↩ LERATOR. Requests just the label (e.g. "Print...").

wxSTOCK_WITH_MNEMONIC Request the label with mnemonics character. E.g. "&Print...".

wxSTOCK_WITH_ACCELERATOR Return the label with accelerator following it after TAB. E.g. "Print...\t↩ Ctrl-P". This can be combined with wxSTOCK_WITH_MNEMONIC to get "&Print...\tCtrl-P".

wxSTOCK_WITHOUT_ELLIPSIS Return the label without any ellipsis at the end. By default, stock items text is returned with ellipsis, if appropriate, this flag allows to avoid having it. So using the same example as above, the returned string would be "Print" or "&Print" if wxSTOCK_WITH_MNEMONIC were also used.

This flag can't be combined with wxSTOCK_WITH_ACCELERATOR.

Since

2.9.1

wxSTOCK_FOR_BUTTON Return the label appropriate for a button and not a menu item. Currently the main difference is that the trailing ellipsis used in some stock labels is never included in the returned label. Also, the mnemonics is included if this flag is used. So the returned value for wxID_PRINT when this flag is used is "&Print".

This flag can't be combined with wxSTOCK_WITH_ACCELERATOR.

Since

2.9.1

22.453 interface/wx/stopwatch.h File Reference

Classes

- class wxStopWatch

  The wxStopWatch class allow you to measure time intervals.

22.454 interface/wx/strconv.h File Reference

Classes

- class wxMBConv

  This class is the base class of a hierarchy of classes capable of converting text strings between multibyte (SBCS or DBCS) encodings and Unicode.
- class wxMBConvUTF7

  This class converts between the UTF-7 encoding and Unicode.
- class wxMBConvUTF8

This class converts between the UTF-8 encoding and Unicode.

- class wxMBConvUTF16

  This class is used to convert between multibyte encodings and UTF-16 Unicode encoding (also known as UCS-2).
- class wxMBConvUTF32

  This class is used to convert between multibyte encodings and UTF-32 Unicode encoding (also known as UCS-4).
- class wxCSConv

  This class converts between any character set supported by the system and Unicode.

Variables

- wxMBConv * wxConvFileName

  Conversion object used for converting file names from their external representation to the one used inside the program.

22.455 interface/wx/sysopt.h File Reference

Classes

- class wxSystemOptions

  wxSystemOptions stores option/value pairs that wxWidgets itself or applications can use to alter behaviour at runtime.

22.456 interface/wx/tarstrm.h File Reference

Classes

- class wxTarInputStream

  Input stream for reading tar files.
- class wxTarClassFactory

  Class factory for the tar archive format.
- class wxTarOutputStream

  Output stream for writing tar files.
- class wxTarEntry

  Holds the meta-data for an entry in a tar.

Enumerations

- enum wxTarType {
  wxTAR_REGTYPE = '0',
  wxTAR_LNKTYPE = '1',
  wxTAR_SYMTYPE = '2',
  wxTAR_CHRTYPE = '3',
  wxTAR_BLKTYPE = '4',
  wxTAR_DIRTYPE = '5',
  wxTAR_FIFOTYPE = '6',
  wxTAR_CONTTYPE = '7' }

  wxTarEntry::GetTypeFlag() values
- enum wxTarFormat {
  wxTAR_USTAR,
  wxTAR_PAX }

  Archive Formats (use wxTAR_PAX, it's backward compatible) used by wxTarEntry.

22.456.1 Enumeration Type Documentation

enum wxTarFormat

Archive Formats (use wxTAR_PAX, it's backward compatible) used by wxTarEntry.

Enumerator

wxTAR_USTAR POSIX.1-1990 tar format.

wxTAR_PAX POSIX.1-2001 tar format.

enum wxTarType

wxTarEntry::GetTypeFlag() values

Enumerator

wxTAR_REGTYPE regular file

wxTAR_LNKTYPE hard link

wxTAR_SYMTYPE symbolic link

wxTAR_CHRTYPE character special

wxTAR_BLKTYPE block special

wxTAR_DIRTYPE directory

wxTAR_FIFOTYPE named pipe

wxTAR_CONTTYPE contiguous file

22.457 interface/wx/taskbar.h File Reference

Classes

- class wxTaskBarIconEvent

  The event class used by wxTaskBarIcon.
- class wxTaskBarIcon

  This class represents a taskbar icon.

Enumerations

- enum wxTaskBarIconType {
  wxTBI_DOCK,
  wxTBI_CUSTOM_STATUSITEM,
  wxTBI_DEFAULT_TYPE }

  On OSX Cocoa the taskbar icon can be in the doc or in the status area.

Variables

- wxEventType wxEVT_TASKBAR_MOVE
- wxEventType wxEVT_TASKBAR_LEFT_DOWN
- wxEventType wxEVT_TASKBAR_LEFT_UP
- wxEventType wxEVT_TASKBAR_RIGHT_DOWN
- wxEventType wxEVT_TASKBAR_RIGHT_UP
- wxEventType wxEVT_TASKBAR_LEFT_DCLICK

- wxEventType wxEVT_TASKBAR_RIGHT_DCLICK
- wxEventType wxEVT_TASKBAR_CLICK
- wxEventType wxEVT_TASKBAR_BALLOON_TIMEOUT
- wxEventType wxEVT_TASKBAR_BALLOON_CLICK

22.457.1 Enumeration Type Documentation

enum wxTaskBarIconType

On OSX Cocoa the taskbar icon can be in the doc or in the status area.

This enumeration can be used to select which will be instantiated.

Enumerator

wxTBI_DOCK

wxTBI_CUSTOM_STATUSITEM

wxTBI_DEFAULT_TYPE

22.457.2 Variable Documentation

wxEventType wxEVT_TASKBAR_BALLOON_CLICK

wxEventType wxEVT_TASKBAR_BALLOON_TIMEOUT

wxEventType wxEVT_TASKBAR_CLICK

wxEventType wxEVT_TASKBAR_LEFT_DCLICK

wxEventType wxEVT_TASKBAR_LEFT_DOWN

wxEventType wxEVT_TASKBAR_LEFT_UP

wxEventType wxEVT_TASKBAR_MOVE

wxEventType wxEVT_TASKBAR_RIGHT_DCLICK

wxEventType wxEVT_TASKBAR_RIGHT_DOWN

wxEventType wxEVT_TASKBAR_RIGHT_UP

22.458 interface/wx/taskbarbutton.h File Reference

Classes

- class wxThumbBarButton

  A thumbnail toolbar button is a control that displayed in the thumbnail image of a window in a taskbar button flyout.
- class wxTaskBarButton

  A taskbar button that associated with the window under Windows 7 or later.
- class wxTaskBarJumpListItem

  A wxTaskBarJumpListItem represents an item in a jump list category.
- class wxTaskBarJumpListCategory

  This class represents a category of jump list in the taskbar button.
- class wxTaskBarJumpList

  This class is an transparent wrapper around Windows Jump Lists.

Typedefs

- typedef wxVector
  < wxTaskBarJumpListItem * > wxTaskBarJumpListItems

  A vector of wxTaskBarJumpListItem pointers.
- typedef wxVector
  < wxTaskBarJumpListCategory * > wxTaskBarJumpListCategories

  A vector of wxTaskBarJumpListCategory pointers.

Enumerations

- enum wxTaskBarButtonState {
  wxTASKBAR_BUTTON_NO_PROGRESS = 0,
  wxTASKBAR_BUTTON_INDETERMINATE = 1,
  wxTASKBAR_BUTTON_NORMAL = 2,
  wxTASKBAR_BUTTON_ERROR = 4,
  wxTASKBAR_BUTTON_PAUSED = 8 }

  State of the taskbar button.
- enum wxTaskBarJumpListItemType {
  wxTASKBAR_JUMP_LIST_SEPARATOR,
  wxTASKBAR_JUMP_LIST_TASK,
  wxTASKBAR_JUMP_LIST_DESTIONATION }

  Type of jump list item.

22.458.1 Typedef Documentation

typedef wxVector<wxTaskBarJumpListCategory*> wxTaskBarJumpListCategories

A vector of wxTaskBarJumpListCategory pointers.

typedef wxVector<wxTaskBarJumpListItem*> wxTaskBarJumpListItems

A vector of wxTaskBarJumpListItem pointers.

Since

3.1.0

22.458.2 Enumeration Type Documentation

enum wxTaskBarButtonState

State of the taskbar button.

Since

3.1.0

Enumerator

wxTASKBAR_BUTTON_NO_PROGRESS

wxTASKBAR_BUTTON_INDETERMINATE

wxTASKBAR_BUTTON_NORMAL

wxTASKBAR_BUTTON_ERROR

wxTASKBAR_BUTTON_PAUSED

enum wxTaskBarJumpListItemType

Type of jump list item.

Since

3.1.0

Enumerator

wxTASKBAR_JUMP_LIST_SEPARATOR A separator, Only tasks category supports separators.

wxTASKBAR_JUMP_LIST_TASK A task, represents a link to application.

wxTASKBAR_JUMP_LIST_DESTIONATION Item acts as a link to a file that the application can open.

22.459 interface/wx/textcompleter.h File Reference

Classes

- class wxTextCompleter

  Base class for custom text completer objects.
- class wxTextCompleterSimple

  A simpler base class for custom completer objects.

22.460 interface/wx/textctrl.h File Reference

Classes

- class wxTextAttr

  wxTextAttr represents the character and paragraph attributes, or style, for a range of text in a wxTextCtrl or wxRich↩ TextCtrl.
- class wxTextCtrl

  A text control allows text to be displayed and edited.
- class wxTextUrlEvent
- class wxStreamToTextRedirector

  This class can be used to (temporarily) redirect all output sent to a C++ ostream object to a wxTextCtrl instead.

Macros

- #define wxTE_NO_VSCROLL 0x0002

  wxTextCtrl style flags
- #define wxTE_READONLY 0x0010
- #define wxTE_MULTILINE 0x0020
- #define wxTE_PROCESS_TAB 0x0040
- #define wxTE_LEFT 0x0000
- #define wxTE_CENTER wxALIGN_CENTER_HORIZONTAL
- #define wxTE_RIGHT wxALIGN_RIGHT
- #define wxTE_CENTRE wxTE_CENTER
- #define wxTE_RICH 0x0080
- #define wxTE_PROCESS_ENTER 0x0400
- #define wxTE_PASSWORD 0x0800
- #define wxTE_AUTO_URL 0x1000
- #define wxTE_NOHIDESEL 0x2000

- #define wxTE_DONTWRAP wxHSCROLL
- #define wxTE_CHARWRAP 0x4000
- #define wxTE_WORDWRAP 0x0001
- #define wxTE_BESTWRAP 0x0000
- #define wxTE_RICH2 0x8000
- #define wxTEXT_TYPE_ANY 0

Typedefs

- typedef long wxTextCoord

  wxTextCoord is a line or row number

Enumerations

- enum wxTextAttrAlignment {
  wxTEXT_ALIGNMENT_DEFAULT,
  wxTEXT_ALIGNMENT_LEFT,
  wxTEXT_ALIGNMENT_CENTRE,
  wxTEXT_ALIGNMENT_CENTER = wxTEXT_ALIGNMENT_CENTRE,
  wxTEXT_ALIGNMENT_RIGHT,
  wxTEXT_ALIGNMENT_JUSTIFIED }

  One of the following values can be passed to wxTextAttr::SetAlignment to determine paragraph alignment.

- enum wxTextAttrFlags {
  wxTEXT_ATTR_TEXT_COLOUR = 0x00000001,
  wxTEXT_ATTR_BACKGROUND_COLOUR = 0x00000002,
  wxTEXT_ATTR_FONT_FACE = 0x00000004,
  wxTEXT_ATTR_FONT_POINT_SIZE = 0x00000008,
  wxTEXT_ATTR_FONT_PIXEL_SIZE = 0x10000000,
  wxTEXT_ATTR_FONT_WEIGHT = 0x00000010,
  wxTEXT_ATTR_FONT_ITALIC = 0x00000020,
  wxTEXT_ATTR_FONT_UNDERLINE = 0x00000040,
  wxTEXT_ATTR_FONT_STRIKETHROUGH = 0x08000000,
  wxTEXT_ATTR_FONT_ENCODING = 0x02000000,
  wxTEXT_ATTR_FONT_FAMILY = 0x04000000,
  wxTEXT_ATTR_FONT_SIZE,
  wxTEXT_ATTR_FONT,
  wxTEXT_ATTR_ALIGNMENT = 0x00000080,
  wxTEXT_ATTR_LEFT_INDENT = 0x00000100,
  wxTEXT_ATTR_RIGHT_INDENT = 0x00000200,
  wxTEXT_ATTR_TABS = 0x00000400,
  wxTEXT_ATTR_PARA_SPACING_AFTER = 0x00000800,
  wxTEXT_ATTR_PARA_SPACING_BEFORE = 0x00001000,
  wxTEXT_ATTR_LINE_SPACING = 0x00002000,
  wxTEXT_ATTR_CHARACTER_STYLE_NAME = 0x00004000,
  wxTEXT_ATTR_PARAGRAPH_STYLE_NAME = 0x00008000,
  wxTEXT_ATTR_LIST_STYLE_NAME = 0x00010000,
  wxTEXT_ATTR_BULLET_STYLE = 0x00020000,
  wxTEXT_ATTR_BULLET_NUMBER = 0x00040000,
  wxTEXT_ATTR_BULLET_TEXT = 0x00080000,
  wxTEXT_ATTR_BULLET_NAME = 0x00100000,
  wxTEXT_ATTR_BULLET,
  wxTEXT_ATTR_URL = 0x00200000,
  wxTEXT_ATTR_PAGE_BREAK = 0x00400000,
  wxTEXT_ATTR_EFFECTS = 0x00800000,
  wxTEXT_ATTR_OUTLINE_LEVEL = 0x01000000,
  wxTEXT_ATTR_AVOID_PAGE_BREAK_BEFORE = 0x20000000,
  wxTEXT_ATTR_AVOID_PAGE_BREAK_AFTER = 0x40000000,
  wxTEXT_ATTR_CHARACTER,
  wxTEXT_ATTR_PARAGRAPH,
  wxTEXT_ATTR_ALL = (wxTEXT_ATTR_CHARACTER|wxTEXT_ATTR_PARAGRAPH) }

  The following values are passed in a bitlist to wxTextAttr::SetFlags() to determine what attributes will be considered when setting the attributes for a text control.

- enum wxTextAttrBulletStyle {
  wxTEXT_ATTR_BULLET_STYLE_NONE = 0x00000000,
  wxTEXT_ATTR_BULLET_STYLE_ARABIC = 0x00000001,
  wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER = 0x00000002,
  wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER = 0x00000004,
  wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER = 0x00000008,
  wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER = 0x00000010,
  wxTEXT_ATTR_BULLET_STYLE_SYMBOL = 0x00000020,
  wxTEXT_ATTR_BULLET_STYLE_BITMAP = 0x00000040,
  wxTEXT_ATTR_BULLET_STYLE_PARENTHESES = 0x00000080,
  wxTEXT_ATTR_BULLET_STYLE_PERIOD = 0x00000100,
  wxTEXT_ATTR_BULLET_STYLE_STANDARD = 0x00000200,
  wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS = 0x00000400,
  wxTEXT_ATTR_BULLET_STYLE_OUTLINE = 0x00000800,
  wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT = 0x00000000,
  wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT = 0x00001000,
  wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE = 0x00002000,
  wxTEXT_ATTR_BULLET_STYLE_CONTINUATION = 0x00004000 }

Styles for wxTextAttr::SetBulletStyle.

- enum wxTextAttrEffects {
  wxTEXT_ATTR_EFFECT_NONE = 0x00000000,
  wxTEXT_ATTR_EFFECT_CAPITALS = 0x00000001,
  wxTEXT_ATTR_EFFECT_SMALL_CAPITALS = 0x00000002,
  wxTEXT_ATTR_EFFECT_STRIKETHROUGH = 0x00000004,
  wxTEXT_ATTR_EFFECT_DOUBLE_STRIKETHROUGH = 0x00000008,
  wxTEXT_ATTR_EFFECT_SHADOW = 0x00000010,
  wxTEXT_ATTR_EFFECT_EMBOSS = 0x00000020,
  wxTEXT_ATTR_EFFECT_OUTLINE = 0x00000040,
  wxTEXT_ATTR_EFFECT_ENGRAVE = 0x00000080,
  wxTEXT_ATTR_EFFECT_SUPERSCRIPT = 0x00000100,
  wxTEXT_ATTR_EFFECT_SUBSCRIPT = 0x00000200,
  wxTEXT_ATTR_EFFECT_RTL = 0x00000400,
  wxTEXT_ATTR_EFFECT_SUPPRESS_HYPHENATION = 0x00001000 }

  Styles for wxTextAttr::SetTextEffects().
- enum wxTextAttrLineSpacing {
  wxTEXT_ATTR_LINE_SPACING_NORMAL = 10,
  wxTEXT_ATTR_LINE_SPACING_HALF = 15,
  wxTEXT_ATTR_LINE_SPACING_TWICE = 20 }

  Convenience line spacing values; see wxTextAttr::SetLineSpacing().
- enum wxTextCtrlHitTestResult {
  wxTE_HT_UNKNOWN = -2,
  wxTE_HT_BEFORE,
  wxTE_HT_ON_TEXT,
  wxTE_HT_BELOW,
  wxTE_HT_BEYOND }

  Describes the possible return values of wxTextCtrl::HitTest().

Variables

- wxEventType wxEVT_TEXT
- wxEventType wxEVT_TEXT_ENTER
- wxEventType wxEVT_TEXT_URL
- wxEventType wxEVT_TEXT_MAXLEN

22.460.1 Macro Definition Documentation

#define wxTE_AUTO_URL 0x1000

#define wxTE_BESTWRAP 0x0000

#define wxTE_CENTER wxALIGN_CENTER_HORIZONTAL

#define wxTE_CENTRE wxTE_CENTER

#define wxTE_CHARWRAP 0x4000

#define wxTE_DONTWRAP wxHSCROLL

#define wxTE_LEFT 0x0000

#define wxTE_MULTILINE 0x0020

#define wxTE_NO_VSCROLL 0x0002

wxTextCtrl style flags

#define wxTE_NOHIDESEL 0x2000

#define wxTE_PASSWORD 0x0800

#define wxTE_PROCESS_ENTER 0x0400

#define wxTE_PROCESS_TAB 0x0040

#define wxTE_READONLY 0x0010

#define wxTE_RICH 0x0080

#define wxTE_RICH2 0x8000

#define wxTE_RIGHT wxALIGN_RIGHT

#define wxTE_WORDWRAP 0x0001

#define wxTEXT_TYPE_ANY 0

22.460.2 Typedef Documentation

typedef long wxTextCoord

wxTextCoord is a line or row number

22.460.3 Enumeration Type Documentation

enum wxTextAttrAlignment

One of the following values can be passed to wxTextAttr::SetAlignment to determine paragraph alignment.

Enumerator

wxTEXT_ALIGNMENT_DEFAULT

wxTEXT_ALIGNMENT_LEFT

wxTEXT_ALIGNMENT_CENTRE

wxTEXT_ALIGNMENT_CENTER

wxTEXT_ALIGNMENT_RIGHT

wxTEXT_ALIGNMENT_JUSTIFIED wxTEXT_ALIGNMENT_JUSTIFIED is unimplemented. In future justification may be supported when printing or previewing, only.

enum wxTextAttrBulletStyle

Styles for wxTextAttr::SetBulletStyle.

They can be combined together as a bitlist.

Enumerator

wxTEXT_ATTR_BULLET_STYLE_NONE

wxTEXT_ATTR_BULLET_STYLE_ARABIC

wxTEXT_ATTR_BULLET_STYLE_LETTERS_UPPER

wxTEXT_ATTR_BULLET_STYLE_LETTERS_LOWER

wxTEXT_ATTR_BULLET_STYLE_ROMAN_UPPER

wxTEXT_ATTR_BULLET_STYLE_ROMAN_LOWER

wxTEXT_ATTR_BULLET_STYLE_SYMBOL

wxTEXT_ATTR_BULLET_STYLE_BITMAP wxTEXT_ATTR_BULLET_STYLE_BITMAP is unimplemented.

wxTEXT_ATTR_BULLET_STYLE_PARENTHESES

wxTEXT_ATTR_BULLET_STYLE_PERIOD

wxTEXT_ATTR_BULLET_STYLE_STANDARD

wxTEXT_ATTR_BULLET_STYLE_RIGHT_PARENTHESIS

wxTEXT_ATTR_BULLET_STYLE_OUTLINE

wxTEXT_ATTR_BULLET_STYLE_ALIGN_LEFT

wxTEXT_ATTR_BULLET_STYLE_ALIGN_RIGHT

wxTEXT_ATTR_BULLET_STYLE_ALIGN_CENTRE

wxTEXT_ATTR_BULLET_STYLE_CONTINUATION

enum wxTextAttrEffects

Styles for wxTextAttr::SetTextEffects().

They can be combined together as a bitlist.

Of these, only wxTEXT_ATTR_EFFECT_CAPITALS, wxTEXT_ATTR_EFFECT_STRIKETHROUGH, wxTEXT_↩ ATTR_EFFECT_SUPERSCRIPT and wxTEXT_ATTR_EFFECT_SUBSCRIPT are implemented.

Enumerator

wxTEXT_ATTR_EFFECT_NONE

wxTEXT_ATTR_EFFECT_CAPITALS

wxTEXT_ATTR_EFFECT_SMALL_CAPITALS

wxTEXT_ATTR_EFFECT_STRIKETHROUGH

wxTEXT_ATTR_EFFECT_DOUBLE_STRIKETHROUGH

wxTEXT_ATTR_EFFECT_SHADOW

wxTEXT_ATTR_EFFECT_EMBOSS

wxTEXT_ATTR_EFFECT_OUTLINE

wxTEXT_ATTR_EFFECT_ENGRAVE

wxTEXT_ATTR_EFFECT_SUPERSCRIPT

wxTEXT_ATTR_EFFECT_SUBSCRIPT

wxTEXT_ATTR_EFFECT_RTL

wxTEXT_ATTR_EFFECT_SUPPRESS_HYPHENATION

enum wxTextAttrFlags

The following values are passed in a bitlist to wxTextAttr::SetFlags() to determine what attributes will be considered when setting the attributes for a text control.

Enumerator

wxTEXT_ATTR_TEXT_COLOUR

wxTEXT_ATTR_BACKGROUND_COLOUR
wxTEXT_ATTR_FONT_FACE
wxTEXT_ATTR_FONT_POINT_SIZE
wxTEXT_ATTR_FONT_PIXEL_SIZE
wxTEXT_ATTR_FONT_WEIGHT
wxTEXT_ATTR_FONT_ITALIC
wxTEXT_ATTR_FONT_UNDERLINE
wxTEXT_ATTR_FONT_STRIKETHROUGH
wxTEXT_ATTR_FONT_ENCODING
wxTEXT_ATTR_FONT_FAMILY
wxTEXT_ATTR_FONT_SIZE
wxTEXT_ATTR_FONT Defined as the combination of all `wxTEXT_ATTR_FONT_*` values above.
wxTEXT_ATTR_ALIGNMENT
wxTEXT_ATTR_LEFT_INDENT
wxTEXT_ATTR_RIGHT_INDENT
wxTEXT_ATTR_TABS
wxTEXT_ATTR_PARA_SPACING_AFTER
wxTEXT_ATTR_PARA_SPACING_BEFORE
wxTEXT_ATTR_LINE_SPACING
wxTEXT_ATTR_CHARACTER_STYLE_NAME
wxTEXT_ATTR_PARAGRAPH_STYLE_NAME
wxTEXT_ATTR_LIST_STYLE_NAME
wxTEXT_ATTR_BULLET_STYLE
wxTEXT_ATTR_BULLET_NUMBER
wxTEXT_ATTR_BULLET_TEXT
wxTEXT_ATTR_BULLET_NAME
wxTEXT_ATTR_BULLET Defined as the combination of all `wxTEXT_ATTR_BULLET_*` values above.
wxTEXT_ATTR_URL
wxTEXT_ATTR_PAGE_BREAK
wxTEXT_ATTR_EFFECTS
wxTEXT_ATTR_OUTLINE_LEVEL
wxTEXT_ATTR_AVOID_PAGE_BREAK_BEFORE
wxTEXT_ATTR_AVOID_PAGE_BREAK_AFTER
wxTEXT_ATTR_CHARACTER Combines the styles `wxTEXT_ATTR_FONT`, `wxTEXT_ATTR_EFFECTS`, `wxTEXT_ATTR_BACKGROUND_COLOUR`, `wxTEXT_ATTR_TEXT_COLOUR`, `wxTEXT_ATTR_CHARACTER_STYLE_NAME`, `wxTEXT_ATTR_URL`.
wxTEXT_ATTR_PARAGRAPH Combines all the styles regarding text paragraphs.
wxTEXT_ATTR_ALL Combines all previous values.

enum wxTextAttrLineSpacing

Convenience line spacing values; see wxTextAttr::SetLineSpacing().

Enumerator

wxTEXT_ATTR_LINE_SPACING_NORMAL
wxTEXT_ATTR_LINE_SPACING_HALF
wxTEXT_ATTR_LINE_SPACING_TWICE

enum wxTextCtrlHitTestResult

Describes the possible return values of wxTextCtrl::HitTest().

The element names correspond to the relationship between the point asked for and the character returned, e.g. `wxTE_HT_BEFORE` means that the point is before (leftward or upward) it and so on.

Enumerator

wxTE_HT_UNKNOWN Indicates that wxTextCtrl::HitTest() is not implemented on this platform.

wxTE_HT_BEFORE The point is before the character returned.

wxTE_HT_ON_TEXT The point is directly on the character returned.

wxTE_HT_BELOW The point is below the last line of the control.

wxTE_HT_BEYOND The point is beyond the end of line containing the character returned.

22.460.4 Variable Documentation

wxEventType wxEVT_TEXT

wxEventType wxEVT_TEXT_ENTER

wxEventType wxEVT_TEXT_MAXLEN

wxEventType wxEVT_TEXT_URL

22.461 interface/wx/textdlg.h File Reference

Classes

- class wxPasswordEntryDialog

  This class represents a dialog that requests a one-line password string from the user.
- class wxTextEntryDialog

  This class represents a dialog that requests a one-line text string from the user.

Macros

- #define wxTextEntryDialogStyle (wxOK | wxCANCEL | wxCENTRE | wxWS_EX_VALIDATE_RECURSIVELY)

  Default text dialog style.

Functions

- wxString wxGetTextFromUser (const wxString &message, const wxString &caption=wxGetTextFromUserPromptStr, const wxString &default_value=wxEmptyString, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true)

  Pop up a dialog box with title set to caption, `message`, and a `default_value`.
- wxString wxGetPasswordFromUser (const wxString &message, const wxString &caption=wxGetPasswordFromUserPromptStr, const wxString &default_value=wxEmptyString, wxWindow *parent=NULL, int x=wxDefaultCoord, int y=wxDefaultCoord, bool centre=true)

  Similar to wxGetTextFromUser() but the text entered in the dialog is not shown on screen but replaced with stars.

Variables

- const char wxGetTextFromUserPromptStr [] = "Input Text"

  Default text dialog caption.
- const char wxGetPasswordFromUserPromptStr [] = "Enter Password"

  Default password dialog caption.

22.461.1 Macro Definition Documentation

#define wxTextEntryDialogStyle (wxOK | wxCANCEL | wxCENTRE | wxWS_EX_VALIDATE_RECURSIVELY)

Default text dialog style.

22.461.2 Variable Documentation

const char wxGetPasswordFromUserPromptStr[ ] = "Enter Password"

Default password dialog caption.

const char wxGetTextFromUserPromptStr[ ] = "Input Text"

Default text dialog caption.

22.462 interface/wx/textentry.h File Reference

Classes

- class wxTextEntry

  Common base class for single line text entry fields.

Typedefs

- typedef long wxTextPos

  wxTextPos is a position in the text

22.462.1 Typedef Documentation

typedef long wxTextPos

wxTextPos is a position in the text

22.463 interface/wx/textfile.h File Reference

Classes

- class wxTextFile

  The wxTextFile is a simple class which allows to work with text files on line by line basis.

Enumerations

- enum wxTextFileType {
  wxTextFileType_None,
  wxTextFileType_Unix,
  wxTextFileType_Dos,
  wxTextFileType_Mac,
  wxTextFileType_Os2 }

  The line termination type.

22.463.1 Enumeration Type Documentation

enum wxTextFileType

The line termination type.

Enumerator

wxTextFileType_None incomplete (the last line of the file only)

wxTextFileType_Unix line is terminated with 'LF' = 0xA = 10 = '\n'

wxTextFileType_Dos line is terminated with 'CR' 'LF'

wxTextFileType_Mac line is terminated with 'CR' = 0xD = 13 = '\r'

wxTextFileType_Os2 line is terminated with 'CR' 'LF'

22.464 interface/wx/textwrapper.h File Reference

Classes

- class wxTextWrapper

  Helps wrap lines of text to given width.

22.465 interface/wx/tglbtn.h File Reference

Classes

- class wxToggleButton

  wxToggleButton is a button that stays pressed when clicked by the user.
- class wxBitmapToggleButton

  wxBitmapToggleButton is a wxToggleButton that contains a bitmap instead of text.

Variables

- wxEventType wxEVT_TOGGLEBUTTON

22.465.1 Variable Documentation

wxEventType wxEVT_TOGGLEBUTTON

22.466 interface/wx/time.h File Reference

Functions

- int wxGetTimeZone ()

  Returns the difference between UTC and local time in seconds.
- long wxGetLocalTime ()

  Returns the number of seconds since local time 00:00:00 Jan 1st 1970.
- wxLongLong wxGetLocalTimeMillis ()

  Returns the number of milliseconds since local time 00:00:00 Jan 1st 1970.
- long wxGetUTCTime ()

  Returns the number of seconds since GMT 00:00:00 Jan 1st 1970.
- wxLongLong wxGetUTCTimeMillis ()

  Returns the number of milliseconds since GMT 00:00:00 Jan 1st 1970.
- wxLongLong wxGetUTCTimeUSec ()

  Returns the number of microseconds since GMT 00:00:00 Jan 1st 1970.

22.467 interface/wx/timectrl.h File Reference

Classes

- class wxTimePickerCtrl

  This control allows the user to enter time.

Enumerations

- enum { wxTP_DEFAULT = 0 }

  Styles used with wxTimePickerCtrl.

22.468 interface/wx/timer.h File Reference

Classes

- class wxTimer

  The wxTimer class allows you to execute code at specified intervals.
- class wxTimerRunner

  Starts the timer in its ctor, stops in the dtor.
- class wxTimerEvent

  wxTimerEvent object is passed to the event handler of timer events (see wxTimer::SetOwner).

Macros

- #define wxTIMER_CONTINUOUS false
- #define wxTIMER_ONE_SHOT true

Variables

- wxEventType wxEVT_TIMER

22.468.1 Macro Definition Documentation

#define wxTIMER_CONTINUOUS false

#define wxTIMER_ONE_SHOT true

22.468.2 Variable Documentation

wxEventType wxEVT_TIMER

22.469 interface/wx/tipdlg.h File Reference

Classes

- class wxTipProvider
  This is the class used together with wxShowTip() function.

Functions

- wxTipProvider * wxCreateFileTipProvider (const wxString &filename, size_t currentTip)
  This function creates a wxTipProvider which may be used with wxShowTip().
- bool wxShowTip (wxWindow *parent, wxTipProvider *tipProvider, bool showAtStartup=true)
  This function shows a "startup tip" to the user.

22.470 interface/wx/tipwin.h File Reference

Classes

- class wxTipWindow
  Shows simple text in a popup tip window on creation.

22.471 interface/wx/tls.h File Reference

Macros

- #define wxTLS_TYPE(type) compiler-dependent-implementation
  Macro to be used for thread-specific variables declarations.
- #define wxTLS_VALUE(var)
  Macro to access thread-specific variables.
- #define wxTLS_PTR(var)
  Macro to return address of a thread-specific variables.

22.471.1 Macro Definition Documentation

#define wxTLS_PTR(*var*)

Macro to return address of a thread-specific variables.

This macro is similar to wxTLS_VALUE() except that it always returns a pointer to the type of thread-specific variable.

Notice that this is not a constant expression even if the macro is defined simply as `&var` – the value returned is still different for every thread.

#define wxTLS_TYPE(*type*) compiler-dependent-implementation

Macro to be used for thread-specific variables declarations.

This macro can be used to define thread-specific variables of the specified *type*. Such variables must be global or static and must be POD, i.e. not have any constructors or destructor (even implicitly generated by the compiler due to use of base classes or members which are not POD in a struct).

Example of use:

```
1 struct PerThreadData
2 {
3     ... data which will be different for every thread ...
4 };
5
6 static wxTLS_TYPE(PerThreadData) s_threadDataVar;
7 #define s_threadData (wxTLS_VALUE(s_threadDataVar))
8
9 ... use s_threadData as a variable of type PerThreadData ...
```

Notice that the use of the ugly wxTLS_VALUE() macro is unfortunately required if you need to support platforms without native compiler support for thread-specific variables. If you compile your code only on platforms which do have such support (recent versions of GNU C++ compiler, Microsoft Visual C++ and Sun C++ compiler are known to have it), you can avoid it and use the variable directly.

#define wxTLS_VALUE(*var*)

Macro to access thread-specific variables.

This macro is used to hide the difference in implementation of thread-specific variables under different platforms: they can be of type T used in wxTLS_TYPE() if they are directly supported by the compiler or of type emulating `T *`, i.e. a pointer to this type otherwise. This macro always returns an expression of type `T` itself.

As shown in wxTLS_TYPE() example, you may want to `#define` a symbol wrapping a thread-specific variable with this macro. And, as also explained in wxTLS_TYPE() documentation, you may avoid using it entirely if you target only recent compilers.

See also

wxTLS_PTR()

22.472 interface/wx/tokenzr.h File Reference

Classes

- class wxStringTokenizer

  wxStringTokenizer helps you to break a string up into a number of tokens.

Macros

- #define wxDEFAULT_DELIMITERS " \t\r\n"

  Default wxStringTokenizer delimiters are the usual white space characters.

Enumerations

- enum wxStringTokenizerMode {
  wxTOKEN_INVALID = -1,
  wxTOKEN_DEFAULT,
  wxTOKEN_RET_EMPTY,
  wxTOKEN_RET_EMPTY_ALL,
  wxTOKEN_RET_DELIMS,
  wxTOKEN_STRTOK }

  The behaviour of wxStringTokenizer is governed by the wxStringTokenizer::wxStringTokenizer() or wxStringTokenizer::SetString() with the parameter mode, which may be one of the following:

Functions

- wxArrayString wxStringTokenize (const wxString &str, const wxString &delims=wxDEFAULT_DELIMITERS, wxStringTokenizerMode mode=wxTOKEN_DEFAULT)

  This is a convenience function wrapping wxStringTokenizer which simply returns all tokens found in the given str as an array.

22.472.1 Macro Definition Documentation

#define wxDEFAULT_DELIMITERS " \t\r\n"

Default wxStringTokenizer delimiters are the usual white space characters.

22.472.2 Enumeration Type Documentation

enum wxStringTokenizerMode

The behaviour of wxStringTokenizer is governed by the wxStringTokenizer::wxStringTokenizer() or wxStringTokenizer::SetString() with the parameter *mode*, which may be one of the following:

Enumerator

wxTOKEN_INVALID Invalid tokenizer mode.

wxTOKEN_DEFAULT Default behaviour: wxStringTokenizer will behave in the same way as `strtok()` (wxTOKEN_STRTOK) if the delimiters string only contains white space characters but, unlike the standard function, it will behave like wxTOKEN_RET_EMPTY, returning empty tokens if this is not the case. This is helpful for parsing strictly formatted data where the number of fields is fixed but some of them may be empty (i.e. `TAB` or comma delimited text files).

wxTOKEN_RET_EMPTY In this mode, the empty tokens in the middle of the string will be returned, i.e. `"a::b:"` will be tokenized in three tokens `'a'`, `''` and `'b'`. Notice that all trailing delimiters are ignored in this mode, not just the last one, i.e. a string `"a::b::"` would still result in the same set of tokens.

wxTOKEN_RET_EMPTY_ALL In this mode, empty trailing tokens (including the one after the last delimiter character) will be returned as well. The string `"a::b:"` will be tokenized in four tokens: the already mentioned ones and another empty one as the last one and a string `"a::b::"` will have five tokens.

wxTOKEN_RET_DELIMS In this mode, the delimiter character after the end of the current token (there may be none if this is the last token) is returned appended to the token. Otherwise, it is the same mode as wxTOKEN_RET_EMPTY. Notice that there is no mode like this one but behaving like wxTOKEN_RET_EMPTY_ALL instead of wxTOKEN_RET_EMPTY, use wxTOKEN_RET_EMPTY_ALL and wxStringTokenizer::GetLastDelimiter() to emulate it.

wxTOKEN_STRTOK In this mode the class behaves exactly like the standard `strtok()` function: the empty tokens are never returned.

22.473 interface/wx/toolbook.h File Reference

Classes

- class wxToolbook

  wxToolbook is a class similar to wxNotebook but which uses a wxToolBar to show the labels instead of the tabs.

Macros

- #define wxTBK_BUTTONBAR 0x0100
- #define wxTBK_HORZ_LAYOUT 0x8000

Variables

- wxEventType wxEVT_TOOLBOOK_PAGE_CHANGED
- wxEventType wxEVT_TOOLBOOK_PAGE_CHANGING

22.473.1 Macro Definition Documentation

#define wxTBK_BUTTONBAR 0x0100

#define wxTBK_HORZ_LAYOUT 0x8000

22.473.2 Variable Documentation

wxEventType wxEVT_TOOLBOOK_PAGE_CHANGED

wxEventType wxEVT_TOOLBOOK_PAGE_CHANGING

22.474 interface/wx/tooltip.h File Reference

Classes

- class wxToolTip

  This class holds information about a tooltip associated with a window (see wxWindow::SetToolTip()).

22.475 interface/wx/tracker.h File Reference

Classes

- class wxTrackable

  Add-on base class for a trackable object.

22.476 interface/wx/translation.h File Reference

Classes

- class wxTranslations

  This class allows to get translations for strings.

- class wxTranslationsLoader

  Abstraction of translations discovery and loading.
- class wxFileTranslationsLoader

  Standard wxTranslationsLoader implementation.
- class wxResourceTranslationsLoader

  This loader makes it possible to load translations from Windows resources.
- class wxMsgCatalog

  Represents a loaded translations message catalog.

Macros

- #define wxPLURAL(string, plural, n)

  This macro is identical to _() but for the plural variant of wxGetTranslation().
- #define wxTRANSLATE(string)

  This macro doesn't do anything in the program code – it simply expands to the value of its argument.

Functions

- const wxString & wxGetTranslation (const wxString &string, const wxString &domain=wxEmptyString)

  This function returns the translation of string in the current `locale()`.
- const wxString & wxGetTranslation (const wxString &string, const wxString &plural, unsigned n, const wxString &domain=wxEmptyString)

  This is an overloaded version of wxGetTranslation(const wxString&, const wxString&), please see its documentation for general information.
- const wxString & _ (const wxString &string)

  Macro to be used around all literal strings that should be translated.

22.477 interface/wx/treebase.h File Reference

Classes

- class wxTreeItemId

  An opaque reference to a tree item.
- class wxTreeItemData

  wxTreeItemData is some (arbitrary) user class associated with some item.

Macros

- #define wxTR_NO_BUTTONS 0x0000
- #define wxTR_HAS_BUTTONS 0x0001
- #define wxTR_NO_LINES 0x0004
- #define wxTR_LINES_AT_ROOT 0x0008
- #define wxTR_TWIST_BUTTONS 0x0010
- #define wxTR_SINGLE 0x0000
- #define wxTR_MULTIPLE 0x0020
- #define wxTR_HAS_VARIABLE_ROW_HEIGHT 0x0080
- #define wxTR_EDIT_LABELS 0x0200
- #define wxTR_ROW_LINES 0x0400
- #define wxTR_HIDE_ROOT 0x0800
- #define wxTR_FULL_ROW_HIGHLIGHT 0x2000
- #define wxTR_DEFAULT_STYLE (wxTR_HAS_BUTTONS | wxTR_LINES_AT_ROOT)

Enumerations

- enum wxTreeItemIcon {
  wxTreeItemIcon_Normal,
  wxTreeItemIcon_Selected,
  wxTreeItemIcon_Expanded,
  wxTreeItemIcon_SelectedExpanded,
  wxTreeItemIcon_Max }

  Indicates which type to associate an image with a wxTreeCtrl item.

Functions

- bool operator== (const wxTreeItemId &left, const wxTreeItemId &right)
- bool operator!= (const wxTreeItemId &left, const wxTreeItemId &right)

Variables

- static const int wxTREE_ITEMSTATE_NONE = -1

  special values for the 'state' parameter of wxTreeCtrl::SetItemState()
- static const int wxTREE_ITEMSTATE_NEXT = -2
- static const int wxTREE_ITEMSTATE_PREV = -3
- static const int wxTREE_HITTEST_ABOVE = 0x0001
- static const int wxTREE_HITTEST_BELOW = 0x0002
- static const int wxTREE_HITTEST_NOWHERE = 0x0004
- static const int wxTREE_HITTEST_ONITEMBUTTON = 0x0008
- static const int wxTREE_HITTEST_ONITEMICON = 0x0010
- static const int wxTREE_HITTEST_ONITEMINDENT = 0x0020
- static const int wxTREE_HITTEST_ONITEMLABEL = 0x0040
- static const int wxTREE_HITTEST_ONITEMRIGHT = 0x0080
- static const int wxTREE_HITTEST_ONITEMSTATEICON = 0x0100
- static const int wxTREE_HITTEST_TOLEFT = 0x0200
- static const int wxTREE_HITTEST_TORIGHT = 0x0400
- static const int wxTREE_HITTEST_ONITEMUPPERPART = 0x0800
- static const int wxTREE_HITTEST_ONITEMLOWERPART = 0x1000
- static const int wxTREE_HITTEST_ONITEM

22.477.1 Macro Definition Documentation

#define wxTR_DEFAULT_STYLE (wxTR_HAS_BUTTONS | wxTR_LINES_AT_ROOT)

#define wxTR_EDIT_LABELS 0x0200

#define wxTR_FULL_ROW_HIGHLIGHT 0x2000

#define wxTR_HAS_BUTTONS 0x0001

#define wxTR_HAS_VARIABLE_ROW_HEIGHT 0x0080

#define wxTR_HIDE_ROOT 0x0800

#define wxTR_LINES_AT_ROOT 0x0008

#define wxTR_MULTIPLE 0x0020

#define wxTR_NO_BUTTONS 0x0000

#define wxTR_NO_LINES 0x0004

#define wxTR_ROW_LINES 0x0400

#define wxTR_SINGLE 0x0000

#define wxTR_TWIST_BUTTONS 0x0010

22.477.2 Enumeration Type Documentation

enum **wxTreeItemIcon**

Indicates which type to associate an image with a wxTreeCtrl item.

See also

wxTreeCtrl::GetItemImage(), wxTreeCtrl::SetItemImage()

Enumerator

wxTreeItemIcon_Normal To get/set the item image for when the item is **not** selected and **not** expanded.

wxTreeItemIcon_Selected To get/set the item image for when the item is **selected** and **not** expanded.

wxTreeItemIcon_Expanded To get/set the item image for when the item is **not** selected and **expanded**.

wxTreeItemIcon_SelectedExpanded To get/set the item image for when the item is **selected** and **expanded**.

wxTreeItemIcon_Max

22.477.3 Function Documentation

bool operator!= (const **wxTreeItemId** & *left,* const **wxTreeItemId** & *right*)

bool operator== (const **wxTreeItemId** & *left,* const **wxTreeItemId** & *right*)

22.477.4 Variable Documentation

const int wxTREE_HITTEST_ABOVE = 0x0001 [static]

const int wxTREE_HITTEST_BELOW = 0x0002 [static]

const int wxTREE_HITTEST_NOWHERE = 0x0004 [static]

const int wxTREE_HITTEST_ONITEM [static]

Initial value:

```
= wxTREE_HITTEST_ONITEMICON |
                                        wxTREE_HITTEST_ONITEMLABEL
```

const int wxTREE_HITTEST_ONITEMBUTTON = 0x0008 [static]

const int wxTREE_HITTEST_ONITEMICON = 0x0010 [static]

const int wxTREE_HITTEST_ONITEMINDENT = 0x0020 [static]

const int wxTREE_HITTEST_ONITEMLABEL = 0x0040 [static]

const int wxTREE_HITTEST_ONITEMLOWERPART = 0x1000 [static]

const int wxTREE_HITTEST_ONITEMRIGHT = 0x0080 [static]

const int wxTREE_HITTEST_ONITEMSTATEICON = 0x0100 [static]

const int wxTREE_HITTEST_ONITEMUPPERPART = 0x0800 [static]

const int wxTREE_HITTEST_TOLEFT = 0x0200 [static]

const int wxTREE_HITTEST_TORIGHT = 0x0400 [static]

const int wxTREE_ITEMSTATE_NEXT = -2 [static]

const int wxTREE_ITEMSTATE_NONE = -1 [static]

special values for the 'state' parameter of wxTreeCtrl::SetItemState()

const int wxTREE_ITEMSTATE_PREV = -3 [static]

22.478 interface/wx/treelist.h File Reference

Classes

- class wxTreeListItem

  Unique identifier of an item in wxTreeListCtrl.
- class wxTreeListItemComparator

  Class defining sort order for the items in wxTreeListCtrl.
- class wxTreeListCtrl

  A control combining wxTreeCtrl and wxListCtrl features.
- class wxTreeListEvent

  Event generated by wxTreeListCtrl.

Macros

- #define wxTreeListEventHandler(func) wxEVENT_HANDLER_CAST(wxTreeListEventFunction, func)

  Type of wxTreeListEvent event handlers.

Typedefs

- typedef wxVector< wxTreeListItem > wxTreeListItems

  Container of multiple items.

Enumerations

- enum {
  wxTL_SINGLE = 0x0000,
  wxTL_MULTIPLE = 0x0001,
  wxTL_CHECKBOX = 0x0002,
  wxTL_3STATE = 0x0004,
  wxTL_USER_3STATE = 0x0008,
  wxTL_NO_HEADER = 0x0010,
  wxTL_DEFAULT_STYLE = wxTL_SINGLE,
  wxTL_STYLE_MASK }

  wxTreeListCtrl styles.

Variables

- const wxTreeListItem wxTLI_FIRST

  Special wxTreeListItem value meaning "insert before the first item".
- const wxTreeListItem wxTLI_LAST

  Special wxTreeListItem value meaning "insert after the last item".
- wxEventType wxEVT_TREELIST_SELECTION_CHANGED
- wxEventType wxEVT_TREELIST_ITEM_EXPANDING
- wxEventType wxEVT_TREELIST_ITEM_EXPANDED
- wxEventType wxEVT_TREELIST_ITEM_CHECKED
- wxEventType wxEVT_TREELIST_ITEM_ACTIVATED
- wxEventType wxEVT_TREELIST_ITEM_CONTEXT_MENU
- wxEventType wxEVT_TREELIST_COLUMN_SORTED

22.478.1 Macro Definition Documentation

#define wxTreeListEventHandler(*func*) wxEVENT_HANDLER_CAST(wxTreeListEventFunction, func)

Type of wxTreeListEvent event handlers.

This macro should be used with wxEvtHandler::Connect() when connecting to wxTreeListCtrl events.

22.478.2 Typedef Documentation

typedef wxVector<wxTreeListItem> wxTreeListItems

Container of multiple items.

22.478.3 Enumeration Type Documentation

anonymous enum

wxTreeListCtrl styles.

Notice that using wxTL_USER_3STATE implies wxTL_3STATE and wxTL_3STATE in turn implies wxTL_CHEC↩
KBOX.

Enumerator

wxTL_SINGLE

wxTL_MULTIPLE This is the default anyhow.

wxTL_CHECKBOX Allow multiple selection.

wxTL_3STATE Show checkboxes in the first column.

wxTL_USER_3STATE Allow 3rd state in checkboxes.

wxTL_NO_HEADER Allow user to set 3rd state. Don't show the column headers.

By default this control shows the column headers, using this class allows to avoid this and show only the data.

Since

2.9.5

wxTL_DEFAULT_STYLE

wxTL_STYLE_MASK

22.478.4 Variable Documentation

wxEventType wxEVT_TREELIST_COLUMN_SORTED

wxEventType wxEVT_TREELIST_ITEM_ACTIVATED

wxEventType wxEVT_TREELIST_ITEM_CHECKED

wxEventType wxEVT_TREELIST_ITEM_CONTEXT_MENU

wxEventType wxEVT_TREELIST_ITEM_EXPANDED

wxEventType wxEVT_TREELIST_ITEM_EXPANDING

wxEventType wxEVT_TREELIST_SELECTION_CHANGED

const wxTreeListItem wxTLI_FIRST

Special wxTreeListItem value meaning "insert before the first item".

This value can be passed to wxTreeListCtrl::InsertItem() to achieve the same effect as calling wxTreeListCtrl::↩ PrependItem().

const wxTreeListItem wxTLI_LAST

Special wxTreeListItem value meaning "insert after the last item".

This value can be passed to wxTreeListCtrl::InsertItem() to achieve the same effect as calling wxTreeListCtrl::↩ AppendItem().

22.479 interface/wx/txtstrm.h File Reference

Classes

- class wxTextInputStream

  This class provides functions that reads text data using an input stream, allowing you to read text, floats, and integers.
- class wxTextOutputStream

  This class provides functions that write text data using an output stream, allowing you to write text, floats, and integers.

Enumerations

- enum wxEOL {
  wxEOL_NATIVE,
  wxEOL_UNIX,
  wxEOL_MAC,
  wxEOL_DOS }

  Specifies the end-of-line characters to use with wxTextOutputStream.

22.479.1 Enumeration Type Documentation

enum wxEOL

Specifies the end-of-line characters to use with wxTextOutputStream.

Enumerator

wxEOL_NATIVE Specifies wxTextOutputStream to use the native end-of-line characters.

wxEOL_UNIX Specifies wxTextOutputStream to use Unix end-of-line characters.

wxEOL_MAC Specifies wxTextOutputStream to use Mac end-of-line characters.

wxEOL_DOS Specifies wxTextOutputStream to use DOS end-of-line characters.

22.480 interface/wx/uiaction.h File Reference

Classes

- class wxUIActionSimulator

  wxUIActionSimulator is a class used to simulate user interface actions such as a mouse click or a key press.

22.481 interface/wx/unichar.h File Reference

Classes

- class wxUniChar

  This class represents a single Unicode character.
- class wxUniCharRef

  Writeable reference to a character in wxString.

22.482 interface/wx/uri.h File Reference

Classes

- class wxURI

  wxURI is used to extract information from a URI (Uniform Resource Identifier).

Enumerations

- enum wxURIHostType {
  wxURI_REGNAME,
  wxURI_IPV4ADDRESS,
  wxURI_IPV6ADDRESS,
  wxURI_IPVFUTURE }

  Host type of URI returned from wxURI::GetHostType().

22.482.1 Enumeration Type Documentation

enum wxURIHostType

Host type of URI returned from wxURI::GetHostType().

Enumerator

wxURI_REGNAME Host is a normal register name (`"www.mysite.com"`).

wxURI_IPV4ADDRESS Host is a version 4 ip address (`"192.168.1.100"`).

wxURI_IPV6ADDRESS Host is a version 6 ip address (`"[aa:aa:aa:aa::aa:aa]:5050"`).

wxURI_IPVFUTURE Host is a future ip address, wxURI is unsure what kind.

22.483 interface/wx/url.h File Reference

Classes

- class wxURL

  wxURL is a specialization of wxURI for parsing URLs.

Enumerations

- enum wxURLError {
  wxURL_NOERR = 0,
  wxURL_SNTXERR,
  wxURL_NOPROTO,
  wxURL_NOHOST,
  wxURL_NOPATH,
  wxURL_CONNERR,
  wxURL_PROTOERR }

  Error types returned from wxURL::GetError().

22.483.1 Enumeration Type Documentation

enum wxURLError

Error types returned from wxURL::GetError().

Enumerator

wxURL_NOERR No error.

wxURL_SNTXERR Syntax error in the URL string.

wxURL_NOPROTO Found no protocol which can get this URL.

wxURL_NOHOST A host name is required for this protocol.

wxURL_NOPATH A path is required for this protocol.

wxURL_CONNERR Connection error.

wxURL_PROTOERR An error occurred during negotiation.

22.484 interface/wx/ustring.h File Reference

Classes

- class wxUString

  wxUString is a class representing a Unicode character string where each character is stored using a 32-bit value.

Functions

- wxUString operator+ (const wxUString &s1, const wxUString &s2)

  Concatenation operator.
- bool operator== (const wxUString &s1, const wxUString &s2)

  Equality operator.
- bool operator!= (const wxUString &s1, const wxUString &s2)

  Inequality operator.
- bool operator< (const wxUString &s1, const wxUString &s2)

  Comparison operator.
- bool operator> (const wxUString &s1, const wxUString &s2)

  Comparison operator.
- bool operator<= (const wxUString &s1, const wxUString &s2)

  Comparison operator.
- bool operator>= (const wxUString &s1, const wxUString &s2)

  Comparison operator.

22.484.1 Function Documentation

bool operator!= (const wxUString & *s1*, const wxUString & *s2*) `[inline]`

Inequality operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

wxUString operator+ (const wxUString & *s1*, const wxUString & *s2*) `[inline]`

Concatenation operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

bool operator< (const wxUString & *s1*, const wxUString & *s2*) `[inline]`

Comparison operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

bool operator<= (const wxUString & *s1,* const wxUString & *s2*) `[inline]`

Comparison operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

bool operator== (const wxUString & *s1,* const wxUString & *s2*) `[inline]`

Equality operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

bool operator> (const wxUString & *s1,* const wxUString & *s2*) `[inline]`

Comparison operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

bool operator>= (const wxUString & *s1,* const wxUString & *s2*) `[inline]`

Comparison operator.

wxUString additionally provides overloads for wxString, C string, UTF-16 strings, 32-bit strings, char buffers, single characters etc.

22.485 interface/wx/utils.h File Reference

Classes

- class wxWindowDisabler

  This class disables all windows of the application (may be with the exception of one of them) in its constructor and enables them back in its destructor.

- class wxBusyCursor

  This class makes it easy to tell your user that the program is temporarily busy.

- struct wxExecuteEnv

  This structure can optionally be passed to wxExecute() to specify additional options to use for the child process.

Typedefs

- typedef wxStringToStringHashMap wxEnvVariableHashMap

  A map type containing environment variables names and values.

- typedef int(∗ wxSortCallback)(const void ∗pItem1, const void ∗pItem2, const void ∗user_data)

  Compare function type for use with wxQsort()

Enumerations

- enum wxSignal {
  wxSIGNONE = 0,
  wxSIGHUP,
  wxSIGINT,
  wxSIGQUIT,
  wxSIGILL,
  wxSIGTRAP,
  wxSIGABRT,
  wxSIGEMT,
  wxSIGFPE,
  wxSIGKILL,
  wxSIGBUS,
  wxSIGSEGV,
  wxSIGSYS,
  wxSIGPIPE,
  wxSIGALRM,
  wxSIGTERM }

  Signal constants used by wxProcess.
- enum wxKillError {
  wxKILL_OK,
  wxKILL_BAD_SIGNAL,
  wxKILL_ACCESS_DENIED,
  wxKILL_NO_PROCESS,
  wxKILL_ERROR }

  Return values for wxProcess::Kill.
- enum wxKillFlags {
  wxKILL_NOCHILDREN = 0,
  wxKILL_CHILDREN = 1 }
- enum wxShutdownFlags {
  wxSHUTDOWN_FORCE = 1,
  wxSHUTDOWN_POWEROFF = 2,
  wxSHUTDOWN_REBOOT = 4,
  wxSHUTDOWN_LOGOFF = 8 }
- enum {
  wxStrip_Mnemonics = 1,
  wxStrip_Accel = 2,
  wxStrip_All = wxStrip_Mnemonics | wxStrip_Accel }

  flags for wxStripMenuCodes
- enum {
  wxEXEC_ASYNC = 0,
  wxEXEC_SYNC = 1,
  wxEXEC_SHOW_CONSOLE = 2,
  wxEXEC_MAKE_GROUP_LEADER = 4,
  wxEXEC_NODISABLE = 8,
  wxEXEC_NOEVENTS = 16,
  wxEXEC_HIDE_CONSOLE = 32,
  wxEXEC_BLOCK = wxEXEC_SYNC | wxEXEC_NOEVENTS }

  Bit flags that can be used with wxExecute().

Functions

- void wxBeginBusyCursor (const wxCursor *cursor=wxHOURGLASS_CURSOR)

  Changes the cursor to the given cursor for all windows in the application.
- void wxEndBusyCursor ()

Changes the cursor back to the original cursor, for all windows in the application.

- bool wxIsBusy ()

  Returns true if between two wxBeginBusyCursor() and wxEndBusyCursor() calls.

- void wxBell ()

  Ring the system bell.

- void wxInfoMessageBox (wxWindow *parent)

  Shows a message box with the information about the wxWidgets build used, including its version, most important build parameters and the version of the underlying GUI toolkit.

- wxVersionInfo wxGetLibraryVersionInfo ()

  Get wxWidgets version information.

- wxChar * wxGetenv (const wxString &var)

  This is a macro defined as `getenv()` or its wide char version in Unicode mode.

- bool wxGetEnv (const wxString &var, wxString *value)

  Returns the current value of the environment variable var in value.

- bool wxSetEnv (const wxString &var, const wxString &value)

  Sets the value of the environment variable var (adding it if necessary) to value.

- bool wxUnsetEnv (const wxString &var)

  Removes the variable var from the environment.

- bool wxGetEnvMap (wxEnvVariableHashMap *map)

  Fill a map with the complete content of current environment.

- wxBatteryState wxGetBatteryState ()

  Returns battery state as one of `wxBATTERY_NORMAL_STATE`, `wxBATTERY_LOW_STATE`, `wxBATTERY_CRITICAL_STATE`, `wxBATTERY_SHUTDOWN_STATE` or `wxBATTERY_UNKNOWN_STATE`.

- wxPowerType wxGetPowerType ()

  Returns the type of power source as one of `wxPOWER_SOCKET`, `wxPOWER_BATTERY` or `wxPOWER_UNKNOWN`.

- wxString wxGetDisplayName ()

  Under X only, returns the current display name.

- bool wxGetKeyState (wxKeyCode key)

  For normal keys, returns true if the specified key is currently down.

- wxPoint wxGetMousePosition ()

  Returns the mouse position in screen coordinates.

- wxMouseState wxGetMouseState ()

  Returns the current state of the mouse.

- void wxEnableTopLevelWindows (bool enable=true)

  This function enables or disables all top level windows.

- wxWindow * wxFindWindowAtPoint (const wxPoint &pt)

  Find the deepest window at the given mouse position in screen coordinates, returning the window if found, or NULL if not.

- wxWindow * wxFindWindowByLabel (const wxString &label, wxWindow *parent=NULL)
- wxWindow * wxFindWindowByName (const wxString &name, wxWindow *parent=NULL)
- int wxFindMenuItemId (wxFrame *frame, const wxString &menuString, const wxString &itemString)

  Find a menu item identifier associated with the given frame's menu bar.

- int wxNewId ()
- void wxRegisterId (int id)

  Ensures that Ids subsequently generated by wxNewId() do not clash with the given id.

- bool wxLaunchDefaultApplication (const wxString &document, int flags=0)

  Opens the document in the application associated with the files of this type.

- bool wxLaunchDefaultBrowser (const wxString &url, int flags=0)

  Opens the url in user's default browser.

- bool wxLoadUserResource (const void **outData, size_t *outLen, const wxString &resourceName, const wxChar *resourceType="TEXT", WXHINSTANCE module=0)

  Loads an object from Windows resource file.

- char * wxLoadUserResource (const wxString &resourceName, const wxChar *resourceType="TEXT", int *pLen=NULL, WXHINSTANCE module=0)

  Loads a user-defined Windows resource as a string.
- void wxPostDelete (wxObject *object)
- void wxQsort (void *pbase, size_t total_elems, size_t size, wxSortCallback cmp, const void *user_data)

  Function implementing quick sort algorithm.
- void wxSetDisplayName (const wxString &displayName)

  Under X only, sets the current display name.
- wxString wxStripMenuCodes (const wxString &str, int flags=wxStrip_All)

  Strips any menu codes from str and returns the result.
- wxString wxGetEmailAddress ()

  Copies the user's email address into the supplied buffer, by concatenating the values returned by wxGetFullHostName() and wxGetUserId().
- bool wxGetEmailAddress (char *buf, int sz)
- wxMemorySize wxGetFreeMemory ()

  Returns the amount of free memory in bytes under environments which support it, and -1 if not supported or failed to perform measurement.
- wxString wxGetHomeDir ()

  Return the (current) user's home directory.
- wxString wxGetHostName ()

  Copies the current host machine's name into the supplied buffer.
- bool wxGetHostName (char *buf, int sz)
- wxString wxGetFullHostName ()

  Returns the FQDN (fully qualified domain host name) or an empty string on error.
- wxString wxGetUserHome (const wxString &user=wxEmptyString)

  Returns the home directory for the given user.
- wxString wxGetUserId ()

  This function returns the "user id" also known as "login name" under Unix (i.e.
- bool wxGetUserId (char *buf, int sz)
- wxString wxGetUserName ()

  This function returns the full user name (something like "Mr. John Smith").
- bool wxGetUserName (char *buf, int sz)
- wxString wxGetOsDescription ()

  Returns the string containing the description of the current platform in a user-readable form.
- wxOperatingSystemId wxGetOsVersion (int *major=NULL, int *minor=NULL)

  Gets the version and the operating system ID for currently running OS.
- bool wxIsPlatform64Bit ()

  Returns true if the operating system the program is running under is 64 bit.
- bool wxIsPlatformLittleEndian ()

  Returns true if the current platform is little endian (instead of big endian).
- wxLinuxDistributionInfo wxGetLinuxDistributionInfo ()

  Returns a structure containing information about the currently running Linux distribution.
- long wxExecute (const wxString &command, int flags=wxEXEC_ASYNC, wxProcess *callback=NULL, const wxExecuteEnv *env=NULL)

  Executes another program in Unix or Windows.
- long wxExecute (char **argv, int flags=wxEXEC_ASYNC, wxProcess *callback=NULL, const wxExecuteEnv *env=NULL)

  This is an overloaded version of wxExecute(const wxString&,int,wxProcess), please see its documentation for general information.*
- long wxExecute (wchar_t **argv, int flags=wxEXEC_ASYNC, wxProcess *callback=NULL, const wxExecuteEnv *env=NULL)
- long wxExecute (const wxString &command, wxArrayString &output, int flags=0, const wxExecuteEnv *env=NULL)

This is an overloaded version of wxExecute(const wxString&,int,wxProcess∗), please see its documentation for general information.

- long wxExecute (const wxString &command, wxArrayString &output, wxArrayString &errors, int flags=0, const wxExecuteEnv ∗env=NULL)

  This is an overloaded version of wxExecute(const wxString&,int,wxProcess∗), please see its documentation for general information.
- unsigned long wxGetProcessId ()

  Returns the number uniquely identifying the current process in the system.
- int wxKill (long pid, wxSignal sig=wxSIGTERM, wxKillError ∗rc=NULL, int flags=wxKILL_NOCHILDREN)

  Equivalent to the Unix kill function: send the given signal sig to the process with PID pid.
- bool wxShell (const wxString &command=wxEmptyString)

  Executes a command in an interactive shell window.
- bool wxShutdown (int flags=wxSHUTDOWN_POWEROFF)

  This function shuts down or reboots the computer depending on the value of the flags.
- void wxMicroSleep (unsigned long microseconds)

  Sleeps for the specified number of microseconds.
- void wxMilliSleep (unsigned long milliseconds)

  Sleeps for the specified number of milliseconds.
- wxString wxNow ()

  Returns a string representing the current date and time.
- void wxSleep (int secs)

  Sleeps for the specified number of seconds.
- void wxUsleep (unsigned long milliseconds)

22.485.1 Enumeration Type Documentation

enum wxKillError

Return values for wxProcess::Kill.

Enumerator

wxKILL_OK no error
wxKILL_BAD_SIGNAL no such signal
wxKILL_ACCESS_DENIED permission denied
wxKILL_NO_PROCESS no such process
wxKILL_ERROR another, unspecified error

enum wxKillFlags

Enumerator

wxKILL_NOCHILDREN don't kill children
wxKILL_CHILDREN kill children

enum wxShutdownFlags

Enumerator

wxSHUTDOWN_FORCE can be combined with other flags (MSW-only)
wxSHUTDOWN_POWEROFF power off the computer
wxSHUTDOWN_REBOOT shutdown and reboot
wxSHUTDOWN_LOGOFF close session (currently MSW-only)

enum wxSignal

Signal constants used by wxProcess.

Enumerator

wxSIGNONE verify if the process exists under Unix

wxSIGHUP

wxSIGINT

wxSIGQUIT

wxSIGILL

wxSIGTRAP

wxSIGABRT

wxSIGEMT

wxSIGFPE

wxSIGKILL forcefully kill, dangerous!

wxSIGBUS

wxSIGSEGV

wxSIGSYS

wxSIGPIPE

wxSIGALRM

wxSIGTERM terminate the process gently

22.486 interface/wx/valgen.h File Reference

Classes

- class wxGenericValidator

  wxGenericValidator performs data transfer (but not validation or filtering) for many type of controls.

22.487 interface/wx/validate.h File Reference

Classes

- class wxValidator

  wxValidator is the base class for a family of validator classes that mediate between a class of control, and application data.

Variables

- const wxValidator wxDefaultValidator

  An empty, "null" wxValidator instance.

22.487.1 Variable Documentation

const wxValidator wxDefaultValidator

An empty, "null" wxValidator instance.

22.488 interface/wx/valnum.h File Reference

Classes

- class wxNumValidator< T >

  wxNumValidator is the common base class for numeric validator classes.
- class wxIntegerValidator< T >

  Validator for text entries used for integer entry.
- class wxFloatingPointValidator< T >

  Validator for text entries used for floating point numbers entry.

Enumerations

- enum wxNumValidatorStyle {
  wxNUM_VAL_DEFAULT = 0,
  wxNUM_VAL_THOUSANDS_SEPARATOR = 1,
  wxNUM_VAL_ZERO_AS_BLANK = 2,
  wxNUM_VAL_NO_TRAILING_ZEROES }

  Bit masks used for numeric validator styles.

Functions

- template<typename T >
  wxIntegerValidator< T > wxMakeIntegerValidator (T *value, int style=wxNUM_VAL_DEFAULT)

  Creates a wxIntegerValidator object with automatic type deduction.

- template<typename T >
  wxFloatingPointValidator< T > wxMakeFloatingPointValidator (T *value, int style=wxNUM_VAL_DEFAULT)

  Creates a wxFloatingPointValidator object with automatic type deduction.
- template<typename T >
  wxFloatingPointValidator< T > wxMakeFloatingPointValidator (int precision, T *value, int style=wxNUM_V↩
  AL_DEFAULT)

  Creates a wxFloatingPointValidator object with automatic type deduction.

22.488.1 Function Documentation

template<typename T > wxFloatingPointValidator<T> wxMakeFloatingPointValidator (T * *value,* int *style =* wxNUM_VAL_DEFAULT) [inline]

Creates a wxFloatingPointValidator object with automatic type deduction.

Similarly to wxMakeIntegerValidator(), this function allows to avoid explicitly specifying the validator type.

Since

2.9.2

template<typename T > wxFloatingPointValidator<T> wxMakeFloatingPointValidator (int *precision,* T * *value,* int *style* = wxNUM_VAL_DEFAULT) [inline]

Creates a wxFloatingPointValidator object with automatic type deduction.

Similarly to wxMakeIntegerValidator(), this function allows to avoid explicitly specifying the validator type.

Since

2.9.2

template<typename T > wxIntegerValidator<T> wxMakeIntegerValidator (T * *value*, int *style* = wxNUM_VAL_DEFAULT)

Creates a wxIntegerValidator object with automatic type deduction.

This function can be used to create wxIntegerValidator object without explicitly specifying its type, e.g. write just:

```
1 new wxTextCtrl(..., wxMakeIntegerValidator(&m_var));
```

instead of more verbose

```
1 new wxTextCtrl(..., wxIntegerValidator<unsigned long>(&m_var));
```

Since

2.9.2

22.489 interface/wx/valtext.h File Reference

Classes

- class wxTextValidator

  wxTextValidator validates text controls, providing a variety of filtering behaviours.

Enumerations

- enum wxTextValidatorStyle {
  wxFILTER_NONE,
  wxFILTER_EMPTY,
  wxFILTER_ASCII,
  wxFILTER_ALPHA,
  wxFILTER_ALPHANUMERIC,
  wxFILTER_DIGITS,
  wxFILTER_NUMERIC,
  wxFILTER_INCLUDE_LIST,
  wxFILTER_INCLUDE_CHAR_LIST,
  wxFILTER_EXCLUDE_LIST,
  wxFILTER_EXCLUDE_CHAR_LIST }

  Styles used by wxTextValidator.

22.489.1 Enumeration Type Documentation

enum wxTextValidatorStyle

Styles used by wxTextValidator.

Note that when you specify more styles in wxTextValidator the validation checks are performed in the order in which the styles of this enumeration are defined.

Enumerator

wxFILTER_NONE No filtering takes place.

wxFILTER_EMPTY Empty strings are filtered out. If this style is not specified then empty strings are accepted only if they pass the other checks (if you use more than one wxTextValidatorStyle).

wxFILTER_ASCII Non-ASCII characters are filtered out. See wxString::IsAscii.

wxFILTER_ALPHA Non-alpha characters are filtered out. Uses the wxWidgets wrapper for the standard CRT function `isalpha` (which is locale-dependent) on all characters of the string.

wxFILTER_ALPHANUMERIC Non-alphanumeric characters are filtered out. Uses the wxWidgets wrapper for the standard CRT function `isalnum` (which is locale-dependent) on all characters of the string.

wxFILTER_DIGITS Non-numeric characters are filtered out. Uses the wxWidgets wrapper for the standard CRT function `isdigit` (which is locale-dependent) on all characters of the string.

wxFILTER_NUMERIC Non-numeric characters are filtered out. Works like `wxFILTER_DIGITS` but allows also decimal points, minus/plus signs and the 'e' or 'E' character to input exponents. Note that this is not the same behaviour of wxString::IsNumber().

wxFILTER_INCLUDE_LIST Use an include list. The validator checks if the user input is on the list, complaining if not. See wxTextValidator::SetIncludes().

wxFILTER_INCLUDE_CHAR_LIST Use an include list. The validator checks if each input character is in the list (one character per list element), complaining if not. See wxTextValidator::SetCharIncludes().

wxFILTER_EXCLUDE_LIST Use an exclude list. The validator checks if the user input is on the list, complaining if it is. See wxTextValidator::SetExcludes().

wxFILTER_EXCLUDE_CHAR_LIST Use an exclude list. The validator checks if each input character is in the list (one character per list element), complaining if it is. See wxTextValidator::SetCharExcludes().

22.490 interface/wx/variant.h File Reference

Classes

- class wxVariant

  The wxVariant class represents a container for any type.
- class wxVariantData

  The wxVariantData class is used to implement a new type for wxVariant.

Macros

- #define wxGetVariantCast(var, classname)

  This macro returns a pointer to the data stored in var (wxVariant) cast to the type classname if the data is of this type (the check is done during the run-time) or NULL otherwise.

22.491 interface/wx/vector.h File Reference

Classes

- class wxVector< T >

  wxVector<T> is a template class which implements most of the `std::vector` class and can be used like it.

Functions

- template<typename T >
  void wxVectorSort (wxVector< T > &v)

  Sort the contents of a `wxVector<T>`.

22.491.1 Function Documentation

template<typename T > void wxVectorSort (wxVector< T > & *v*)

Sort the contents of a wxVector<T>.

In a STL build this function will be defined as a thin wrapper around std::sort. To be sortable the contained type must support the less-than operator.

```
1 wxVector<SomeClass> v;
2 ... // items are added to the vector v...
3 wxVectorSort(v);
```

See also

wxVector<T>

22.492 interface/wx/version.h File Reference

Macros

- #define wxCHECK_VERSION(major, minor, release)

  This is a macro which evaluates to true if the current wxWidgets version is at least major.minor.release.
- #define wxCHECK_VERSION_FULL(major, minor, release, subrel)

  Same as wxCHECK_VERSION() but also checks that wxSUBRELEASE_NUMBER is at least subrel.

22.493 interface/wx/versioninfo.h File Reference

Classes

- class wxVersionInfo

  wxVersionInfo contains version information.

22.494 interface/wx/vidmode.h File Reference

Classes

- struct wxVideoMode

  Determines the sizes and locations of displays connected to the system.

Variables

- const wxVideoMode wxDefaultVideoMode

  A global wxVideoMode instance used by wxDisplay.

22.494.1 Variable Documentation

const wxVideoMode wxDefaultVideoMode

A global wxVideoMode instance used by wxDisplay.

22.495 interface/wx/vlbox.h File Reference

Classes

- class wxVListBox

  wxVListBox is a wxListBox-like control with the following two main differences from a regular wxListBox: it can have an arbitrarily huge number of items because it doesn't store them itself but uses the OnDrawItem() callback to draw them (so it is a virtual listbox) and its items can have variable height as determined by OnMeasureItem() (so it is also a listbox with the lines of variable height).

22.496 interface/wx/volume.h File Reference

Classes

- class wxFSVolume

  wxFSVolume represents a volume (also known as 'drive') in a file system under wxMSW.

Enumerations

- enum wxFSVolumeFlags {
  wxFS_VOL_MOUNTED = 0x0001,
  wxFS_VOL_REMOVABLE = 0x0002,
  wxFS_VOL_READONLY = 0x0004,
  wxFS_VOL_REMOTE = 0x0008 }

  The volume flags.
- enum wxFSVolumeKind {
  wxFS_VOL_FLOPPY,
  wxFS_VOL_DISK,
  wxFS_VOL_CDROM,
  wxFS_VOL_DVDROM,
  wxFS_VOL_NETWORK,
  wxFS_VOL_OTHER,
  wxFS_VOL_MAX }

  The volume types.
- enum wxFSIconType {
  wxFS_VOL_ICO_SMALL = 0,
  wxFS_VOL_ICO_LARGE,
  wxFS_VOL_ICO_SEL_SMALL,
  wxFS_VOL_ICO_SEL_LARGE,
  wxFS_VOL_ICO_MAX }

  Icon types used by wxFSVolume.

22.496.1 Enumeration Type Documentation

enum wxFSIconType

Icon types used by wxFSVolume.

Enumerator

wxFS_VOL_ICO_SMALL

wxFS_VOL_ICO_LARGE

wxFS_VOL_ICO_SEL_SMALL

wxFS_VOL_ICO_SEL_LARGE
wxFS_VOL_ICO_MAX

enum wxFSVolumeFlags

The volume flags.

Enumerator

wxFS_VOL_MOUNTED Is the volume mounted?
wxFS_VOL_REMOVABLE Is the volume removable (floppy, CD, ...)?
wxFS_VOL_READONLY Read only? (otherwise read write).
wxFS_VOL_REMOTE Network resources.

enum wxFSVolumeKind

The volume types.

Enumerator

wxFS_VOL_FLOPPY
wxFS_VOL_DISK
wxFS_VOL_CDROM
wxFS_VOL_DVDROM
wxFS_VOL_NETWORK
wxFS_VOL_OTHER
wxFS_VOL_MAX

22.497 interface/wx/vscroll.h File Reference

Classes

- class wxVarScrollHelperBase

  This class provides all common base functionality for scroll calculations shared among all variable scrolled window implementations as well as automatic scrollbar functionality, saved scroll positions, controlling target windows to be scrolled, as well as defining all required virtual functions that need to be implemented for any orientation specific work.
- class wxVarVScrollHelper

  This class provides functions wrapping the wxVarScrollHelperBase class, targeted for vertical-specific scrolling.
- class wxVarHScrollHelper

  This class provides functions wrapping the wxVarScrollHelperBase class, targeted for horizontal-specific scrolling.
- class wxVarHVScrollHelper

  This class provides functions wrapping the wxVarHScrollHelper and wxVarVScrollHelper classes, targeted for scrolling a window in both axis.
- class wxVScrolledWindow

  In the name of this class, "V" may stand for "variable" because it can be used for scrolling rows of variable heights; "virtual", because it is not necessary to know the heights of all rows in advance – only those which are shown on the screen need to be measured; or even "vertical", because this class only supports scrolling vertically.
- class wxHScrolledWindow

  In the name of this class, "H" stands for "horizontal" because it can be used for scrolling columns of variable widths.
- class wxHVScrolledWindow

  This window inherits all functionality of both vertical and horizontal, variable scrolled windows.

22.498 interface/wx/weakref.h File Reference

Classes

- class wxWeakRefDynamic< T >

  wxWeakRefDynamic<T> is a template class for weak references that is used in the same way as wxWeakRef<T>.
- class wxWeakRef< T >

  wxWeakRef<T> is a template class for weak references to wxWidgets objects, such as wxEvtHandler, wxWindow and wxObject.

22.499 interface/wx/webview.h File Reference

Classes

- class wxWebViewHistoryItem

  A simple class that contains the URL and title of an element of the history of a wxWebView.
- class wxWebViewFactory

  An abstract factory class for creating wxWebView backends.
- class wxWebViewHandler

  The base class for handling custom schemes in wxWebView, for example to allow virtual file system support.
- class wxWebView

  This control may be used to render web (HTML / CSS / javascript) documents.
- class wxWebViewEvent

  A navigation event holds information about events associated with wxWebView objects.

Enumerations

- enum wxWebViewZoom {
  wxWEBVIEW_ZOOM_TINY,
  wxWEBVIEW_ZOOM_SMALL,
  wxWEBVIEW_ZOOM_MEDIUM,
  wxWEBVIEW_ZOOM_LARGE,
  wxWEBVIEW_ZOOM_LARGEST }

  Zoom levels available in wxWebView.
- enum wxWebViewZoomType {
  wxWEBVIEW_ZOOM_TYPE_LAYOUT,
  wxWEBVIEW_ZOOM_TYPE_TEXT }

  The type of zooming that the web view control can perform.
- enum wxWebViewNavigationError {
  wxWEBVIEW_NAV_ERR_CONNECTION,
  wxWEBVIEW_NAV_ERR_CERTIFICATE,
  wxWEBVIEW_NAV_ERR_AUTH,
  wxWEBVIEW_NAV_ERR_SECURITY,
  wxWEBVIEW_NAV_ERR_NOT_FOUND,
  wxWEBVIEW_NAV_ERR_REQUEST,
  wxWEBVIEW_NAV_ERR_USER_CANCELLED,
  wxWEBVIEW_NAV_ERR_OTHER }

  Types of errors that can cause navigation to fail.
- enum wxWebViewReloadFlags {
  wxWEBVIEW_RELOAD_DEFAULT,
  wxWEBVIEW_RELOAD_NO_CACHE }

  Type of refresh.

- enum wxWebViewFindFlags {
  wxWEBVIEW_FIND_WRAP = 0x0001,
  wxWEBVIEW_FIND_ENTIRE_WORD = 0x0002,
  wxWEBVIEW_FIND_MATCH_CASE = 0x0004,
  wxWEBVIEW_FIND_HIGHLIGHT_RESULT = 0x0008,
  wxWEBVIEW_FIND_BACKWARDS = 0x0010,
  wxWEBVIEW_FIND_DEFAULT = 0 }

  Find flags used when searching for text on page.

Variables

- wxEventType wxEVT_WEBVIEW_NAVIGATING
- wxEventType wxEVT_WEBVIEW_NAVIGATED
- wxEventType wxEVT_WEBVIEW_LOADED
- wxEventType wxEVT_WEBVIEW_ERROR
- wxEventType wxEVT_WEBVIEW_NEWWINDOW
- wxEventType wxEVT_WEBVIEW_TITLE_CHANGED

22.499.1 Enumeration Type Documentation

enum wxWebViewFindFlags

Find flags used when searching for text on page.

Enumerator

wxWEBVIEW_FIND_WRAP Causes the search to restart when end or beginning reached.

wxWEBVIEW_FIND_ENTIRE_WORD Matches an entire word when searching.

wxWEBVIEW_FIND_MATCH_CASE Match case, i.e. case sensitive searching

wxWEBVIEW_FIND_HIGHLIGHT_RESULT Highlights the search results.

wxWEBVIEW_FIND_BACKWARDS Searches for phrase in backward direction.

wxWEBVIEW_FIND_DEFAULT The default flag, which is simple searching.

enum wxWebViewNavigationError

Types of errors that can cause navigation to fail.

Enumerator

wxWEBVIEW_NAV_ERR_CONNECTION Connection error (timeout, etc.)

wxWEBVIEW_NAV_ERR_CERTIFICATE Invalid certificate.

wxWEBVIEW_NAV_ERR_AUTH Authentication required.

wxWEBVIEW_NAV_ERR_SECURITY Other security error.

wxWEBVIEW_NAV_ERR_NOT_FOUND Requested resource not found.

wxWEBVIEW_NAV_ERR_REQUEST Invalid request/parameters (e.g. bad URL, bad protocol, unsupported resource type)

wxWEBVIEW_NAV_ERR_USER_CANCELLED The user cancelled (e.g. in a dialog)

wxWEBVIEW_NAV_ERR_OTHER Another (exotic) type of error that didn't fit in other categories.

enum wxWebViewReloadFlags

Type of refresh.

Enumerator

wxWEBVIEW_RELOAD_DEFAULT Default reload, will access cache.

wxWEBVIEW_RELOAD_NO_CACHE Reload the current view without accessing the cache.

enum wxWebViewZoom

Zoom levels available in wxWebView.

Enumerator

wxWEBVIEW_ZOOM_TINY

wxWEBVIEW_ZOOM_SMALL

wxWEBVIEW_ZOOM_MEDIUM default size

wxWEBVIEW_ZOOM_LARGE

wxWEBVIEW_ZOOM_LARGEST

enum wxWebViewZoomType

The type of zooming that the web view control can perform.

Enumerator

wxWEBVIEW_ZOOM_TYPE_LAYOUT The entire layout scales when zooming, including images.

wxWEBVIEW_ZOOM_TYPE_TEXT Only the text changes in size when zooming, images and other layout elements retain their initial size.

22.499.2 Variable Documentation

wxEventType wxEVT_WEBVIEW_ERROR

wxEventType wxEVT_WEBVIEW_LOADED

wxEventType wxEVT_WEBVIEW_NAVIGATED

wxEventType wxEVT_WEBVIEW_NAVIGATING

wxEventType wxEVT_WEBVIEW_NEWWINDOW

wxEventType wxEVT_WEBVIEW_TITLE_CHANGED

22.500 interface/wx/webviewarchivehandler.h File Reference

Classes

- class wxWebViewArchiveHandler

  A custom handler for the file scheme which also supports loading from archives.

22.501 interface/wx/webviewfshandler.h File Reference

Classes

- class wxWebViewFSHandler

  A wxWebView file system handler to support standard wxFileSystem protocols of the form `example:page.htm` *The handler allows wxWebView to use wxFileSystem in a similar fashion to its use with wxHtml.*

22.502 interface/wx/wfstream.h File Reference

Classes

- class wxTempFileOutputStream

  wxTempFileOutputStream is an output stream based on wxTempFile.
- class wxFFileOutputStream

  This class represents data written to a file.
- class wxFileOutputStream

  This class represents data written to a file.
- class wxFileInputStream

  This class represents data read in from a file.
- class wxFFileInputStream

  This class represents data read in from a file.
- class wxFFileStream

  This stream allows to both read from and write to a file using buffered STDIO functions.
- class wxFileStream

  This class represents data that can be both read from and written to a file.

22.503 interface/wx/windowid.h File Reference

Classes

- class wxIdManager

  wxIdManager is responsible for allocating and releasing window IDs.

Typedefs

- typedef int wxWindowID

  The type of unique identifiers (ID) used for wxWindow-derived classes.

22.503.1 Typedef Documentation

typedef int wxWindowID

The type of unique identifiers (ID) used for wxWindow-derived classes.

22.504 interface/wx/windowptr.h File Reference

Classes

- class wxWindowPtr< T >

  A reference-counted smart pointer for holding wxWindow instances.

22.505 interface/wx/withimages.h File Reference

Classes

- class wxWithImages

  A mixin class to be used with other classes that use a wxImageList.

22.506 interface/wx/wizard.h File Reference

Classes

- class wxWizardPage

  wxWizardPage is one of the screens in wxWizard: it must know what are the following and preceding pages (which may be NULL for the first/last page).
- class wxWizardEvent

  wxWizardEvent class represents an event generated by the wxWizard: this event is first sent to the page itself and, if not processed there, goes up the window hierarchy as usual.
- class wxWizardPageSimple

  wxWizardPageSimple is the simplest possible wxWizardPage implementation: it just returns the pointers given to its constructor from wxWizardPage::GetNext() and wxWizardPage::GetPrev() functions.
- class wxWizard

  wxWizard is the central class for implementing 'wizard-like' dialogs.

Macros

- #define wxWIZARD_EX_HELPBUTTON 0x00000010
- #define wxWIZARD_VALIGN_TOP 0x01
- #define wxWIZARD_VALIGN_CENTRE 0x02
- #define wxWIZARD_VALIGN_BOTTOM 0x04
- #define wxWIZARD_HALIGN_LEFT 0x08
- #define wxWIZARD_HALIGN_CENTRE 0x10
- #define wxWIZARD_HALIGN_RIGHT 0x20
- #define wxWIZARD_TILE 0x40

Variables

- wxEventType wxEVT_WIZARD_PAGE_CHANGED
- wxEventType wxEVT_WIZARD_PAGE_CHANGING
- wxEventType wxEVT_WIZARD_CANCEL
- wxEventType wxEVT_WIZARD_HELP
- wxEventType wxEVT_WIZARD_FINISHED
- wxEventType wxEVT_WIZARD_PAGE_SHOWN
- wxEventType wxEVT_WIZARD_BEFORE_PAGE_CHANGED

22.506.1 Macro Definition Documentation

#define wxWIZARD_EX_HELPBUTTON 0x00000010

#define wxWIZARD_HALIGN_CENTRE 0x10

#define wxWIZARD_HALIGN_LEFT 0x08

#define wxWIZARD_HALIGN_RIGHT 0x20

#define wxWIZARD_TILE 0x40

#define wxWIZARD_VALIGN_BOTTOM 0x04

#define wxWIZARD_VALIGN_CENTRE 0x02

#define wxWIZARD_VALIGN_TOP 0x01

22.506.2 Variable Documentation

wxEventType wxEVT_WIZARD_BEFORE_PAGE_CHANGED

wxEventType wxEVT_WIZARD_CANCEL

wxEventType wxEVT_WIZARD_FINISHED

wxEventType wxEVT_WIZARD_HELP

wxEventType wxEVT_WIZARD_PAGE_CHANGED

wxEventType wxEVT_WIZARD_PAGE_CHANGING

wxEventType wxEVT_WIZARD_PAGE_SHOWN

22.507 interface/wx/wrapsizer.h File Reference

Classes

- class wxWrapSizer

  A wrap sizer lays out its items in a single line, like a box sizer – as long as there is space available in that direction.

Enumerations

- enum {
  wxEXTEND_LAST_ON_EACH_LINE,
  wxREMOVE_LEADING_SPACES,
  wxWRAPSIZER_DEFAULT_FLAGS }

22.507.1 Enumeration Type Documentation

anonymous enum

Enumerator

wxEXTEND_LAST_ON_EACH_LINE

wxREMOVE_LEADING_SPACES

wxWRAPSIZER_DEFAULT_FLAGS

22.508 interface/wx/wupdlock.h File Reference

Classes

- class wxWindowUpdateLocker

  This tiny class prevents redrawing of a wxWindow during its lifetime by using wxWindow::Freeze() and wxWindow::Thaw() methods.

22.509 interface/wx/wxcrt.h File Reference

Functions

- bool wxIsEmpty (const char ∗s)
- bool wxIsEmpty (const wchar_t ∗s)
- bool wxIsEmpty (const wxCharBuffer &s)
- bool wxIsEmpty (const wxWCharBuffer &s)
- bool wxIsEmpty (const wxString &s)
- bool wxIsEmpty (const wxCStrData &s)
- wxChar ∗ wxTmemchr (const wxChar ∗s, wxChar c, size_t l)
- int wxTmemcmp (const wxChar ∗sz1, const wxChar ∗sz2, size_t len)
- wxChar ∗ wxTmemcpy (wxChar ∗szOut, const wxChar ∗szIn, size_t len)
- wxChar ∗ wxTmemmove (wxChar ∗szOut, const wxChar ∗szIn, size_t len)
- wxChar ∗ wxTmemset (wxChar ∗szOut, const wxChar cIn, size_t len)
- char ∗ wxTmemchr (const char ∗s, char c, size_t len)
- int wxTmemcmp (const char ∗sz1, const char ∗sz2, size_t len)
- char ∗ wxTmemcpy (char ∗szOut, const char ∗szIn, size_t len)
- char ∗ wxTmemmove (char ∗szOut, const char ∗szIn, size_t len)
- char ∗ wxTmemset (char ∗szOut, const char cIn, size_t len)
- char ∗ wxSetlocale (int category, const wxCharBuffer &locale)
- char ∗ wxSetlocale (int category, const wxString &locale)
- char ∗ wxSetlocale (int category, const wxCStrData &locale)
- size_t wxStrlen (const wxCharBuffer &s)
- size_t wxStrlen (const wxWCharBuffer &s)
- size_t wxStrlen (const wxString &s)
- size_t wxStrlen (const wxCStrData &s)
- size_t wxStrnlen (const char ∗str, size_t maxlen)
- size_t wxStrnlen (const wchar_t ∗str, size_t maxlen)
- char ∗ wxStrdup (const wxCharBuffer &s)
- wchar_t ∗ wxStrdup (const wxWCharBuffer &s)
- char ∗ wxStrdup (const wxString &s)
- char ∗ wxStrdup (const wxCStrData &s)
- char ∗ wxStrcpy (char ∗dest, const char ∗src)
- wchar_t ∗ wxStrcpy (wchar_t ∗dest, const wchar_t ∗src)
- char ∗ wxStrcpy (char ∗dest, const wxString &src)
- char ∗ wxStrcpy (char ∗dest, const wxCStrData &src)
- char ∗ wxStrcpy (char ∗dest, const wxCharBuffer &src)
- wchar_t ∗ wxStrcpy (wchar_t ∗dest, const wxString &src)
- wchar_t ∗ wxStrcpy (wchar_t ∗dest, const wxCStrData &src)
- wchar_t ∗ wxStrcpy (wchar_t ∗dest, const wxWCharBuffer &src)

- char * wxStrcpy (char *dest, const wchar_t *src)
- wchar_t * wxStrcpy (wchar_t *dest, const char *src)
- char * wxStrncpy (char *dest, const char *src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wchar_t *src, size_t n)
- char * wxStrncpy (char *dest, const wxString &src, size_t n)
- char * wxStrncpy (char *dest, const wxCStrData &src, size_t n)
- char * wxStrncpy (char *dest, const wxCharBuffer &src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wxString &src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wxCStrData &src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const wxWCharBuffer &src, size_t n)
- char * wxStrncpy (char *dest, const wchar_t *src, size_t n)
- wchar_t * wxStrncpy (wchar_t *dest, const char *src, size_t n)
- size_t wxStrlcpy (char *dest, const char *src, size_t n)
- size_t wxStrlcpy (wchar_t *dest, const wchar_t *src, size_t n)
- char * wxStrcat (char *dest, const char *src)
- wchar_t * wxStrcat (wchar_t *dest, const wchar_t *src)
- char * wxStrcat (char *dest, const wxString &src)
- char * wxStrcat (char *dest, const wxCStrData &src)
- char * wxStrcat (char *dest, const wxCharBuffer &src)
- wchar_t * wxStrcat (wchar_t *dest, const wxString &src)
- wchar_t * wxStrcat (wchar_t *dest, const wxCStrData &src)
- wchar_t * wxStrcat (wchar_t *dest, const wxWCharBuffer &src)
- char * wxStrcat (char *dest, const wchar_t *src)
- wchar_t * wxStrcat (wchar_t *dest, const char *src)
- char * wxStrncat (char *dest, const char *src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wchar_t *src, size_t n)
- char * wxStrncat (char *dest, const wxString &src, size_t n)
- char * wxStrncat (char *dest, const wxCStrData &src, size_t n)
- char * wxStrncat (char *dest, const wxCharBuffer &src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wxString &src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wxCStrData &src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const wxWCharBuffer &src, size_t n)
- char * wxStrncat (char *dest, const wchar_t *src, size_t n)
- wchar_t * wxStrncat (wchar_t *dest, const char *src, size_t n)
- int wxStrcmp_String (const wxString &s1, const T &s2)
- int wxStricmp_String (const wxString &s1, const T &s2)
- int wxStrcoll_String (const wxString &s1, const T &s2)
- size_t wxStrspn_String (const wxString &s1, const T &s2)
- size_t wxStrcspn_String (const wxString &s1, const T &s2)
- int wxStrncmp_String (const wxString &s1, const T &s2, size_t n)
- int wxStrnicmp_String (const wxString &s1, const T &s2, size_t n)
- size_t wxStrxfrm (char *dest, const char *src, size_t n)
- size_t wxStrxfrm (wchar_t *dest, const wchar_t *src, size_t n)
- size_t wxStrxfrm (T *dest, const wxCharTypeBuffer< T > &src, size_t n)
- size_t wxStrxfrm (char *dest, const wxString &src, size_t n)
- size_t wxStrxfrm (wchar_t *dest, const wxString &src, size_t n)
- size_t wxStrxfrm (char *dest, const wxCStrData &src, size_t n)
- size_t wxStrxfrm (wchar_t *dest, const wxCStrData &src, size_t n)
- char * wxStrtok (char *str, const char *delim, char **saveptr)
- wchar_t * wxStrtok (wchar_t *str, const wchar_t *delim, wchar_t **saveptr)
- char * wxStrtok (char *str, const wxCStrData &delim, char **saveptr)
- wchar_t * wxStrtok (wchar_t *str, const wxCStrData &delim, wchar_t **saveptr)
- char * wxStrtok (char *str, const wxString &delim, char **saveptr)
- wchar_t * wxStrtok (wchar_t *str, const wxString &delim, wchar_t **saveptr)
- const char * wxStrstr (const char *haystack, const char *needle)

- const wchar_t * wxStrstr (const wchar_t *haystack, const wchar_t *needle)
- const char * wxStrstr (const char *haystack, const wxString &needle)
- const wchar_t * wxStrstr (const wchar_t *haystack, const wxString &needle)
- const char * wxStrstr (const wxString &haystack, const wxString &needle)
- const char * wxStrstr (const wxCStrData &haystack, const wxString &needle)
- const char * wxStrstr (const wxCStrData &haystack, const wxCStrData &needle)
- const char * wxStrstr (const wxString &haystack, const char *needle)
- const char * wxStrstr (const wxCStrData &haystack, const char *needle)
- const wchar_t * wxStrstr (const wxString &haystack, const wchar_t *needle)
- const wchar_t * wxStrstr (const wxCStrData &haystack, const wchar_t *needle)
- const char * wxStrchr (const char *s, char c)
- const wchar_t * wxStrchr (const wchar_t *s, wchar_t c)
- const char * wxStrrchr (const char *s, char c)
- const wchar_t * wxStrrchr (const wchar_t *s, wchar_t c)
- const char * wxStrchr (const char *s, const wxUniChar &c)
- const wchar_t * wxStrchr (const wchar_t *s, const wxUniChar &c)
- const char * wxStrrchr (const char *s, const wxUniChar &c)
- const wchar_t * wxStrrchr (const wchar_t *s, const wxUniChar &c)
- const char * wxStrchr (const char *s, const wxUniCharRef &c)
- const wchar_t * wxStrchr (const wchar_t *s, const wxUniCharRef &c)
- const char * wxStrrchr (const char *s, const wxUniCharRef &c)
- const wchar_t * wxStrrchr (const wchar_t *s, const wxUniCharRef &c)
- const T * wxStrchr (const wxCharTypeBuffer< T > &s, T c)
- const T * wxStrrchr (const wxCharTypeBuffer< T > &s, T c)
- const T * wxStrchr (const wxCharTypeBuffer< T > &s, const wxUniChar &c)
- const T * wxStrrchr (const wxCharTypeBuffer< T > &s, const wxUniChar &c)
- const T * wxStrchr (const wxCharTypeBuffer< T > &s, const wxUniCharRef &c)
- const T * wxStrrchr (const wxCharTypeBuffer< T > &s, const wxUniCharRef &c)
- const char * wxStrchr (const wxString &s, char c)
- const char * wxStrrchr (const wxString &s, char c)
- const char * wxStrchr (const wxString &s, int c)
- const char * wxStrrchr (const wxString &s, int c)
- const char * wxStrchr (const wxString &s, const wxUniChar &c)
- const char * wxStrrchr (const wxString &s, const wxUniChar &c)
- const char * wxStrchr (const wxString &s, const wxUniCharRef &c)
- const char * wxStrrchr (const wxString &s, const wxUniCharRef &c)
- const wchar_t * wxStrchr (const wxString &s, wchar_t c)
- const wchar_t * wxStrrchr (const wxString &s, wchar_t c)
- const char * wxStrchr (const wxCStrData &s, char c)
- const char * wxStrrchr (const wxCStrData &s, char c)
- const char * wxStrchr (const wxCStrData &s, int c)
- const char * wxStrrchr (const wxCStrData &s, int c)
- const char * wxStrchr (const wxCStrData &s, const wxUniChar &c)
- const char * wxStrrchr (const wxCStrData &s, const wxUniChar &c)
- const char * wxStrchr (const wxCStrData &s, const wxUniCharRef &c)
- const char * wxStrrchr (const wxCStrData &s, const wxUniCharRef &c)
- const wchar_t * wxStrchr (const wxCStrData &s, wchar_t c)
- const wchar_t * wxStrrchr (const wxCStrData &s, wchar_t c)
- const char * wxStrpbrk (const char *s, const char *accept)
- const wchar_t * wxStrpbrk (const wchar_t *s, const wchar_t *accept)
- const char * wxStrpbrk (const char *s, const wxString &accept)
- const char * wxStrpbrk (const char *s, const wxCStrData &accept)
- const wchar_t * wxStrpbrk (const wchar_t *s, const wxString &accept)
- const wchar_t * wxStrpbrk (const wchar_t *s, const wxCStrData &accept)
- const char * wxStrpbrk (const wxString &s, const wxString &accept)

- const char * wxStrpbrk (const wxString &s, const char *accept)
- const wchar_t * wxStrpbrk (const wxString &s, const wchar_t *accept)
- const char * wxStrpbrk (const wxString &s, const wxCStrData &accept)
- const char * wxStrpbrk (const wxCStrData &s, const wxString &accept)
- const char * wxStrpbrk (const wxCStrData &s, const char *accept)
- const wchar_t * wxStrpbrk (const wxCStrData &s, const wchar_t *accept)
- const char * wxStrpbrk (const wxCStrData &s, const wxCStrData &accept)
- const T * wxStrpbrk (const S &s, const wxCharTypeBuffer< T > &accept)
- char * wxStrstr (char *haystack, const char *needle)
- wchar_t * wxStrstr (wchar_t *haystack, const wchar_t *needle)
- char * wxStrstr (char *haystack, const wxString &needle)
- wchar_t * wxStrstr (wchar_t *haystack, const wxString &needle)
- char * wxStrchr (char *s, char c)
- char * wxStrrchr (char *s, char c)
- wchar_t * wxStrchr (wchar_t *s, wchar_t c)
- wchar_t * wxStrrchr (wchar_t *s, wchar_t c)
- char * wxStrpbrk (char *s, const char *accept)
- wchar_t * wxStrpbrk (wchar_t *s, const wchar_t *accept)
- char * wxStrpbrk (char *s, const wxString &accept)
- wchar_t * wxStrpbrk (wchar_t *s, const wxString &accept)
- FILE * wxFopen (const wxString &path, const wxString &mode)
- FILE * wxFreopen (const wxString &path, const wxString &mode, FILE *stream)
- int wxRemove (const wxString &path)
- int wxRename (const wxString &oldpath, const wxString &newpath)
- char * wxFgets (char *s, int size, FILE *stream)
- int wxFgetc (FILE *stream)
- int wxUngetc (int c, FILE *stream)
- int wxAtoi (const wxString &str)
- long wxAtol (const wxString &str)
- double wxAtof (const wxString &str)
- double wxStrtod (const char *nptr, char **endptr)
- double wxStrtod (const wchar_t *nptr, wchar_t **endptr)
- double wxStrtod (const wxCharTypeBuffer< T > &nptr, T **endptr)
- double wxStrtod (const wxString &nptr, T endptr)
- double wxStrtod (const wxCStrData &nptr, T endptr)
- int wxSystem (const wxString &str)
- char * wxGetenv (const char *name)
- wchar_t * wxGetenv (const wchar_t *name)
- char * wxGetenv (const wxString &name)
- char * wxGetenv (const wxCStrData &name)
- char * wxGetenv (const wxCharBuffer &name)
- wchar_t * wxGetenv (const wxWCharBuffer &name)
- size_t wxStrftime (char *s, size_t max, size_t max, const wxString &format, const struct tm *tm)
- size_t wxStrftime (wchar_t *s, size_t max, size_t max, const wxString &format, const struct tm *tm)
- bool wxIsalnum (const wxUniChar &c)
- bool wxIsalpha (const wxUniChar &c)
- bool wxIscntrl (const wxUniChar &c)
- bool wxIsdigit (const wxUniChar &c)
- bool wxIsgraph (const wxUniChar &c)
- bool wxIslower (const wxUniChar &c)
- bool wxIsprint (const wxUniChar &c)
- bool wxIspunct (const wxUniChar &c)
- bool wxIsspace (const wxUniChar &c)
- bool wxIsupper (const wxUniChar &c)
- bool wxIsxdigit (const wxUniChar &c)
- wxUniChar wxTolower (const wxUniChar &c)
- wxUniChar wxToupper (const wxUniChar &c)
- int wxIsctrl (const wxUniChar &c)

22.510 interface/wx/xlocale.h File Reference

Classes

- class wxXLocale

  This class represents a locale object used by so-called xlocale API.

Functions

- int wxIsalnum_l (wchar_t c, const wxXLocale &loc)
- int wxIsalpha_l (wchar_t c, const wxXLocale &loc)
- int wxIscntrl_l (wchar_t c, const wxXLocale &loc)
- int wxIsdigit_l (wchar_t c, const wxXLocale &loc)
- int wxIsgraph_l (wchar_t c, const wxXLocale &loc)
- int wxIslower_l (wchar_t c, const wxXLocale &loc)
- int wxIsprint_l (wchar_t c, const wxXLocale &loc)
- int wxIspunct_l (wchar_t c, const wxXLocale &loc)
- int wxIsspace_l (wchar_t c, const wxXLocale &loc)
- int wxIsupper_l (wchar_t c, const wxXLocale &loc)
- int wxIsxdigit_l (wchar_t c, const wxXLocale &loc)
- wchar_t wxTolower_l (wchar_t c, const wxXLocale &loc)
- wchar_t wxToupper_l (wchar_t c, const wxXLocale &loc)
- double wxStrtod_l (const wchar_t *c, wchar_t **endptr, const wxXLocale &loc)
- long wxStrtol_l (const wchar_t *c, wchar_t **endptr, int base, const wxXLocale &loc)
- unsigned long wxStrtoul_l (const wchar_t *c, wchar_t **endptr, int base, const wxXLocale &loc)

Variables

- wxXLocale wxNullXLocale

  An empty and invalid wxXLocale object.

22.510.1 Variable Documentation

wxXLocale wxNullXLocale

An empty and invalid wxXLocale object.

22.511 interface/wx/xml/xml.h File Reference

Classes

- class wxXmlNode

  Represents a node in an XML document.
- class wxXmlAttribute

  Represents a node attribute.
- class wxXmlDocument

  This class holds XML data/document as parsed by XML parser in the root node.

Macros

- #define wxXML_NO_INDENTATION (-1)

Enumerations

- enum wxXmlNodeType {
  wxXML_ELEMENT_NODE = 1,
  wxXML_ATTRIBUTE_NODE = 2,
  wxXML_TEXT_NODE = 3,
  wxXML_CDATA_SECTION_NODE = 4,
  wxXML_ENTITY_REF_NODE = 5,
  wxXML_ENTITY_NODE = 6,
  wxXML_PI_NODE = 7,
  wxXML_COMMENT_NODE = 8,
  wxXML_DOCUMENT_NODE = 9,
  wxXML_DOCUMENT_TYPE_NODE = 10,
  wxXML_DOCUMENT_FRAG_NODE = 11,
  wxXML_NOTATION_NODE = 12,
  wxXML_HTML_DOCUMENT_NODE = 13 }

  Represents XML node type.
- enum wxXmlDocumentLoadFlag {
  wxXMLDOC_NONE,
  wxXMLDOC_KEEP_WHITESPACE_NODES }

22.511.1 Macro Definition Documentation

#define wxXML_NO_INDENTATION (-1)

22.511.2 Enumeration Type Documentation

enum wxXmlDocumentLoadFlag

Enumerator

wxXMLDOC_NONE

wxXMLDOC_KEEP_WHITESPACE_NODES

enum wxXmlNodeType

Represents XML node type.

Enumerator

wxXML_ELEMENT_NODE

wxXML_ATTRIBUTE_NODE

wxXML_TEXT_NODE

wxXML_CDATA_SECTION_NODE

wxXML_ENTITY_REF_NODE

wxXML_ENTITY_NODE

wxXML_PI_NODE

wxXML_COMMENT_NODE

wxXML_DOCUMENT_NODE

wxXML_DOCUMENT_TYPE_NODE

wxXML_DOCUMENT_FRAG_NODE

wxXML_NOTATION_NODE

wxXML_HTML_DOCUMENT_NODE

22.512 interface/wx/xrc/xh_sizer.h File Reference

Classes

- class wxSizerXmlHandler

22.513 interface/wx/xrc/xmlres.h File Reference

Classes

- class wxXmlResource

  This is the main class for interacting with the XML-based resource system.
- class wxXmlResourceHandler

  wxSizerXmlHandler is a class for resource handlers capable of creating a wxSizer object from an XML node.

Enumerations

- enum wxXmlResourceFlags {
  wxXRC_USE_LOCALE = 1,
  wxXRC_NO_SUBCLASSING = 2,
  wxXRC_NO_RELOADING = 4 }

  Flags which can be used with wxXmlResource::wxXmlResource.

22.513.1 Enumeration Type Documentation

enum wxXmlResourceFlags

Flags which can be used with wxXmlResource::wxXmlResource.

Enumerator

wxXRC_USE_LOCALE Translatable strings will be translated via _().

wxXRC_NO_SUBCLASSING Subclass property of object nodes will be ignored (useful for previews in XRC editors).

wxXRC_NO_RELOADING Prevent the XRC files from being reloaded from disk in case they have been modified there since being last loaded (may slightly speed up loading them).

22.514 interface/wx/zipstrm.h File Reference

Classes

- class wxZipNotifier

  If you need to know when a wxZipInputStream updates a wxZipEntry, you can create a notifier by deriving from this abstract base class, overriding wxZipNotifier::OnEntryUpdated().
- class wxZipEntry

  Holds the meta-data for an entry in a zip.
- class wxZipInputStream

  Input stream for reading zip files.
- class wxZipClassFactory

  Class factory for the zip archive format.
- class wxZipOutputStream

  Output stream for writing zip files.

Enumerations

- enum wxZipMethod {
  wxZIP_METHOD_STORE,
  wxZIP_METHOD_SHRINK,
  wxZIP_METHOD_REDUCE1,
  wxZIP_METHOD_REDUCE2,
  wxZIP_METHOD_REDUCE3,
  wxZIP_METHOD_REDUCE4,
  wxZIP_METHOD_IMPLODE,
  wxZIP_METHOD_TOKENIZE,
  wxZIP_METHOD_DEFLATE,
  wxZIP_METHOD_DEFLATE64,
  wxZIP_METHOD_BZIP2 = 12,
  wxZIP_METHOD_DEFAULT = 0xffff }

  Compression Method, only 0 (store) and 8 (deflate) are supported here.
- enum wxZipSystem {
  wxZIP_SYSTEM_MSDOS,
  wxZIP_SYSTEM_AMIGA,
  wxZIP_SYSTEM_OPENVMS,
  wxZIP_SYSTEM_UNIX,
  wxZIP_SYSTEM_VM_CMS,
  wxZIP_SYSTEM_ATARI_ST,
  wxZIP_SYSTEM_OS2_HPFS,
  wxZIP_SYSTEM_MACINTOSH,
  wxZIP_SYSTEM_Z_SYSTEM,
  wxZIP_SYSTEM_CPM,
  wxZIP_SYSTEM_WINDOWS_NTFS,
  wxZIP_SYSTEM_MVS,
  wxZIP_SYSTEM_VSE,
  wxZIP_SYSTEM_ACORN_RISC,
  wxZIP_SYSTEM_VFAT,
  wxZIP_SYSTEM_ALTERNATE_MVS,
  wxZIP_SYSTEM_BEOS,
  wxZIP_SYSTEM_TANDEM,
  wxZIP_SYSTEM_OS_400 }

  Originating File-System.
- enum wxZipAttributes {
  wxZIP_A_RDONLY = 0x01,
  wxZIP_A_HIDDEN = 0x02,
  wxZIP_A_SYSTEM = 0x04,
  wxZIP_A_SUBDIR = 0x10,
  wxZIP_A_ARCH = 0x20,
  wxZIP_A_MASK = 0x37 }

  Dos/Win file attributes.
- enum wxZipFlags {
  wxZIP_ENCRYPTED = 0x0001,
  wxZIP_DEFLATE_NORMAL = 0x0000,
  wxZIP_DEFLATE_EXTRA = 0x0002,
  wxZIP_DEFLATE_FAST = 0x0004,
  wxZIP_DEFLATE_SUPERFAST = 0x0006,
  wxZIP_DEFLATE_MASK = 0x0006,
  wxZIP_SUMS_FOLLOW = 0x0008,
  wxZIP_ENHANCED = 0x0010,
  wxZIP_PATCH = 0x0020,
  wxZIP_STRONG_ENC = 0x0040,
  wxZIP_UNUSED = 0x0F80,
  wxZIP_RESERVED = 0xF000 }

Values for the flags field in the zip headers.

22.514.1 Enumeration Type Documentation

enum wxZipAttributes

Dos/Win file attributes.

Enumerator

wxZIP_A_RDONLY

wxZIP_A_HIDDEN

wxZIP_A_SYSTEM

wxZIP_A_SUBDIR

wxZIP_A_ARCH

wxZIP_A_MASK

enum wxZipFlags

Values for the flags field in the zip headers.

Enumerator

wxZIP_ENCRYPTED

wxZIP_DEFLATE_NORMAL

wxZIP_DEFLATE_EXTRA

wxZIP_DEFLATE_FAST

wxZIP_DEFLATE_SUPERFAST

wxZIP_DEFLATE_MASK

wxZIP_SUMS_FOLLOW

wxZIP_ENHANCED

wxZIP_PATCH

wxZIP_STRONG_ENC

wxZIP_UNUSED

wxZIP_RESERVED

enum wxZipMethod

Compression Method, only 0 (store) and 8 (deflate) are supported here.

Enumerator

wxZIP_METHOD_STORE

wxZIP_METHOD_SHRINK

wxZIP_METHOD_REDUCE1

wxZIP_METHOD_REDUCE2

wxZIP_METHOD_REDUCE3

wxZIP_METHOD_REDUCE4

wxZIP_METHOD_IMPLODE

wxZIP_METHOD_TOKENIZE

wxZIP_METHOD_DEFLATE

wxZIP_METHOD_DEFLATE64

wxZIP_METHOD_BZIP2

wxZIP_METHOD_DEFAULT

enum wxZipSystem

Originating File-System.

These are Pkware's values. Note that Info-zip disagree on some of them, most notably NTFS.

Enumerator

wxZIP_SYSTEM_MSDOS

wxZIP_SYSTEM_AMIGA

wxZIP_SYSTEM_OPENVMS

wxZIP_SYSTEM_UNIX

wxZIP_SYSTEM_VM_CMS

wxZIP_SYSTEM_ATARI_ST

wxZIP_SYSTEM_OS2_HPFS

wxZIP_SYSTEM_MACINTOSH

wxZIP_SYSTEM_Z_SYSTEM

wxZIP_SYSTEM_CPM

wxZIP_SYSTEM_WINDOWS_NTFS

wxZIP_SYSTEM_MVS

wxZIP_SYSTEM_VSE

wxZIP_SYSTEM_ACORN_RISC

wxZIP_SYSTEM_VFAT

wxZIP_SYSTEM_ALTERNATE_MVS

wxZIP_SYSTEM_BEOS

wxZIP_SYSTEM_TANDEM

wxZIP_SYSTEM_OS_400

22.515 interface/wx/zstream.h File Reference

Classes

- class wxZlibOutputStream

  This stream compresses all data written to it.

- class wxZlibInputStream

  This filter stream decompresses a stream that is in zlib or gzip format.

Enumerations

- enum wxZlibCompressionLevels {
  wxZ_DEFAULT_COMPRESSION = -1,
  wxZ_NO_COMPRESSION = 0,
  wxZ_BEST_SPEED = 1,
  wxZ_BEST_COMPRESSION = 9 }

  Compression level.
- enum wxZLibFlags {
  wxZLIB_NO_HEADER = 0,
  wxZLIB_ZLIB = 1,
  wxZLIB_GZIP = 2,
  wxZLIB_AUTO = 3 }

  Flags.

22.515.1 Enumeration Type Documentation

enum wxZlibCompressionLevels

Compression level.

Enumerator

wxZ_DEFAULT_COMPRESSION

wxZ_NO_COMPRESSION

wxZ_BEST_SPEED

wxZ_BEST_COMPRESSION

enum wxZLibFlags

Flags.

Enumerator

wxZLIB_NO_HEADER raw deflate stream, no header or checksum

wxZLIB_ZLIB zlib header and checksum

wxZLIB_GZIP gzip header and checksum, requires zlib 1.2.1+

wxZLIB_AUTO autodetect header zlib or gzip